Inspecting and Repairing Notefiles

Xerox Corporation

Randy Trigg

[First written: 8/9/85 Randy Trigg]
[Last updated: 8/26/85 Randy Trigg]

This document describes the Notefile inspector facility available via the Inspect&Repair option on the Notefile Ops menu in NoteCards Release1.2i.

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.

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 begins with a brief discussion of the organization of a notefile. Then follows sections describing each of the three phases. Finally, I outline some tips, strategies and pitfalls to watch for.

1. What you need to know about a notefile's innards.

1.1 Notefile structure.

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 has 5 parts, a status field and 4 pointers. The status field specifies whether the index entry is free or occupied by an active or deleted card. There is one pointer in the index entry to each of the 4 parts of a card: substance, title, links and property list. These point into the data area. 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. To throw away the old versions of card parts, it is necessary to compact the notefile.

1.2 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. Note that these boxes cannot be deleted. The IDs from NC00004 through NC00020 are unused. Currently, an old ID is never reused, even if its card is deleted and the notefile is compacted. Thus, 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.

1.3 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. If you have not been attaching properties to your cards, then the inspector will only show those properties that the system maintains. Currently the only such property is "Updates" with value equal to a list of dates on which the card was updated going chronologically backwards from the front of the list to the back.

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 a text substance on the notefile looks like the way TEdit writes out text streams (text followed by "looks" information).

1.4 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.

1.5 The links rebuilder.

The third phase of the inspector rebuilds the links of a notefile as follows: First it 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 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.

The inspector provides the option of having the links rebuilder phase rebuild bad filebox substances, bad link types, and bad global links. See Section ?? below.

2. 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. I recommend 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. You can monitor the progress of phase 1 by watching the prompt window. It will be printing messages like "Processing byte xxxxx of yyyyy."

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 are free and will never be reused. See Section 1.2 above.) If all seems well with the world, the next line will read "All active cards look okay." If not, there will be various messages outlining the problems. (See Figure 1.)

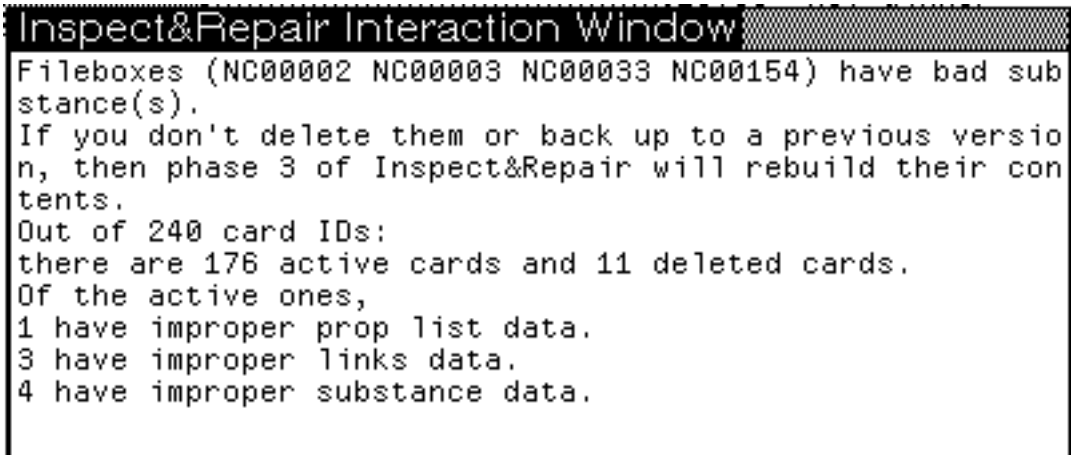


Figure 1: Snapshot of a sample interaction window

Note that several fileboxes have bad substances and that a special message is printed on their behalf. This indicates that if you don't wish to delete or back these up to a previous version, then phase 3 will rebuild them. (See Section 1.5 above.)

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 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. The particular options you get in that menu depend on the state of your notefile and are described below. The first two 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.

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.)

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.

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.

CONTINUE INSPECT&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 1.5 above). Selecting it causes Inspect&Repair to move to phase 3 and rebuild your notefile's links.

3. 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.

Attached to the upper right corner of the cards inspector menu is a menu containing at least the two options: ABORT and DONE. 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 swapped with 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. (This is equivalent to choosing ABORT from the inspector window as described in Section 2.)

Choosing DONE from this attached menu indicates that you're done tinkering with card parts and wish to return to the main interaction window. Normally, this causes the card inspector

menu to close and the phase 1 process outlined in Section 2 to be performed again. Thus the data area will be rescoured and new statistics on the health of your notefile will be printed. This cycle of scour data area (phase 1) followed by inspect (phase 2) can be repeated as often as desired. Eventually, you must either abort, end the Inspect&Repair process gracefully, or continue to phase 3. Because phase 1 can be quite slow for large uncompacted notefiles, there is one optimization: if you've made no changes in phase 2, then the data area scouring is not repeated in phase 1. Rather, the old information from the last scouring is recovered and used instead.

If you've clicked DONE, but there are still cards with bad prop lists, titles, or links (because you haven't either deleted them or backed up their card parts to previous versions), you will be asked up to three questions. The list of card IDs of cards with bad prop lists is printed out and you're asked whether it's okay to set the prop lists of those cards to NIL. Then, the list of card IDs of cards with bad titles is printed out and you're asked whether it's okay to set the titles of those cards to "Untitled." Finally, card IDs for cards with bad links are printed out and you're asked whether it's okay to set the global links to nil. (Global links will be rebuilt as much as possible in phase 3. See Sections 1.5 and 4.) If you want phase 3 to be able to run, it is necessary to either answer yes to these questions or fix each of the bad card parts by hand in phase 2.

3.1 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.)

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.

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.

3.2 The card parts inspector

Figure 2 below shows an example of a card parts inspector. It is composed of four attached menus arranged vertically and one attached operations menu atop the stack.

Figure 2: A card parts inspector

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 we began recording card parts write dates.)

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, effecting 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 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. 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.

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 NCFile 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 affect 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, thus backing up the card to the selected version. (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.)

4. Running the Notefile inspector: Phase 3: Rebuilding your links

To complete the Inspect&Repair process, select the **Continue Inspect&Repair** option from the interaction window menu. This invokes phase 3, the links rebuilder. Normally, this simply rebuilds links from card substances (see Section 1.5). 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.) Again, for details, see Section 1.5.

5. 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 inadvertantly deleted. Or back up a card that you inadvertantly changed to a previous version.

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 ID and close the notefile. Then, run Inspect&Repair, find the card ID in the inspector menu 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 Sections 1.5 and 4), and others that it can't (e.g. bad title). 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).

NoteCards Programmer's Interface

Release 1.2i

Randy Trigg

Xerox PARC

Updated version: 18-Mar-85

Modified: 26-Aug-85 by Randy Trigg

Modified: 1-Aug-85 by Lissa Monty

Introduction

This document 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

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 release1.1 format without user confirmation.

(NCP.CloseNoteFile [<stream>])

Closes <stream> if it is 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.

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.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.

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.CreateNoteCard.

(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 (layed out as a forest, sacrificing screen space for speed), COMPACT (layed out as a forest, using minimal screen space), LATTICE (layed out as a directed acyclic graph, the default), *GRAPH* (layed 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 layed on its side or up and down. The third element, if present, should be the atom REVERSE. This indicates that horizontal graphs should be layed out from right to left instead of left to right and that vertical graphs should be layed 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 litatoms 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.

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 synomym 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.

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 diplay 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.

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.

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 the Notecards user's manual 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

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 litatoms {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.

Notice of release of Notecards 1.2i

The 1.2i Intermezzo release of Notecards is hereby officially released.

To run NoteCards, load onto an Intermezzo sysout the file {qv}<notecards>release1.2i>notecards.dcom. As usual, send bug reports by choosing "NoteCards Report" from the Lafite middle button send mail menu. Mail of more general interest to the NoteCards community should be sent to NoteCards↑.pa.

Even if you have been using 1.2i for some time and feel comfortable with Notecards, please take a look at the release notes in {qv}<notecards>release1.2i>doc>ReleaseNotes.ted.

The release notes describe in detail the changes since 1.1. These include incorporation of the latest version of sketch, new functionality in the browser, a new notefile inspect and repair facility, several new library packages, and many other feature additions and bug fixes.

Also on {qv}<notecards>release1.2i>doc> you can find updated documentation on the programmer's interface, ProgIntFace.ted, and a new document describing the inspect and repair facility, NoteFileInspector.ted.

The library packages and accompanying documentation can be found in {qv}<notecards>release1.2i>library>.

Enjoy!

- Randy

NoteCards Release1.2i Announcement

Xerox Corporation

Randy Trigg
Frank Halasz

[Location: {qv}<notecards>release1.2i>notecards.dcom]
[First written: 3/27/85 Randy Trigg]
[Last updated: 8/26/85 Randy Trigg]

This document updates the NoteCards Release1.1 User's Manual, describing changes and new features for Release1.2i. As usual, send bug reports to NoteCardsSupport.pa (or use the Lafite SendMail middle button menu) and matters of more global interest to Notecards↑.pa.

You must be in Intermezzo to run NoteCards Release1.2i. From now on, you can depend on the letter suffix following the release number to indicate the appropriate version of Interlisp.

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.

1. Operating on a Notefile.

Checkpointing and aborting a session:

A fundamental change was made to the way Notecards updates its working notefile that allows 1.2i users to checkpoint their work, abort a session (losing work since the last checkpoint), and recover more gracefully from crashes. First, a word about the way Notecards notefiles are structured.

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 Release1.1 (and earlier), the index modifications happened out on the file as they occured. Now, in Release1.2i they happen in an in-core array and are not written to the file till 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).

More concretely, there are now two new NoteFile Ops menu entries: "Checkpoint Session" and "Abort Session." Checkpointing causes any active cards to have their contents saved to the notefile (but not closed), the index array to be written back out to the file, and the checkpoint pointer to be reset to the end of the file. (Note that closing a notefile automatically does a CheckpointSession.) Aborting a session causes Notecards to close down, 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 you're asked whether the file should be truncated. You're also given the option of saving the extra work since the last checkpoint to a file. If valuable cards were created (or modifications made) since the last checkpoint, then you should answer yes and provide the name of a file in which to store the truncated information.

Next, 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.]

Compacting a 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 Release1.2i 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.

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 for multiple users sharing a notefile. Still in the experimental stages, it must be called via the programmer's interface. See the programmer's interface documentation.

Inspecting and healing broken notefiles:

The old Repair option on the Notefile Ops menu is now called Inspect&Repair 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, see the document titled NotefileInspector.ted.

2. Changes to the Notecards user interface.

Stylesheets:

Several places in Notecards now use Tayloe Stansbury's 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. 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" causes the stylesheet to be exited 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 top level global parameters menu has several new additions. These (as well as some old 1.1 ones) are described below. To change the value of a global parameter, click on the variable name. The value will toggle between "Yes" and "No" if binary, and allow selection from an appropriate menu otherwise.

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."

CloseCardsOffScreen: If "Yes," then when a card is closed, it is first dragged off screen so that the close happens invisibly.

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. 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.

AlphabetizedFileBoxChildren: If "Yes," then new fileboxes will have the property OrderingFn set to NC.IDAlphOrder. 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 4.

DefaultLinkIconAttachBitmap, DefaultLinkIconShowTitle, DefaultLinkIconShowLinkType: These 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 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 and selecting from the three menus in the resulting stylesheet. These are: AttachBitmap, ShowTitle, and ShowLinkType. AttachBitmap, if "Yes," causes link icons to be shown with a bitmap representing the type of the destination card attached at the left. ShowTitle and ShowLinkType, if "Yes," 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. (See description of global parameters above.) If all fields are set to "No" (or the floating ones inherit No from the global parameters), then a small, uninformative icon is used to display the link.

"Pushing" and "Pulling" link icons:

There are now two ways to move or copy a link icon between cards or within a card. "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. The new style is called "pushing" and is done by holding down the shift key while left clicking in the middle part of 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.

3. Changes to the Notecards browser.

Multiple roots:

Browsers can now contain multiple roots, in which case the graph will be laid out as a forest.

Dashed links:

Dashed browser links was a rarely used option in Release1.1, largely because of speed considerations. The speed of drawing dashed links has improved in Release1.2i 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 2).

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-settable parameter in the GlobalParameters

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).

Browser specs:

Whereas in Release1.1 only the link types to traverse could be specified, in Release1.2i, 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. For general details on the stylesheet interface see Section 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.

Forward and backward link types function as in Release1.1. 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."

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 default is LATTICE.

Browser orientation is one of Horizontal, Vertical, Reverse/Horizontal, or Reverse/Vertical. These specify whether the graph is layed 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 now RecomputeBrowser, RelayoutGraph, ReconnectNodes, UnconnectNodes, ExpandBrowserNode, GraphEditMenu, and ChangeBrowserSpecs.

RecomputeBrowser causes the current contents of the browser to be thrown away and recomputed as in Release1.1. However, in Release1.2i, you can optionally specify a new set of root nodes.

RelayoutGraph 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.

ReconnectNodes 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 for several reasons:

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.

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

ExpandBrowserNode 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.

GraphEditMenu brings up the graph editing menu. See the description below.

ChangeBrowserSpecs 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":

The browser can be edited through the use of the GraphEditMenu. This menu can be obtained either by right-buttoning in the browser window or by choosing GraphEditMenu from the title bar middle button menu. The GraphEditMenu includes options for "structure editing"; that is, changing underlying NoteCards structure by editing the browser. The old options for editing without changing structure are also present. Given below are the menu items in GraphEditMenu and the actions they engender.

CreateCard&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.

CreateLink&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.

DeleteCard&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.

DeleteLink&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.

AddLabel 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.)

AddNode 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.

AddEdge 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 Section ??) 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.

RemoveNode 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.

RemoveEdge 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.

MoveNode 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.

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

LabelLarger 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.]

4. Miscellaneous changes.

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 "InsertLinks" 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:

This is now a normal entry in the left button title bar menu of a card (rather than a subentry under Edit Properties). The format of the ShowLinks 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 NCFile cards.)

Data saved at card closing:

When a card is closed, only those parts that are dirty are written out to the notefile. A message indicating which parts are being saved is now printed to the card's prompt window during closing. Furthermore, certain card types (in particular, browsers) were saving their substance even if no changes were made. This source of space inefficiency has been fixed in Release1.2i.

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 2.

Programmer's interface:

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

Notecards system date:

You can find the date of your Notecards system in the variable NC.SystemDate. The 'NewestFile property on the NC.SystemDate atom contains the name of the last modified Notecards file.

The Notecards library packages:

The old Release1.1 library packages have been converted to 1.2i and documented and several new ones have been added. These can be found on {qv}<notecards>release1.2>library> and include NCScreen, NCCluster, NCChain, NCFileCard, NCKeys, NCHacks, and ARIDemo. Documentation can be found in <filename>.ted.

NCScreen defines several handy functions for arranging cards on the screen callable from the programmer's interface. NCCluster defines several new card types, most notably CaseCluster, a cluster of cards for use in the sample domain of legal case analysis. NCChain defines the Chain card type, useful for breaking up a large text card into a linked chain of cards. NCFileCard defines the new File card type and FILE substance allowing a notefile to link to external files via standard Notecards links. NCKeys provides a shorthand language for invoking various handy programmer's interface functions. NCHacks contains several handy functions written using the programmer's interface. Two of these allow global text searches and replaces throughout a notefile. In addition there is a function that searches by last card modification date and one that links cards to form chains. Finally, ARIDemo is an example of how the programmer's interface can be used to construct notefiles that demo themselves.

Loading NoteCards from different directories:

NoteCards now uses the values of four directories variables to decide from whence to load the code. These are NOTECARDSDIRECTORIES, NOTECARDSMAPDIRECTORIES, QUADTREEDIRECTORY, and MAPFILEDIRECTORY. They default to ({QV}<NOTECARDS>RELEASE1.2I>), ({QV}<NOTECARDS>MAPS>NEW>), {QV}<NOTECARDS>MAPS>, and {QV}<NOTECARDS>MAPS> respectively.

5. Known bugs and plans for future improvements:

o The compactor should check first for available space.

o There are major speed problems in redrawing large browsers. Changing link display mode could also use some streamlining.

o Integrate the document compiler and the types mechanism so that instances of new card types can be sucked into TEdit documents.

o Make links into full-fledged objects having properties and type hierarchies.

The NoteCards Types Mechanism

Release 1.2

Frank G. Halasz

Xerox PARC

First Written: 22-Mar-85

Modified: 26-Mar-85 by Frank G. Halasz

Modified: 1-Aug-85 by Lissa Monty

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.

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))))
```

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, deleteing, 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.

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.

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 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 non-NIL, then the card is to be displayed in a window on the screen. If NoDisplayFlg is 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 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 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 memeber 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 memeber 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 specififed 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 in 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 in 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.

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.

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.

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)
                                (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)
          (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)))
```

NoteCards
Evolving Outline

Chapter Status

##########

4 TEdit Manual

##########

4 Sketch Manual

##########

2 Programmers Interface to NoteCards

##########

Status Codes:

0 Nothing done.
1 Chapter started.
2 Initial draft written, needs major cleanup.
3 Needs minor cleanup.
4 Ready for initial release.
5 Ready for final release.

ENVOS NOTECARDS™
USER'S GUIDE

Release 1.1
300300
March, 1989

Address comments to:
Envos Corporation
User Documentation
1157 San Antonio Rd.
Mountain View, CA 94043
415-966-6200

ENVOS NOTECARDS™ USER'S GUIDE

Release 1.1

Part Number 300300

March, 1989

Text was written and produced with Envos text formatting tools; Xerox printers were used to produce text masters. The typeface is Modern.

1. INTRODUCTION

NoteCards

NoteCards is a computer environment designed to help people work with ideas. Its users are authors, researchers, designers, and other intellectual laborers engaged in analyzing information, constructing models, formulating arguments, designing artifacts, and generally processing ideas. The system provides these users with a variety of tools for collecting, representing, managing, interrelating, and communicating ideas. NoteCards is based on the notion that creative intellectual work is a hand-craft, a uniquely human skill that cannot be easily automated.

Notecards provides the user with a "semantic network" of electronic notecards interrconnected by typed links. This network serves as a medium in which the user can represent collections of related ideas. It also functions as a structure for organizing, storing, and retrieving information. The system provides the user with tools for displaying, modifying, manipulating, and navigating through this network.

From: *NoteCards in a Nutshell*, by Frank G. Halasz, Thomas P. Moran, Randall H. Trigg of the Intelligent Systems Laboratory in the Xerox Palo Alto Research Center, Preceedings of the ACM CHI+GI ’87 Conference, Toronto, Canada April 1987

System Overview

Functionally, the NoteCards system for the Sun Workstation consists of the following parts:

emulator — A SunOS executable program which executes the NoteCards system contained in the sysout and provides access to the Sun host’s hardware.

sysout — A virtual memory image (the *sysout*) containing both the NoteCards program and its data structures. The sysout provided can be used both on the Sun Workstations and on the Xerox 1100 series workstations.

notefiles — A file containing your cards on a particular topic and all their links. This is your data and its organizational structure.

fonts — Data describing the "looks" of printed characters used by NoteCards’ graphics, windowing, and hardcopying subsystems. Font directories are in four groups, display fonts, PostScript printer fonts, Interpress printer fonts, and Press printer fonts.

Useful SunOS and UNIX Conventions

SunOS is Sun Microsystems' version of the UNIX operating system. For users unfamiliar with the Sun Workstation, the following SunOS (and UNIX) conventions, are used in the manual.

For complete information on UNIX and SunOS, refer to your Sun documentation set.

case, filenames — Type-in to UNIX is case sensitive. Typically, input is in lower case. When UNIX searches for a name, it is case sensitive; it distinguishes between lower and upper case characters. By convention, most names are lower case characters.

shell — Command interpreter; the commands shown are in the C-Shell, unless otherwise noted.

NoteCards Device Conventions

NoteCards allows users to interact with SunOS file systems (including file systems mounted from other machines) by using host device names. The device names are

{DSK} — A host name which gives you access to the SunOS file system using Xerox workstation local disk conventions.

{UNIX} — A host name which gives you access to the file system using normal SunOS conventions.

The {DSK} device name provides an interface to the Sun Workstation for users who want to maintain compatibility with existing development tools and applications originally developed on a Xerox workstation. The {UNIX} device name provides a way for new applications to interact naturally with UNIX. Chapter 4, System Use Issues, explains, in greater detail, some important exceptions and restrictions to the {DSK} and {UNIX} device name.

Stylistic Conventions

Text marked by a revision bar in the right margin contains information that was added or modified since the last release.

Prompts

All examples which include SunOS dialogues use the following conventions for the SunOS prompt:

A number sign (#), as part of the system prompt, indicates that the user is logged on as root or is running su; e.g.,

```
prompt #
```

A percentage sign (%), as part of the system prompt, indicates that a user other than root is logged on; e.g.,

prompt %

Font Usage

`Bold text in TITAN font` indicates text you should type in exactly as printed.

`Regular text in TITAN font` indicates what the system prints on your workstation screen. UNIX files and programs are shown in TITAN FONT.

`Italic text in Titan font` indicates variables or parameters that you should replace with the appropriate word or string.

Bold text in Modern font is used to indicate menu commands and NoteCards parameters.

Italic text in Modern font is used to indicate emphasis.

Quote marks are used to indicate window titles, parameter values, and when refering to the names of section headings within chapters; e.g., The "Stylistic Conventions" section in Chapter 1, Introduction.

Keyboard Conventions

Keys that you press are in uppercase (e.g., COPY, for the Copy key). A carriage return is displayed as <RETURN>. Instructions that ask you to press two or three keys simultaneously are indicated as follows:

"Press CONTROL-E"

Note that on some Xerox machines, the CONTROL key is labeled PROPS or EDIT, but has the same function as the CONTROL key.

[This page intentionally left blank]

PREFACE

Welcome to NoteCards, an idea manipulation tool for the modern intellectual. NoteCards provides you with a new way to organize and manage your ideas on your computer desk top.

This guide explains in detail how to install NoteCards on your Sun 3 or Sun 4 workstation, provides a basic tutorial for the NoteCards environment, and contains the reference manual for NoteCards.

Audience

The *Envos NoteCards Users's Guide* was written for users of NoteCards on Sun workstations. The *Guide* assumes that you are already familiar with UNIX and SunOS concepts.

Section I of this manual, Introduction and Installation, should be read by the System Administrator or person installing the NoteCards system. The NoteCards user should also read this section, in particular, Chapters 1 and 4, Introduction and System Use Issues. The user who wishes to remain safely ignorant of system-level aspects of UNIX and NoteCards can skip over Chapters 2 and 3, System Requirements and Software Installation. (The system installer should also read Appendices C and D.)

Section II, NoteCards Tutorial, provides an introduction to NoteCards and the system which underlies it. Chapter 5, NoteCards Basics, is the place for all new users of NoteCards to start once they have their system up and running.

Section III, Reference Manual, is a compilation of all the necessary background information on NoteCards new and old users will need to learn to fully exploit the NoteCards system.

The final section contains the Appendices, Glossary, and Index for the *Envos NoteCards User's Guide*.

Included with the NoteCards Guide are the manuals for TEdit and Sketch, *A User's Guide to TEdit* and *A User's Guide to Sketch*.

What's in the Manual

Here is what you will find in this manual, chapter by chapter, accompanied by a brief description of when you will want to read each chapter.

Chapter 1, Introduction, gives a brief introduction to and some background on NoteCards as well as explaining the stylistic conventions used in the *Guide*. Read this chapter before you install NoteCards.

Chapter 2, System Requirements, goes over the hardware requirements of the system. Read this chapter before you install NoteCards on a particular machine to verify that it will run acceptably on that machine.

Chapter 3, Software Installation, gives you step by step directions for installing NoteCards. Read this chapter before and during system installation.

Chapter 4, System Use Issues, discusses some of the basic ins and outs of using the system, Initialization, Starting, and Exiting NoteCards, as well as file access. Read this chapter after you have installed NoteCards. In particular, read the sections "Starting NoteCards" and "Exiting NoteCards and Saving State" when you are modifying your .cshrc and .login UNIX files for ideas on how to simplify the NoteCards booting process. In order to allow it to run on two different file systems NoteCards differs slightly from the standard SunOS and UNIX file systems. Read the section "Using SunOS Files from NoteCards" when you are learning how to access files from NoteCards.

Chapter 5, NoteCards Basics, takes you by the hand and leads you step by step through the basics of using the NoteCards system. This is the place for new users to start once you have NoteCards up and running. Once you get going, this is also a good chapter to review as it brings together ideas scattered through the users's guides whose importance many not be obvious from their context in the user's guides.

Chapter 6, Building NoteCards Structures, tries to give you some insights on how to use the NoteCards sytems to its best advantage. NoteCards has been around as a PARC research prototype for several years. In this chapter we try to present some of what they have learned about using this system. Read this chapter after you have become somewhat familiar with NoteCards.

Chapter 7, The User Interface, covers some of the more basic aspects of using the system, the mouse, the keyboard, menus, etc. Read this chapter to understand more of the lower level capabilities of the system.

Chapter 8, Links, describes the link icon in all of its manifestations. Links and their physical representation, link icons, live at the core of NoteCards. Read and understand this chapter as soon as possible to make the most effective use of the system.

Chapter 9, Cards and Banners, discusses the user interface to individual notefiles and those aspects that all cards have in common. This is also an important chapter. The section "The Card Menu" is particularly important as it explains the functionality all cards have in common.

Chapter 10, User Cards, discusses cards where you are responsible for creating the contents. Read the sections "Text Cards" and "FileBox Cards" right away. The "Text Cards" section points you to the TEdit manual *A User's Guide to TEdit*. As TEdit forms the core of the system. We suggest that you make an effort to gradually learn more about TEdit all the time. Read the remaining sections as you explore the other card types. Note that the Bit Map Editor is discussed at the end of this chapter even though there is no bit-map card type

Chapter 11, System Cards, covers those cards where the system creates the contents for you. The section "Browser Cards" is the

most important one to read for a beginner. The others can be read as you need them.

Chapter 12, The MenuBox Icon, discusses the functionality on the three menus available from the MenuBox. The important commands for a new users are on the "Notefile Ops" menu. When just getting used to NoteCards, focus on the commands: **Open**, **Checkpoint**, **Close**, and **Abort**.

Chapter 13, System Parameters, explains how to change the global defaults for the system. New users can safely ingore this chapter. Most users will find that the default settings will suit all their needs.

Chapter 14, The FileBrower, explains how to use the FileBrowser.

Chapter 15, Other Tools, explains how to use the Directory Connector, and the clocks.

Chapter 16, Printing, covers setting up print capabilities and printing to PostScript and Interpress printers. Read this chapter when you have the machine on a network and are ready to set up your print capability.

Chapter 17, Known Problems, Error Conditions and Recovery, covers those problems we know about as of this writing and how to cope with and avoid them. A quick reading of the sections "Known Problems" and "Break Windows" are all that most users will ever need.

Appendix A, Notefile Concepts, explains the inner workings and structure of notefiles. Read this if you want to understand how information is stored in the notefile and why you have to perform operations like compacting occasionally. This is essential background material if you are about to inspect or fix a damaged notefile.

Appendix B, Notefile Inspector, explains how to fix notefiles damaged by power failured, broken net connections and other unforeseeable calamities. Read this before you attempt to fix a damaged notefile.

Appendix C, Initialization Files, collects a lot of information on how to write a file which will automatically set site specific values for parameters like printer names and font locations. Read this appendix when you are first setting up your system.

Appendix D, Checksum Control, covers what to do if you believe the files loaded from the distribution tape are damaged.

Acknowledgements

NoteCards has a long history. Its gestation began in October 1982 when the U.S. Government funded XSIS (Xerox Special Information Systems) and Xerox PARC (Palo Alto Research Center) to prototype a Problem-Structuring-Aids system to allow users to build "semantic networks of textual information." The first successful prototype was completed by Frank Halasz, at PARC, in

November 1983. The first real user of the NoteCards system was Ken Allen, a history graduate student at Stanford University, in the spring of 1984. The birth came in June 1984 when NoteCards was released to the government, in fulfillment of the Problem-Structuring-Aids contract. Since then, XSIS and PARC have released three unsupported versions of NoteCards on Xerox D-machines. In November 1988, the Envos NoteCards product team started work on a supported, productized version of NoteCards. Envos NoteCards Release 1.1 for the Sun Workstation was completed in April 1989.

NoteCards is based on the work of many people.

From Xerox, Frank Halasz, Tomas Moran, Randall Trigg, Richard Burton, Ronald Kaplan, Peggy Irish, Catherine Marshall, and many others.

From Envos, Robert Krivacic, Keith Mountford, Craig Sweat, Karin Sye, Daniel Sagalowicz, Larry Harada, John Sybalsky, and others.

[This page intentionally left blank]

TABLE of CONTENTS

Appendices, Glossary, and Index

[This page intentionally left blank]

6. BUILDING NOTECARDS STRUCTURES

NoteCards has been around for some time and people have developed different ways of using it. In this chapter we have recorded some of the different approaches people have taken towards using NoteCards in the hopes that these notes will help you discover the best way of using NoteCards for you.

If you have ideas about ways of using NoteCards which you would like to share with others please send them to us.

Introduction

By its very design, NoteCards encourages you to use it in specific ways. However, some users have found that they make the best use of NoteCards by not following the intended path but by branching off and forcing NoteCards to conform to their own idea of what a hypertext system should look like.

A Standard View of Building NoteCards Structures

NoteCards was intented to support two parallel modes of organization, a hierarchial one, implemented with FileBoxes, and a relational one, implemented with links.

One possible scenario is that you would collect a body of material and break it up into logical bits which would then be placed into Text, Sketch, Graph, and other card types. Ideally, the material would be scanned in, or collected from electronic sources. You would then build a hierarchial structure with FileBoxes inserting the data cards in their appropriate place in the FileBox hierarchy. Once you had established the structure you would then read through the document building links between related ideas, and constructing other non-hierarchial structures keeping these several lines of thought apart by using different link types. The different lines of thought in the material could then be collected by using document cards to follow the different link types.

This is a stereotypical view of how you might generate a structure in NoteCards however, it is not the only view.

A Non-Standard View of Building NoteCards Structures

FileBoxes and Hierarchies

One frequent comment from long time users is that people have a tendency to use FileBoxes too soon. They try to organize their data into a structure before the best structure is apparent and wind up reorganizing their ideas, wasting much time and energy. These users suggest that you build your structures only after you have worked with your data for awhile.

Other users simply do not use FileBoxes at all or use them to build only very shallow structures. Note that you could use links directly to build the equivalent hierarchial structure as with FileBoxes.

The Flat System

In the flat system, all cards are filed in the "Table of Contents", the "To Be Filed" or some user-specified FileBox card. These users may specify one or several start points for the reader, but aside from that the system is flat.

The Almost Flat System

In this organization the cards are grouped into large bunches which can be based on source, topic or some other broad category. One FileBox is used for each source, topic, or whatever the divisions are. One user likens his use of FileBoxes to library shelves where each FileBox is a different shelf. He then uses Sketch cards as an organization and integration tool, to help him navigate through his system.

Link Types

The system is built with the idea that you will classify each link type so that later you can build documents or browsers based on link types. However, some users make all their link types "Unspecified."

One of the dangers with links is that new users tend to link everything to everything else and end up with spaghetti.

Multiple Users

NoteCards was not designed with collaboration in mind. One way users have gotten around the problem of tracking what each user has done is to assign each collaborator a different font. Each contributor’s work can then be recognized by font face.

Cards and Information Chunk Size

Because cards come up on the screen a particular size, able to display a small amount of information, the perception grows among new users that the information chunks have to be small. It is important to recognize that cards can hold any amount of information, and that they can be scrolled or resized to display all the information they hold.

There are trade offs, however. If you work with large cards, the size of your notefile grows much more rapidly. This is not serious. It only means that you need to compact your notefile more frequently than people who work with smaller information chunks do.

A second trade off is that links always point to the beginning of a card never into the body of a card. If your cards are very large, it may not be immediately clear what the relationship is between the source card and the destination card.

Types of Structures People Have Built with NoteCards

One user working on a design project used a notefile structured with design goals as roots moving down to the component level at leaf nodes. This structure allowed him to see the design history, subgoals, and goals which were retracted.

Other users use NoteCards to maintain indices of where to find sources of information. This is fairly common. Users frequently implement on-line rolodexes, customer-support or customer-tracking notefiles, and other types of text databases, such as project tracking.

The most common use is in writing large reports. NoteCards is used to collect, organize, and cross-reference information until the writer has a firm grasp of the material and how various pieces of it interact.

Specializing NoteCards

It is possible to specialize NoteCards to specific tasks.

One user has created an interface to the Lexis legal database. To do this he created a new text-card type which receives retrieved data from Lexis. He is also working on an interface to videodisk and other sources.

At Xerox PARC there is a group working on an Instructional Design Environment for designing, developing, and presenting instruction. These people are extensively modifying the NoteCards environment

If you have a need for a particular extension to NoteCards there is a programmers interface which you or your company's developers can use to extend the NoteCards environment, or Envos Consulting Services group can provide you with specalized extensions to NoteCards.

[This page intentionally left blank]

7. THE USER INTERFACE

This chapter covers the basics of using the window system NoteCards is built on top of. Read this chapter if you are a new user of the system or you need to understand how to manipulate windows, icons, and use the functions on the background menu. For a lower-level introduction to mouse-button and keyboard use, read Chapter 5, NoteCards Basics.

This chapter explains:

Where to find system menus.

What these menus do.

The Window Menu

The NoteCards window system allows you to interactively manipulate the windows on the screen, moving them around, changing their shape, etc. by selecting various operations from the window menu.

Figure 7-1. The window menu.

When you press and hold down the right mouse button in a window's title bar, as shown in Figure 7-2, that window will come to the top and a menu of window operations will appear. In some cases, you can hold down the right mouse button anywhere in the window to access the window menu.

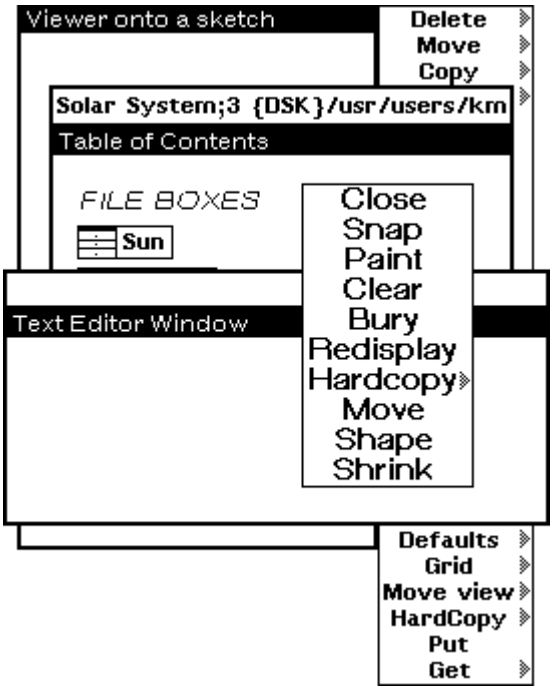


Figure 7-2. The window menu being accessed from a TEdit-window title bar. Note how the TEdit window has been brought to the top of all the windows.

Close

Closes a window, i.e., removes it from the screen. If you have modified the contents of the window, the system may ask you to confirm, by pressing the left mouse button, that you want to close the window and lose your changes. Pressing the right or middle mouse button cancels the close operation.

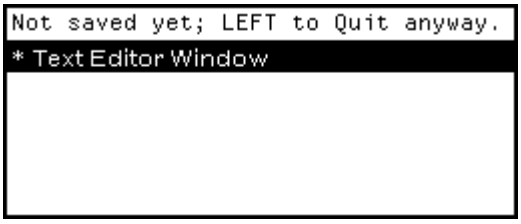


Figure 7-3. A TEdit window requesting you to confirm the **Close** command.

Snap

Prompts for a region on the screen and makes a new window whose image is a snapshot of the image currently in that region. Useful for saving some particularly choice image before the window image changes.

Figure 7-4. The mouse cursor prompting you to sweep out a new size and shape for a window.

Most of the images presented as figures in this manual are screen snaps.

When making a snap, it is possible to change corners and change the snap shape from a different side of the ghost frame. In the middle of a snap process, while you still have the left mouse button down, simultaneously press and hold down the right mouse button.

When you do this, the forceps cursor ( ) will appear. Move this cursor to the corner of the ghost frame you want to move and still holding the left mouse button down, release the right mouse button. You will now be able to adjust the snap from the corner you just selected.

Paint

Switches to a mode in which the cursor can be used like a paint brush to draw in a window. This can be useful for making notes on a window or touching up snaps. While the left key is down, pixels are blackened. While the middle key is down, they are erased. The right button pops up a command menu that allows you to change the brush size, shape, shade, mode, and to quit the paint utility.

Figure 7-5. Paint command menu.

Set Mode

Brings up the menu shown in Figure 7-6.

Figure 7-6. The Mode menu for Paint.

Changing the paint mode changes the way the bits in the brush combine with the existing bits in the snap.

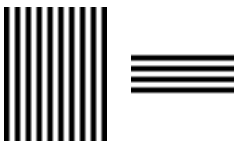


Figure 7-7. The vertical bars show a sample bit map pattern from a snap. The horizontal bars are the pattern in the example brush.

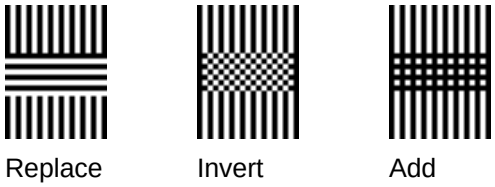


Figure 7-8. The snap and brush patterns shown in Figure 7-7 combine as shown here.

Set Shade

Pops up a menu which allows you to choose one of a preexisting set of shades or specify your own 4x4 shade. New shades are added to the "Choose shade" menu.

Figure 7-9. The "Choose shade" menu.

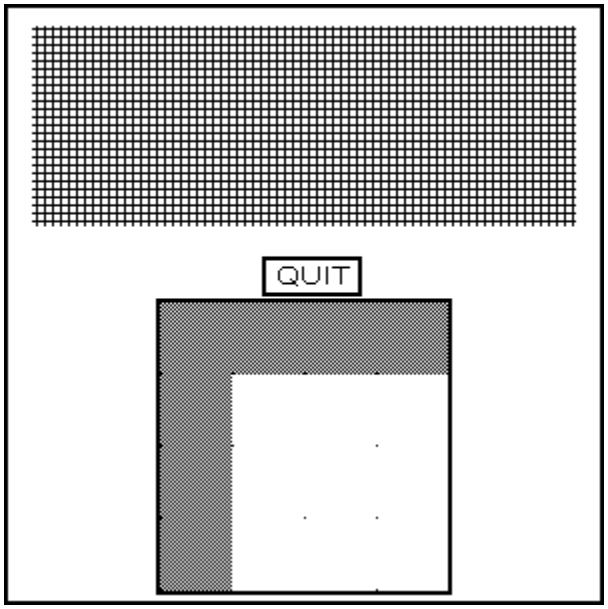


Figure 7-10. The 4x4 shade tool.

Set Shape

Pops us the menu below which allows you to set the shape of the mouse cursor.

Figure 7-11. The shape menu.

Set Size

Brings ut the following menu which sets the cursor brush size.

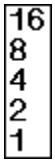


Figure 7-12. The size menu.

Quit

Quits the paint program.

Clear

Clears the window. Sets all pixels in the window to white.

Bury

Places the window under all other windows which it covers or overlaps, thereby exposing any windows that it was hiding.

Redisplay

Clears the window and rewrites the window contents to the window.

Hardcopy

Sends the contents of the window to the printer. If the window is associated with a text, sketch, or graph editor the editor's contents are printed. In the case of TEdit and Sketch, this means that the associated TEdit file, if there is one, is printed. If the window is not associated with an editor a bit-map image of the window is sent to the printer.

To save the image in a PostScript, Interpress, or Press formatted file, or to send it to a non-default printer, use the submenu of the **Hardcopy** command. When the mouse is moved off to the right of the **Hardcopy** menu item, a second pop-up menu appears giving the choices **To a file** or **To a printer**.

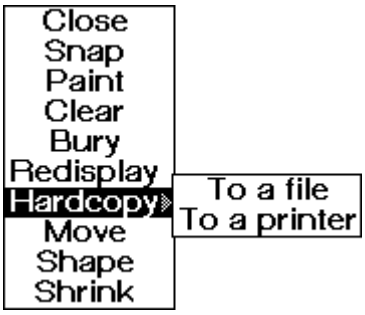


Figure 7-13. The Hardcopy submenu.

If **To a file** is selected, you are prompted to supply a file name, and the format of the file (PostScript or Interpress) from the "File type?" menu and the contents of the window is stored in that file, formatted for the printer type you specified.

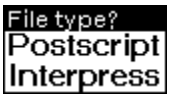


Figure 7-14. The "File type?" menu.

The system provides the suffixes for these files. PostScript files have a "ps" suffix. Interpress files have an "ip" suffix. For example, given a file name of "Letter," the system stores a file with the name "Letter.ps" for a PostScript formatted file and "Letter.ip" for an Interpress formatted file. If you change the suffix for a formatted file from "ps" or "ip" to some other name, the system will prompt you to let it know the file type when you try to print the file. For a formatted file, select the **BINARY** response. For an unformatted file, select the **TEXT** response. You print a formatted file the same way you print an unformatted file, by using the **Hardcopy** option on the FileBrowser menu.

Figure 7-15. The "File Type?" menu requesting the file type of a file formatted for a particular printer type.

If you select **To a printer**, you are prompted to select a printer from the list of known printers, or to type the name of another printer.

Figure 7-16. The "Which printer?" menu asking which printer you want to send to.

The topmost printer in the "Which printer?" menu is the default printer. If the printer selected is not the topmost printer on the list, indicating that it is not the default printer, you will be asked whether to make the printer you selected the new default printer.

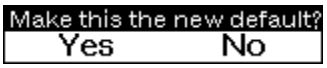


Figure 7-17. The "Make this the new default?" menu asking you whether you want to make the printer you selected the default printer.

Note, unless you are using only fully specified Interpress printer names or you have set the DEFAULTPRINTERTYPE variable from your initialization file, it is unadvisable to use the **Other...** option on the "Which printer?" menu shown in Figure 7-16, as using this option does not allow you to specify the printer type.

Move

Moves the window to a new screen location. When you select **Move** you are presented with a ghost frame which is the size and shape of the original of the window you are moving. You move the mouse to position the ghost frame where you want the window to appear and then you plant the window by clicking the left mouse button.

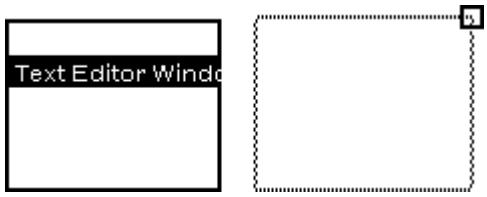


Figure 7-18. A TEdit window and its ghost frame.

Shape

Allows the user to specify a new size and shape for an existing window. You can use either the left or middle mouse buttons to shape a window.

Using the left mouse button allows you to change the size, shape, and location of a window. Once you have selected **Shape** from the window menu, position the cursor where you want the upper left corner of the window to be located, press and hold down the left mouse button, sweep out a new region for the window, and release the mouse button when the window has the size and shape you want.

Using the middle mouse button allows you to change the size and shape of a window by moving just one corner of the window. Select **Shape** from the window menu and position the mouse cursor at the corner you want to move. Press and hold down the middle mouse button and move the mouse cursor to the location you want that corner of the window to be at. A ghost frame will appear to indicate the new size and shape of the window.

When reshaping a window with either the left or middle mouse button it is possible to change corners and change the window shape from a different side of the window. In the middle of a shape process, while you still have the left or middle mouse button down, simultaneously press and hold the right mouse button down. When you do this, the forceps cursor () will appear. Move this cursor to the window corner you want to move and still holding the left or middle mouse button down, release the right mouse button. You will now be able to reshape the window from the corner you just selected.

If you have too many windows on the screen and you are having problems accessing them, you probably want to try shrinking some windows.

Shrink

Removes the window from the screen and brings up its icon. The window can be restored by clicking the middle mouse button inside the window's icon, or by selecting **Expand** from the right button icon menu. Some icons are shown below.

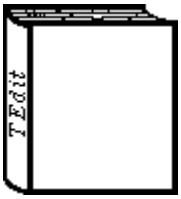


Figure 7-19. TEdit-window shrink icons look like books.

Figure 7-20. Sketch-window shrink icons look like sketch pads.

Figure 7-21. FileBrowser-window shrink icons look like filing cabinet drawers.

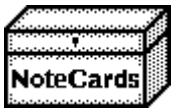


Figure 7-22. The NoteCards MenuBox shrink icon looks like a recipe file box.

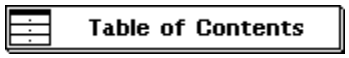


Figure 7-23. Notecard shrink icons look like link icons with an extra border.

The Icon Menu

Icons are a varient of windows and have a menu similar in form and function to the window menu.

Figure 7-24. The icon menu.

For the options Close, Snap, Paint, Bury, Move, and Shape see the discussion above under the "Window Menu" heading.

Expand

Restores the window associated with the icon accessed and removes the icon image from the screen. You can also expend an icon by clicking the middle mouse button inside the icon.

The Background Menu

If the right button is pressed while the cursor is not in any window, the background menu appears.

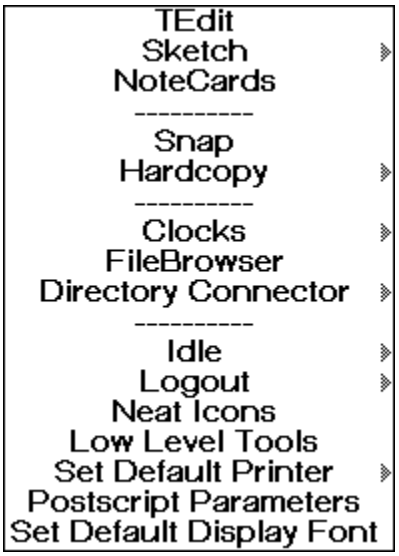


Figure 7-25. The background menu.

TEdit

Opens a new TEdit window and starts a new TEdit session.

Sketch

Opens a new Sketch window and starts a new Sketch session.

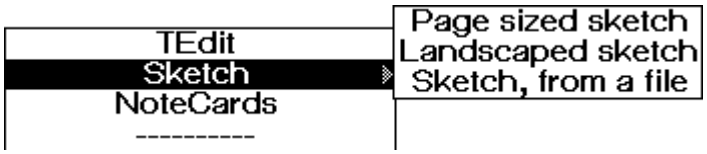


Figure 7-26. The Sketch submenu options.

Page sized sketch

Allows you to start a sketch which will fit exactly on an 8 1/2 x 11 inch sheet of paper when held in the normal vertical or portrait position.

Landscaped sketch

Allows you to start a sketch which will fit exactly on an 8 1/2 x 11 inch sheet of paper when held in a horizontal or portrate position.

Sketch, from a file

Allows you to start a preexisting sketch which is stored in a file.

NoteCards

Opens the NoteCards MenuBox Icon which is the interface for a NoteCards session.

Snap

The same as the window menu command Snap described above.

Also, TEdit, Sketch, and NoteCards allow information to be shift-inserted at the current cursor position by selecting an area of the screen with the SHIFT key held down. To shift-insert the bitmap of a snap into an editor, position the cursor where you want the image to appear, hold the SHIFT key down, press and hold down the right mouse button in the background, and select **Snap** from the single item menu which appears. Finally, sweep out the area for the snap and release the mouse button and the SHIFT key. The snap will appear in the editor where the cursor was positioned. Note, sometimes it is necessary to scroll or redisplay the window for the snap to appear.

Hardcopy

Prompts for a region on the screen, and sends the bit map image to the default printer. Note that the region can cross window boundaries.

Like the window menu **Hardcopy** command discussed above, you can print to a file or specify a different printer than the default by using the **Hardcopy** submenu. See the discussion of the Hardcopy command under the "Window Menu" section above, for more detail.

Clocks

Starts an analog clock and places it on the screen. The Clocks submenu gives you access to a digital clock.

Figure 7-27. The Clocks option on the background menu.

For a complete discussion of this menu option, see Chapter 15, Other Tools.

FileBrowser

Opens a FileBrowser window which prompts you for a directory to browse.

For a complete discussion of this menu option, see Chapter 14, The FileBrowser.

Directory Connector

Opens a window which displays your currently connected directory, and allows you to change directories.

For a complete discussion of this menu option, see Chapter 15, Other Tools.

Idle

Enters idle mode, which blacks out the display screen to save the phosphor. Idle mode can be exited by pressing any key on the keyboard or mouse. This menu command has subitems that allow the user to interactively set idle options to erase the password cache (for security), to request a password before exiting idle mode, to change the timeout before idle mode is entered automatically, etc.

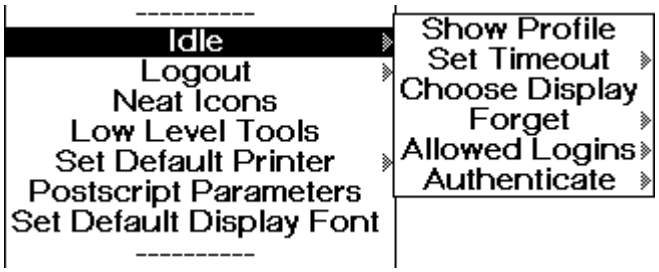


Figure 7-28. The **Idle** submenu.

If either shift key is pressed while NoteCards is in idle mode, the current user name and the amount of time spent idling are displayed in the prompt window. This information appears as long as the shift key is held down.

Show Profile

Displays the current idle profile in the system prompt window.

Set Timeout

Sets the amount of time that the machine will wait for a key stroke or mouse click before automatically going into idle mode.

When you select this option the system brings up a number pad on which you can enter the new idle time out. An idle time out of zero sets the machine so that it will never go into idle mode.

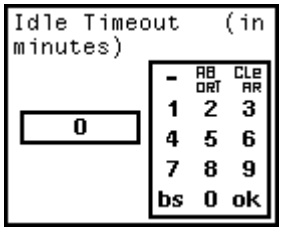


Figure 7-29. The number pad menu requesting a new idle timeout duration.

The submenu option **Never**, will set the machine so that it never goes into idle mode. We counsel against using this option as it can result in the phosphor on your screen being damaged.

Choose Display

Allows you to select what will be displayed on the blacked out screen when the machine is in idle mode. Selecting this option brings up the two-choice menu shown below.

Figure 7-30. The idle display menu.

Bouncing Box Chooses the Envos logo as the idle display.

Figure 7-31. The **Bouncing Box** Envos logo.

Bouncing Username Chooses the your username as the item to bounce around on the screen while in idle mode.

Forget

If set to "Don't" (or NIL), your password is not erased when idle mode is entered. Default is "Do" (or T(rue)) erase password. The initialization file distributed with the NoteCards system resets **Forget** to "Don't."

Figure 7-32. The **Forget** submenu.

Note: If the password is erased, any programs left running when idle mode is entered will fail if they try doing anything requiring passwords, such as accessing file servers.

Allowed Logins

Determines who can exit idle mode.

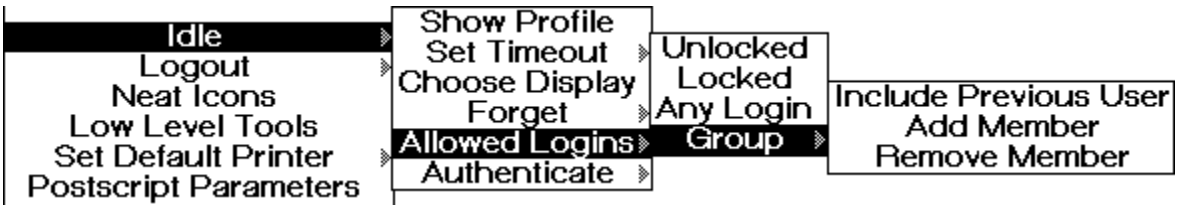


Figure 7-33. The "Allowed Logins" submenus.

Unlocked Sets the system such that login is not required to exit idle mode. Initialization file value is "NIL."

Locked Lets only the previous user exit idle mode. Initialization file value is "(T)."

Any Login Require login, but let anyone exit idle mode. Login overwrites the previous user's user name and password each time idle mode is exited. Initialization file value is "(*)."

Group Allow any members of a specified group to exit idle mode. Figure 7-33 shows the **Group** submenu which allows you to add and delete group members.

Authenticate

The value of this property determines what mechanism the system uses to check passwords.

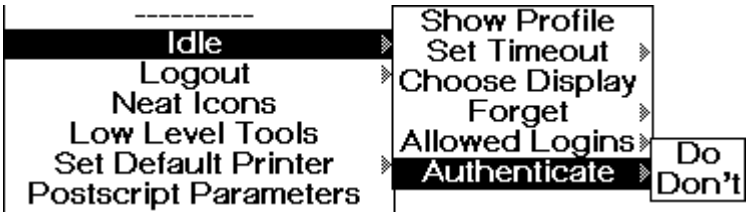


Figure 7-34. The Authenticate submenu.

If the value of this property is "Do" or "T(rue)," the Xerox network system protocol is used to do the authentication.

If the property value is "UNIX," the Sun Operating System is used to do the authentication. This option is set in the standard initialization file. See Appendix C for a complete discussion.

If the value of this property is "Don't" or NIL, the password is not checked. Any password is accepted.

Logout

Saves the NoteCards virtual memory working image to a file and returns the system to UNIX. This is the normal way to return to UNIX.

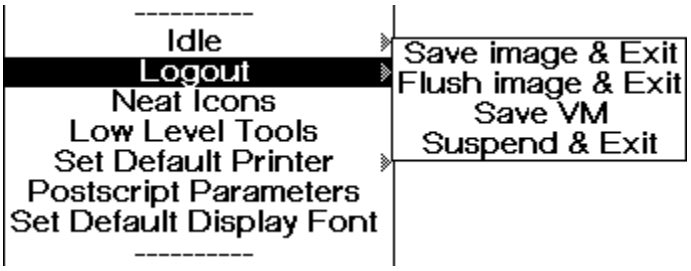


Figure 7-35. The Logout option and its submenu on the background menu.

Save image & Exit

The same as Logout.

Flush image & Exit.

Returns the system to UNIX without saving the working image. Use this option when you have saved all your work to files and you do not care if the system comes up in the same state it was in when you left it.

Save VM

Saves the NoteCards virtual memory working image to a file but does not return the system to UNIX. Use this option frequently if you are concerned about power failures or other problems which may cause your machine to crash.

Suspend & Exit

Suspends the NoteCards session and returns control to UNIX. You restart the NoteCards session by entering the command **`fg`** (foreground), at the UNIX prompt.

Low Level Tools

This menu option provides people extending the NoteCards system access to the Lisp executives, the process status window, and allows them to set the **Show TEdit Props** flag. The NoteCards user should never access these options except under the instruction of a developer or support person.

Set Default Printer

Allows you to specify which printers you want to print to and which one is the default.

For a complete discussion of this menu option, see Chapter 16, Printing.

Postscript Parameters

Allows you to control the way PostScript documents are printed.

For a complete discussion of this menu option, see Chapter 16, Printing.

Set Default Display Font

Brings up a series of menus which allow you to specify what fonts will be used on the screen. This is a particularly useful option if you are using a high resolution screen and want to increase font size or for people who are vision impaired.

Figure 7-36. The "Choose Font Profile to Update" menu.

To change a display font, select the **Set Default Display Font** background menu option and choose the font you want to change from the "Choose Font Profile to Update" menu. Each of the font options is discussed below. Once you have selected the font you want to modify, the "Please select a font:" menu will appear. Select the font family, size, and face you want then select **DONE**. **RESET** makes the menu display the current setting for the default display font you are modifying. **ABORT** terminates this operation and closed the "Please select a font:" menu.

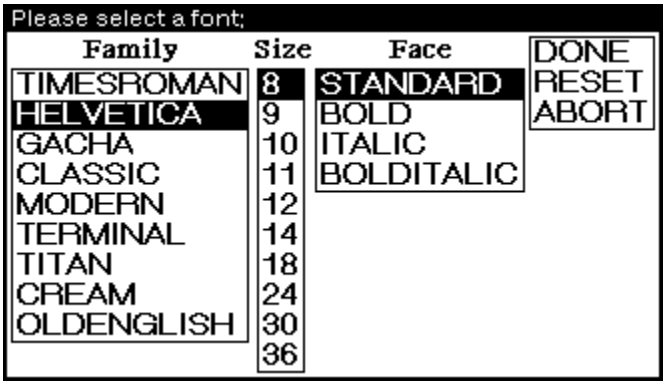


Figure 7-37. The "Please select a font:" menu.

If you choose a font which does not exist or the system is unable to find, the old font specification remains unchanged.

If the system is unable to find a font which you know to exist, check the setting of the display font directory list. This is done with the **Show Font Directories** subsubmenu option off the **Set Default Printer** background menu option. See Chapter 16, Printing, for a complete description of how to use this option.

FileBrowser Display Font

The font in which the information in the main display window is printed, initially 10-point Gacha.

FileBrowser Prompt Font

The font in which FileBrowser prompt messages are printed. Initially 8-point Gacha. Changing this value only affects new FileBrowsers created from the background menu. Existing FileBrowsers are unaffected.

Prompt Window Font

The font used to print information in the black system-prompt window.

Default Font

The starting font used in TEdit and Sketch windows as well as all other windows which do not have an explicitly specified font associated with them.

Note that the default font for NoteCards cards is specified from the "NoteCards System Parameters" menu. See Chapter 13, System Parameters for a complete discussion of how to specify this default font.

Menu Font

The font used in all system menus. The menu font is used in the window, icon, background, and other menus.

[This page intentionally left blank]

8. LINKS

A link is a connection between a source card and a destination card. A link icon represents the link.

Figure 8-1. A link-icon image.

By clicking on the link icon, you can display the destination card. Links are the glue which holds together the relationships and conceptual structures represented in a notefile.

This chapter explains:

What links and link types are.

How links are accessed and used.

How to create links.

How to delete and undelete links.

How to tailor links to your taste.

What cross-file links are.

Links and Link Types

Link types specify the nature of the relationship between the source and destination cards. Some typical link types are Question, Answer, and Explanation.

Two groups of link types exist in the NoteCards system; one group is system-reserved, the other is user-specified. Some of the system-reserved links support the FileBox hierarchy that NoteCards helps you to create. The user-specified links add a relational dimension to information structuring. Here you can link cards together to form networks. These networks allow you to represent the interconnections between various ideas or pieces of information, independent of any categorization into topic areas. In contrast, the set of FileBoxes forms a strict hierarchy typically representing a breakdown of information into subtopics or subcategories. By strict hierarchy, we mean that you are not allowed to create circular lists of FileBoxes. Circular lists of other card types are allowed, however.

System-Reserved Link Types

SubBox A link pointing to a FileBox from another FileBox.

FiledCard A link pointing from a FileBox card to any non-FileBox card. Text, Sketch, Graph, Browser, Search, LinkIndex, and Document cards are non-FileBox cards.

BrowserContents A link pointing to a card or box from a Browser card. This type of link is traversed when you bring up any of the cards or boxes represented as link icons in a Browser card.

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

LinkIndexBackPtr A link pointing to a card or box from a LinkIndex card. This type of link is built when you specify that a LinkIndex card should have back pointers to the cards and boxes being indexed, i.e., link icons should be included in the LinkIndex card.

DocBackPtr A link pointing to a card or box from a Document card. This type of link is built when you specify that a Document card should include back pointers to each card and/or box used to build the document.

User-Specified Link Types

User-specified links indicate the logical relationships between cards. The system comes with three predefined user-specified link types, Comment, See, and Unspecified. However, the system allows you to create new link types as you need them. Typical link types users create include Explanation, Example, Question, Answer, Next, and Source. Choose your link types carefully as they determine the lines along which you will be able to extract information from your notefiles with Document cards.

Randy Trigg, one of the developers of the NoteCards prototype at Xerox PARC, proposed a long list of standard link types for users to follow in categorizing the relationships between their notes. A partial list is included here to give you some idea of the possibilities.

- Citation
  - C-source
  - C-credit
  - C-leads
- Argument
  - A-deduction
  - A-induction
  - A-analogy
  - A-intuition
- Background/Future
- Refutation/Support
- Methodology/Data
- Solution
- Continuation
- Correction/Update
- Simplification/Complication
- Explanation
- Summarization/Detail
- Alternate-view
- Rewrite
- Generalization/Specification
- Abstraction/Example
- Formalization/Application

Note that some long-time users of the system use only one link type. NoteCards gives you a great deal of flexibility. Just because some functionality is there does not mean you should feel obligated to use it if it is not helpful to your task.

Link Directions and Link Ends

Links have a single start point and a single end point. However, whenever you create a link from one card to another, the system automatically builds a backwards link so that it is possible to traverse a linked sequence of cards in either direction. The result is that all cards are linked bidirectionally.

A link's start point is either the card itself or somewhere in the card's contents. In the case of a text card, this would be somewhere within the body of the card's text. A link's end point is always another card. Thus, there are three distinct link categories. There are also a few terms which refer to different groupings of these categories.

Link Categories

To-Links Refer to links whose start point is in a card's contents and whose end point is another card. These links are represented as link icons in a card's body.

Global To-Links Refer to links whose start point is a card itself and whose end point is another card. These links are not represented by link icons in a card's body. To see and access these link icons you must choose the **Show Links** option from a card's left-button title-bar menu.

From-Links For every to-link and global to-link there is a corresponding from-link which points from the destination card to the source card. All from-links are global.

Other Link Terminology

Local Links Links whose start point is somewhere in the contents of a card.

Global Links Links whose start point is the card itself and not some point in the card's contents. These links are not represented by link icons in a card's body. To see and access these link icons, you must choose the **Show Links** option from a card's left-button title-bar menu.

Forward Links Include both to-links and global to-links. This term is used when setting Browser and LinkIndex card parameters.

Backward Links The same as from-links. This term is used when setting Browser and LinkIndex card parameters.

Back Links System-generated to-links. These links point from a Document or LinkIndex card *back* to the source cards from which the Document or LinkIndex was built. Hence the name back link.

Link Types vs. Card Types

Link types and card types are frequently confused but they are totally independent of each other.

Link types classify the assigned relationship between two cards. Link types frequently include Explanation, Question, Answer, and Citation.

A card's type specifies the functional capabilities of that card and is associated with a text or graphics editor. The eight card types are Text, FileBox, Search, LinkIndex, Document, Sketch, Graph, and Browser. The Text, FileBox, Search, LinkIndex, and Document cards are all based on the text editor TEdit. The Sketch card is based on the Sketch editor and the Graph, and Browser cards are based on the graph editor Grapher.

Link Access and Use

Link Icon Functionality

A link icon is an active region with an associated menu and four functions. The most frequently used functionality is to click the left mouse button in the body of the link icon to traverse the link to the destination card. The middle mouse button brings up a menu which allows you to traverse the link, change the link type, change the card title and change the link display mode.

Active Regions of Link Icons

The entire link icon except for 10 pixels at the left edge and 10 pixels at the right edge responds to mouse clicks by carrying out a NoteCards action.

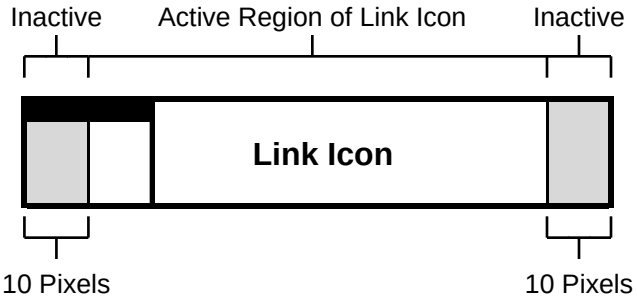


Figure 8-2. Link icon active and inactive regions.

Clicking in the "inactive" 10-pixel strips at the left and right edges of the link icon selects the link icon, in text-based cards (Text, FileBox, Search, LinkIndex, and Document cards), for delete, copy, and move operations, without traversing the link. This region does not exist on sketch- and graph-based cards (Sketch, Graph, and Browser cards).

The inactive area can be somewhat difficult to access without touching the active region of the card. One way to access it is to hold the left or middle mouse button down and slide into the link icon's inactive region. You have selected the link icon when it appears underlined.

Figure 8-3. A selected icon

If you go too far and the card redisplays in reverse video, simply slide the mouse cursor out to the edge of the link icon and watch for it to underline again. Once the icon appears underlined release the mouse button.

If a link icon is so narrow that its active region would be less than 10 pixels wide, the active region is set to the middle 50% of the link icon width, with 25% at left and 25% at the right being the inactive strips. If the link icon is less than 10 pixels in width altogether, then the link icon will have no inactive strips. Its entire area will be active.

Mouse-Button Actions in Link Icons

The exact response to a mouse click in the active region of a link icon varies depending on a number of conditions. For more general mouse button behaviors, see Chapter 7, The User Interface, as well as the *User's Guide to TEdit* and the *User's Guide to Sketch*.

Left Button Brings up the destination card represented by the link icon.

Middle Button Brings up the link icon menu which allows you to change the link type, title, and display mode.

Right Button Following a left- or middle-button click elsewhere in a TEdit-based card, selects a region of the window for moving, copying, or deletion which can include link icons.

Copy/Shift or Move/Shift-Control key plus mouse-button operations are three-step operations. First, you specify the destination window by clicking the left mouse button to position the flashing cursor in the window. Second, hold the appropriate keyboard key down. Third, while still holding the key down, use the mouse to select the piece of text/graphics to be operated on. Once you have completed your selection, release the keyboard key and what you have selected will be copied or moved to the destination window.

Link icons should only be copied or moved to NoteCards windows. TEdit and Sketch, when used outside of NoteCards text-based and sketch-based cards, will not understand link icons.

Copy/Shift key + Left Button This operation is frequently referred to as a shift-select, occasionally as a copy-select. This operation copies a link icon, or link-icon name and link, to another card.

If the flashing cursor is in the body of a card, holding the SHIFT or COPY key down and clicking the left mouse button on a link icon copies the link icon and the link it represents to the position of the caret cursor in the destination card.

If the flashing cursor is a crosshairs and is in a "Selecting cards to file" prompt window, holding the SHIFT or COPY key down and clicking the left mouse button on a link icon will copy the link icon name and link to the select-card prompt window. This operation is used when you are inserting links in cards or filing cards in a FileBox card.

Figure 8-4. A select card prompt window.

Copy/Shift key+ Middle Button Same as left button.

Copy/Shift key + Right Button Holding the COPY or SHIFT key down and pressing the right mouse button following a left- or middle-button click elsewhere in a text-based card, selects a region of the window for copying which can include link icons.

Move/Shift-Control + Left Button Moves a link and link icon from one card to another card by deleting the link in the first card and inserting it into the second. To move a link, position the caret cursor in the destination window where you want the link to appear and, holding the MOVE key down or holding the CONTROL and SHIFT keys down simultaneously, select the link icon you want to move with the left mouse button. When you release the key or keys you are holding down, the link icon will move to its new position.

Move/Shift-Control + Middle Button Same as left button.

Move/Shift-Control + Right Button Holding the MOVE key or SHIFT and CONTROL keys down and pressing the right mouse button following a left- or middle-button click elsewhere in a text-based card, selects a region of the window to be moved which can include link icons.

CONTROL key plus mouse button operations are two-step operations. First, hold the CONTROL key down. Second, while still holding the key down, use the mouse to select the piece of text/graphics to be deleted. Once you have completed your selection, release the CONTROL key and what you have selected will be deleted.

Control key+ Left Button Deletes a link and link icon from a card. Holding the CONTROL key down and pressing the left mouse button on a link icon will select the link icon for deletion. This is shown by displaying the link icon in reverse video.

Figure 8-5. A selected link icon.

The link icon and the link it represents are deleted when you release the CONTROL key. You can cancel this operation before you release the CONTROL key by clicking elsewhere in the window to deselect the link icon.

Figure 8-6. An unselected link icon.

You can undo this operation, after you have released the control key, by hitting the UNDO key.

Control key + Middle Button Same as left button.

Control key + Right Button Holding the CONTROL key down and pressing the right mouse button following a left- or middle-button click elsewhere in a text-based card, selects a region of the window for deletion which can include link icons.

It is also possible to just backspace over a link icon or to select it and then hit the DELETE key. Note that on some Sun keyboards, the backspace key has the word "Delete" printed on it and that the NoteCards DELETE key is one of the function keys, usually the key labeled "L10."

Viewing Local and Global Links

All links are stored in three places.

1) as a to-link on a source card

2) as a from-link on a destination card

and either

3a) for local to-links, in the card's contents as a link icon

3b) for global to-links, in the card's global links list

Only the local to-links, the links in a card's contents, are immediately visible.

To see all of the links to and from a card, hold the left mouse button down in the card's title bar and select the **Show Links** option.

Figure 8-7. The card menu with the **Show Links** option selected.

This will open a window titled: "List of Links". This window always opens with the same initial height. If the list of links is longer than three or four, you will need to scroll or reshape the window to see

the rest of the links. The link icons in this window have the standard left-button functionality, you can traverse links, but no middle-button functionality. Links can also be copied, moved, and deleted from this window.

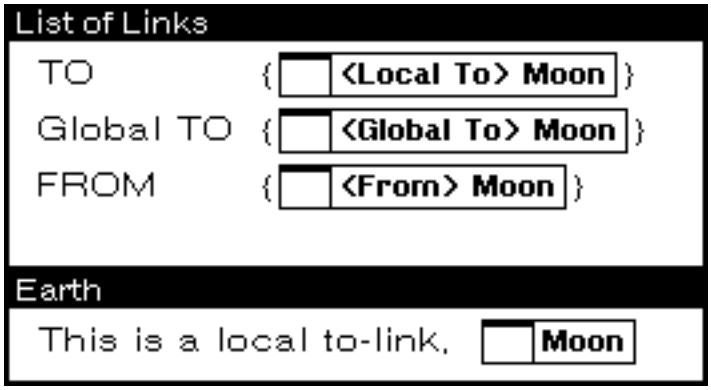


Figure 8-8. A "List of Links" window showing links to and from the "Earth"and "Moon" cards.

The headings to the left of the links are active but have no operations associated with them.

You can close the "List of Links" window by holding down the left or middle mouse button in the window title bar and selecting **Quit** from the single-option menu, or by pressing the right mouse button in the title bar and selecting the **Close** option from the standard window menu.

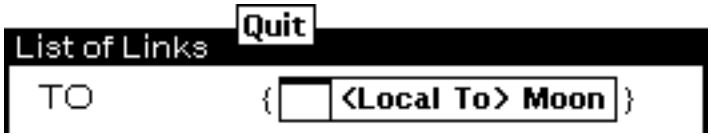


Figure 8-9. The "List of Links" left button menu.

Unfiled and Lost Cards

All cards, including FileBox cards, must be filed in at least one FileBox. This assures that you will have a pointer to your information and helps to remind you to hierarchically organize your information in addition to specifying other types of relationships. When you close a card without linking it to a FileBox card, the closed card is placed in the "To Be Filed" FileBox. You can access this FileBox card by holding the middle mouse button down on the notefile-banner **Special Cards** menu option and selecting the **To Be Filed** option from the menu which appears.

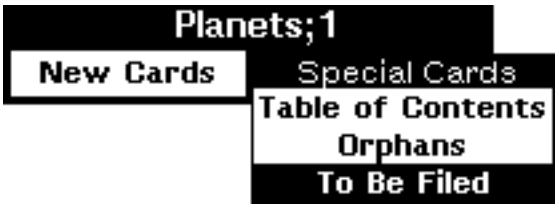


Figure 8-10. The **To Be Filed** option on the "Special Cards" menu.

From this FileBox all "unfiled" cards are accessible. It is possible to force yourself to file cards in a FileBox by setting the **Force Filing**

Edit Parameter. See Chapter 13, System Parameters, for a complete explanation.

You can remove all FileBox links to a card by accident or by design. When you do this, the following message is displayed in the System Prompt Window. "You have just unfiled *card name* from its last filebox. It is being filed in the Orphan FileBox." You can access this FileBox card by holding the middle mouse button down on the notefile-banner **Special Cards** menu option and selecting the **Orphans** option from the menu which appears.

Finally, you can delete card links from the "Orphans" FileBox. When you do this the following message is displayed in the card's prompt window. "You have just deleted the last filing link to *card name*. The Search operation can be used to find it." At this point, if there are no other cards with links to this card, it becomes a lost card and there is no simple way to access the card even though it still exists. As the message says, though, you can retrieve lost cards by building new links to them with a Search card. However, unless you remember some fragment of the card's name, using a Search card to find lost cards can be a time consuming task in a large notefile.

Creating Links

The links you create always run from a source card to a destination card. References to the source card refer to the card with the link icon. References to the destination card refer to the card which opens, or flashes if already open, when you click on the link icon.

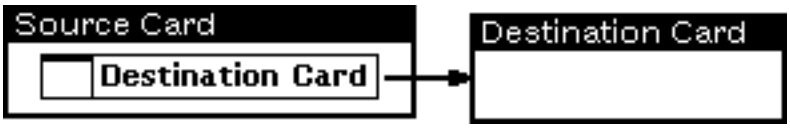


Figure 8-11. An illustration of a link icon pointing to its destination card.

Creating Links in Text Cards

The procedures for creating links are basically the same for all cards. We will use Text cards as the model, and then discuss the differences between Text cards and other cards.

Insert Link

You use the **Insert Link** command to create a link originating in the source card's contents and terminating at another card. This is a multi-step process. First, you place the caret cursor in the card at the position where you want the link to appear by clicking in the source card's window. If the type-in process belongs to another card or window this usually takes two mouse clicks, one to make the card the active card and one to position the mouse cursor. Once you have established the location for the link icon in the

source card, you hold the left mouse button down in the source card's title bar and select the **Insert Link** option.

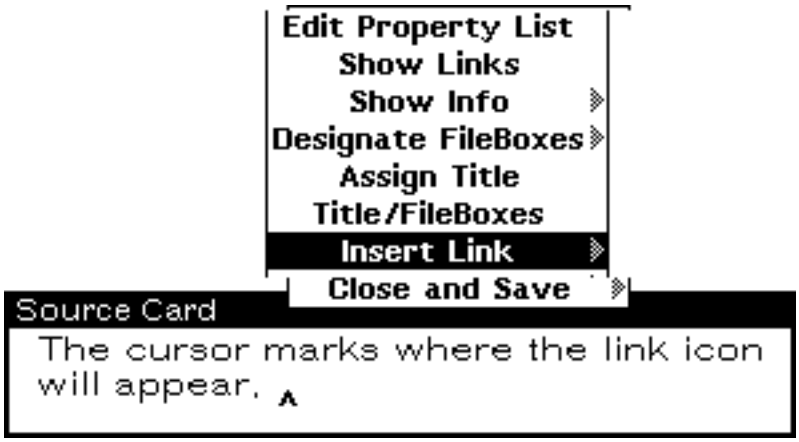


Figure 8-12. The card menu with the **Insert Link** option selected.

At this point the "Link Type" menu will open, asking you to specify the link type.

Figure 8-13. The "Link Type" menu with the **--New Link Type--** option selected.

Choosing appropriate link types is important as they will determine the ways you will be able to extract information from your notefiles. For more information on link types see the section User-Specified Link Types in this chapter and the section on Document cards in Chapter 12, System Cards. You can create new link types as you need them by selecting the **--New Link Type--** option and typing the name of the new link type into the prompt window which will appear above the source card. When you type a carriage return to this prompt widow, without defining a new link type, the system assigns the link type the value "Unspecified."

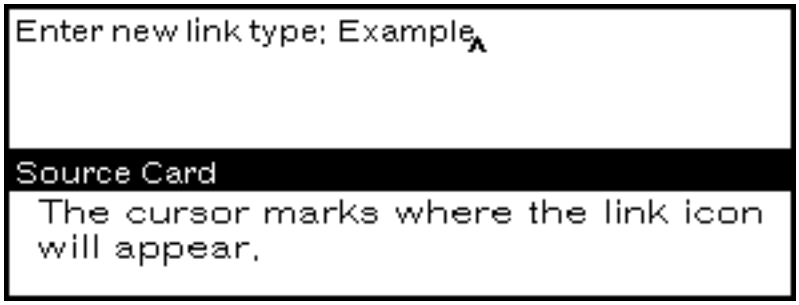


Figure 8-14. A note card prompt window requesting a new link type.

After you have selected an existing link type or defined a new one, the system opens a "Selecting Note Card" window and prompts you to shift-select the card UID from the destination card.

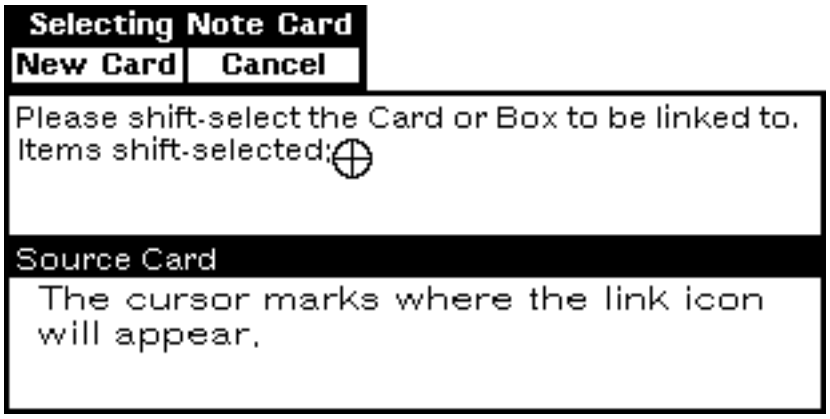


Figure 8-15. A "Selecting Note Card" window prompting the user to shift-select a card ID from the destination card's title bar.

A card UID is shift-selectable from three locations,

a card's title bar 

a card's shrink icon 

and a card's link icon 

When shift-selecting the card UID, there are two distinct behaviors. From the link icon, it is necessary to lift the Shift or Copy key after each selection. This is the expected behavior when shift-selecting anything from an edit window. From the title bar and shrink icon, it is not necessary to lift the Shift key. The transfer happens immediately.

What you see transfered is the destination card's title, but NoteCards also maintains an identifier for each card which is unique across all cards and notefiles. This is what you are really shift-selecting from one card to another. And this is also the reason why it is not possible to shift-select or type text into this window.

Figure 8-16. A link icon to a destination card as it appears in a source card.

At this point, the new link icon will appear in the source card. From now on, when you click on this link icon, it will open the destination card, or flash the destination card if it is already open.

Selecting **New Card** from the menu on top of the "Selecting Note Card" prompt window will bring up a menu allowing you to create a new card, of any type, which will be the destination card.

Selecting **Cancel** from this menu, cancels the entire **Insert Link** operation.

At any point in this process, it is possible to transfer the type-in process from a prompt window to some other window by clicking in the window you want to use. The prompt window will stay open waiting for you to complete the operation. When you are ready to finish inserting the link, simply click in the prompt window to make it the active window, shown by the flashing crosshairs, and continue where you left off.

Insert Links

If you need to insert multiple links of the same link type at the same location in a source card, **Insert Links** will save you from inserting each link individually.

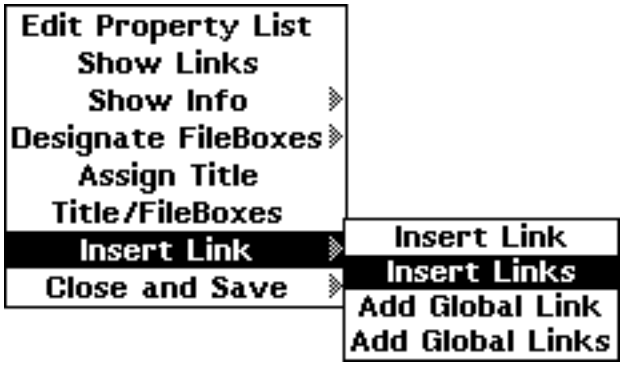


Figure 8-17. The card menu with the **Insert Links** option on the **Insert Link** submenu selected.

Insert Links works exactly as **Insert Link** except you will be allowed to select multiple destination cards. It is possible to shift-select a mixture of text and link icons from a card. In this case, the source card's prompt window will ignore all the extraneous text. To complete the insert operation, either type a carriage return or select the **Done** menu option.

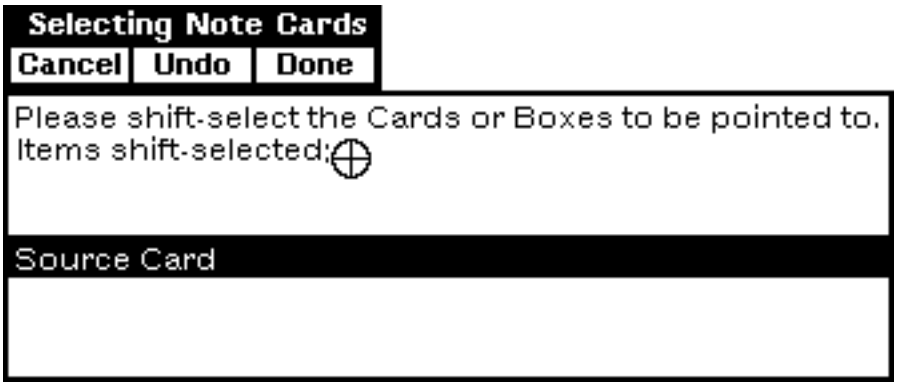


Figure 8-18. A "Selecting Note Cards" prompt window.

The **Cancel** option terminates the entire link insertion operation.

Undo allows you to remove the last item from the list. You can select **Undo** until you have removed all the items from the list.

Add Global Link

Global links are not rooted in a card's contents. For this reason we draw the contrast between *inserting* a link into a card's contents and *adding* a global link to a card.

A second result of this difference is that the procedure for **Add Global Link** is somewhat different from that for **Insert Link**. To add a global link, hold the left mouse button down on the card's title bar and slide off **Insert Link** to access the submenu and select the **Add Global Link** option.

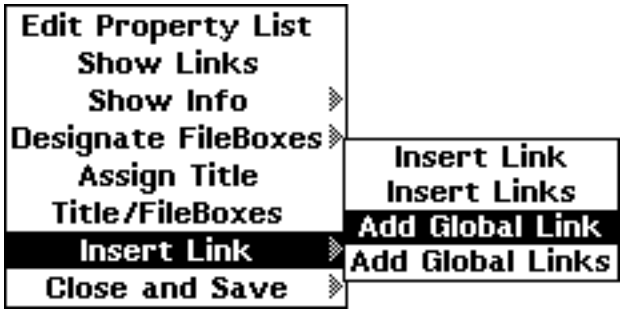


Figure 8-19. The card menu with the **Add Global Links** option on the **Insert Link** submenu selected.

Because you are adding a link to a card itself and not inserting a link into a card's contents, it is not necessary to first position the caret cursor to select the link icon's insertion point. Note that the link will not be visible unless you use the **Show Links** option to show the links to and from that card. Aside from these few minor differences the procedure is exactly the same as that for **Insert Link**.

Add Global Links

Add Global Links functions just like a combination of **Add Global Link** and **Insert Links**. It allows you to add multiple global links of the same link type to a source card without adding each link individually. If you want the global links to have different link types, they must be added one by one using **Add Global Link**.

Creating Links in Other Card Types

There are three basic card types in NoteCards, text-based, sketch-based and graph-based cards.

Text-Based Cards

Text-based cards are based on the TEdit editor and can contain the output from sketch-based and graph-based cards. Text-based cards include Text, FileBox, Search, LinkIndex, and Document cards. The link creation procedure for all text-based cards is the same as for Text cards.

Sketch-Based Cards

Sketch-based cards are based on the Sketch editor. There is only one sketch-based card and it is the Sketch card.

The link icon insertion procedure differs from that for text-based cards only concerning *when* the new link icon is positioned in the card. For text-based cards, the cursor is positioned where you want the link to appear and then one of the link insertion options is selected. For sketch-based cards, the order is reversed. The

menu option is selected first and then, after you have indicated which card you want to link to, you move the cursor into the Sketch card and position the link icon.

Graph-Based Cards

Graph-based cards are based on the graph editor. Graph and Browser cards are graph-based cards.

As with sketch-based cards, the link icon position is selected after the card you are linking to.

Deleting Links

When you delete a link icon you are deleting the to-link that points from the source card to the destination card and also the from-link which points from the destination card back to the source card.

To understand what happens when you delete a link, create a Source Card and a Destination Card. Next insert a link from the Source Card to the Destination Card, and open the "List of Links" windows on each card by selecting the **Show Links** option from the left-button title-bar menu. When you delete one link, all the links will be deleted.

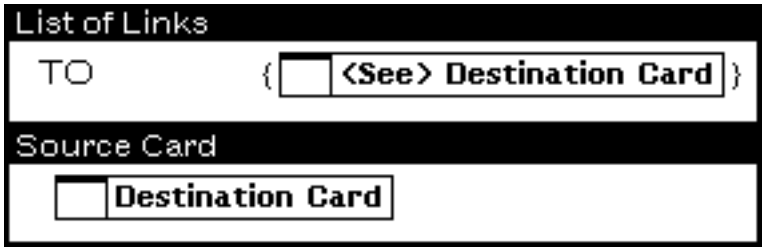


Figure 8-20. A "List of Links" window showing a to-link to a destination card.

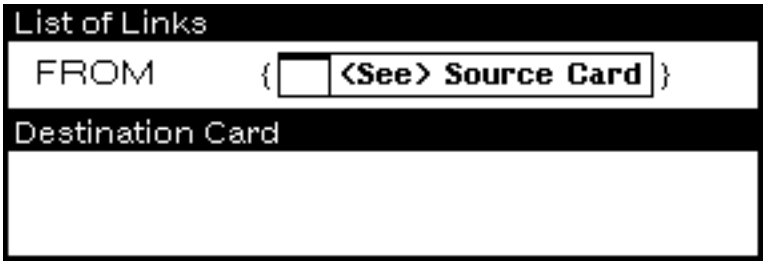


Figure 8-21. A "List of Links" window showing a from-link pointing back to a source card.

The delete procedure for links depends on where you are accessing the link and the card type you are deleting the link from.

Deleting Links from Card Contents

Text-Based Cards

Link icons in the body of text-based cards can be deleted in several ways.

You can select the link icon with the mouse by clicking in the *in*active region of the link icon and hitting the delete key. Also, you can select a region of text in a text-base card with a sequence of left+right or middle+right mouse buttons and hit the delete key to delete the selected region which can include a mixture of text and link icons.

You can also hold the control key down and select the link icon by pressing the left or middle mouse button in the link icon active region, when you release the control key, the link icon will be deleted.

Finally, you can simply backspace over a link to delete it.

All of these operations can be undone by hitting the Undo key.

For more detail on inactive regions of link icons and coordinated keyboard-plus-mouse operations, see the Link Access and Use section above and as well as the TEdit documentation.

Sketch-Based Cards

You can delete link icons in the body of sketch-based cards by bringing up the sketch menu, if it is not permanantly attached to the right side of the sketch window, and selecting the delete option. Next, click in the link icons control point which is the small box attached to one corner of the link icon.

Figure 8-22. A selected link icon in a Sketch card.

This operation can be undone by hitting the Undo key.

For more detail on this and other Sketch operations see the Sketch documentation.

Graph-Based Cards

To delete a link from a Graph card, hold the right mouse button down in the Graph card window, select the **Delete Node** option, and then click on the card you want to delete. When you confirm, the link will be deleted.

Note, this operation cannot be undone.

Because a Browser card is an overview of a link network in addition to being a member of that network, the meanings of delete operations in Browser cards is not straightforward. Please turn to the section on Browser cards in Chapter 11, System Cards, for a detailed explanation of links in Browser cards.

Deleting Links from the Show Links Display

Deleting links from the show links display is very similar to deleting links from text-based cards.

You select the link icon with the mouse by clicking in the inactive region of the link icon and hitting the delete or backspace key.

You can also hold the control key down and select the link icon with the mouse by pressing the left- or middle-mouse button in the link-icon active region, when you release the control key, the link icon will be deleted. This second way is frequently the easier of the two, as it does not require you to access the narrow inactive regions on either end of the link icon.

Note, this delete operation cannot be undone.

Tailoring Links

It is possible to modify the appearance of links to suit your personal tastes and the task at hand. Links can be modified using the middle-button icon menu and the NoteCards "System Parameters" menu.

Link Ops Menu

You bring up the "LinkOps" menu by holding the middle mouse button down in the active region of the link icon.

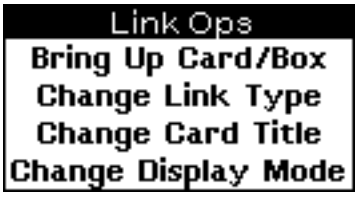


Figure 8-23. The "Link Ops" menu.

Bring Up Card/Box

Performs the same function as clicking the left mouse button in the link icon's active region. It traverses the link to the destination card and opens it, or flashes it if it is already open.

Change Link Type

Allows you to choose a different link type for the link you select.

Change Card Title

Changes the title of the card the link points to and the titles shown in all the link icons pointing to that card. The one exception to this statement is when you have created one-way cross-file links. These links update only when you access the link icon which points to the changed card.

Change Display Mode

Selecting **Change Display Mode** brings up the "Display Mode?" menu. This menu allows you to override the display mode defaults for the individual link icon you have selected.

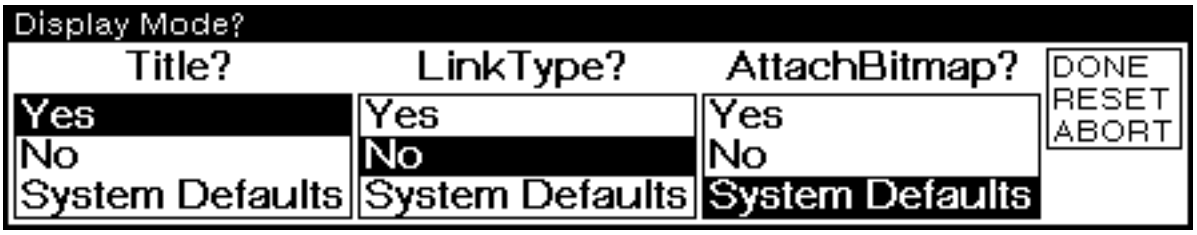


Figure 8-24. The "Display Mode?" menu.

There are two levels of defaults for link icon display modes. Each card defined in the NoteCards system sets how links will be displayed in its contents. This is the first level of defaults and you have no control over them. These defaults can be set to three values, "Yes," "No" and "System Defaults." These are the values which appear in the "Display Mode?" menu shown above. If the value is set to "System Defaults," NoteCards consults the System Parameters to determine whether or not to display that particular attribute. See Chapter 13, System Parameters for how to set link default display modes.

With all the values set to "No," the link appears as a small rectangular box.

Figure 8-25. A link icon with all display mode values set to "No."

With all the values set to "yes," the links have the following appearance.

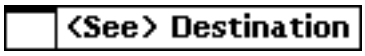


Figure 8-26. A link icon with all display mode values set to "Yes."

The link type is shown in angled brackets and the card title is to the right of the link type. The card-type bitmap is the square piece on the left of the link icon and varies in design depending on the card type.

Represents Text cards.

Represents FileBox cards.
The image is a stylized file drawer.

Represents Sketch cards.
The image is a stylized sketch.

Represents Graph cards.
The image is a stylized letter "G."

Represents Browser cards.
The image is a stylized Browser.

Represents Search cards.
The image is a stylized eye.

Represents LinkIndex cards.
The image is a stylized list of link icons.

Represents Document cards.
The image is a stylized linear list of note cards.

Represents cards for which the type is unknown occurs with cross-file links.

System Parameters for Links

There are nine system parameters to enable you to customize the appearance of links. They are **Link Icon Font**, **Link Icon Border Width**, **Link Icon Multi Line Mode**, **Cross File Link Mode**, **Link Icon Max Width in Pixels**, **Link Icon Show Title Default**, **Link Icon Attach Bitmap Default**, **Link Icon Show Link Type Default**, and **Use Deleted Link Icon Indicators**. For a complete discussion of these parameters, see Chapter 13, System Parameters.

Cross-File Links

The NoteCards system allows you to have links from one notefile that point to cards in another notefile. These types of links are known as cross-file links. See also Chapter 13, System Parameters for how to set the cross-file link mode.

Appearance

Cross-file links are indicated by an arrow bitmap on the right-hand side of the link icon.

Figure 8-27. A cross-file link indicated by an arrow bitmap on the right-hand side of the link icon.

It is also possible to cross-file cards. This is to say that you can file a card from one notefile in the FileBox of another notefile. Note that this is only possible after you have first filed it in its origin notefile. This is, in effect, just another way of creating a cross-file link. Cards are always physcially filed in their origin notefiles.

When a cross-file link is displayed, and the notefile the link points to is closed, then the link can have the following form.

Figure 8-28. A cross-file link to a closed notefile. The question-mark bitmap indicates that the card cannot be accessed to determine its card type.

The card-type bitmap on the left of the link icon is a question mark because the card type is stored on the card and this link has no access to the card because the notefile is closed. As a result, the

link cannot display the appropriate card-type link. When you click on the cross-file link the system will ask you if you want to open the corresponding notefile. At this point, if you redisplay the card, by selecting the **Redisplay** option on the window menu, the link will be shown with its appropriate card-type bitmap.

Missing Notefiles

If you rename or delete a notefile which has cross-file links pointing to it, the system will try to open the notefileby its original name. When it cannot find the notefile under its original name, it will ask you which notefile to look in. You cannot give a name with wild card characters like "*" or "?." You must give actual notefile names. The system will continue to prompt you for new notefiles to search until you give up.

[This page intentionally left blank]

9. CARDS AND BANNERS

Cards are the basic units of information in the NoteCards system. This information can be a mixture of text and graphics. The cards can contain as much or as little information as you deem necessary. You can link each card to any number of other cards to create an information network which you can randomly browse or systematically read. You access cards through the notefile Banner.

This chapter explains:

What you can do from the notefile Banner.

How to access old cards.

How to create new cards.

What card types there are.

What the card menu does.

The Notefile Banner

Each notefile which you have opened is represented on the screen by a notefile Banner. The notefile name appears across the top of the Banner in the region called the title bar. The semi-colon after the name is followed by the notefile version number.

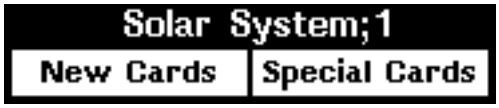


Figure 9-1. The Banner for the Solar System notefile.

The Banner has three active regions, the title bar, the **New Cards** option and the **Special Cards** option.

The Banner Title Bar

From the title bar you can access three notefile menus.

Notefile Ops Menus

There are two "Notefile Ops" Menus. One displays only those options which work on open notefiles. The other displays only those options which work on closed notefiles.

Holding the left mouse button down in the Banner title bar when the notefile is *closed* brings up the following "Notefile Ops" menu. Note that the **New Cards** and **Special Cards** options are grayed over when the notefile is closed.

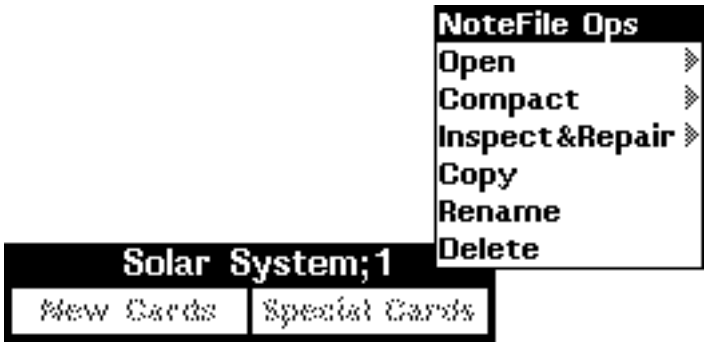


Figure 9-2. "Notefile Ops" menu from the Banner of a closed notefile.

Holding the left mouse button down in the Banner title bar when the notefile is *open* brings up the following "Notefile Ops" menu.

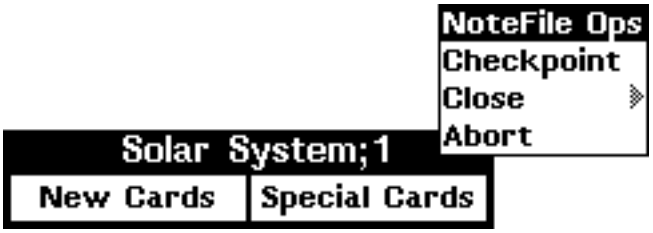


Figure 9-3. "Notefile Ops" menu from the Banner of an open notefile.

These two sets of options are a subset of those found on the "Notefile Ops" menu accessed by holding the left mouse button down on the MenuBox Icon's **Notefile** option.

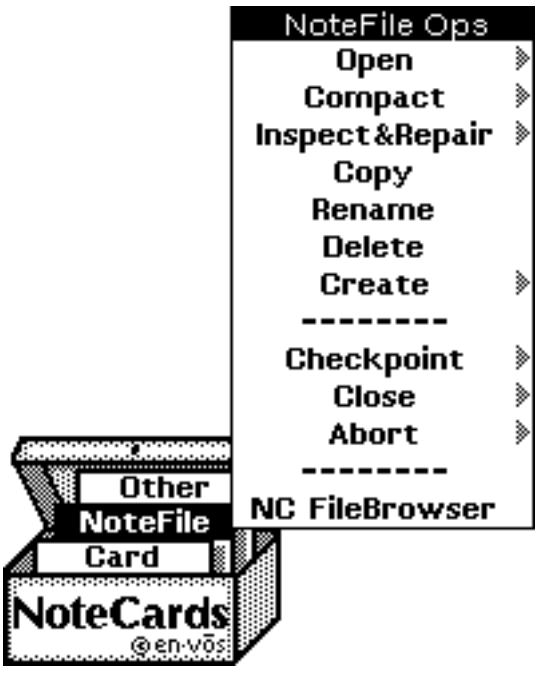


Figure 9-4. The MenuBox Icon's "Notefile Ops" menu.

All of these commands are discussed in Chapter 12, The MenuBox Icon.

Middle-Mouse-Button Title-Bar Menu

Holding the middle mouse button down in a notefile Banner's title bar brings up the following menu.

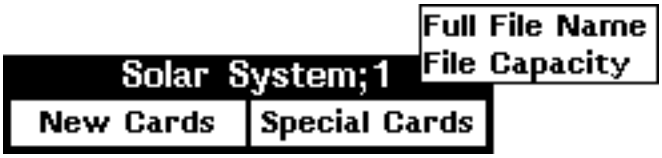


Figure 9-5. Middle-button menu in notefile Banner title bar.

Full File Name

Selecting the **Full File Name** option will bring up a window containing the full name of a file, including its directory path.

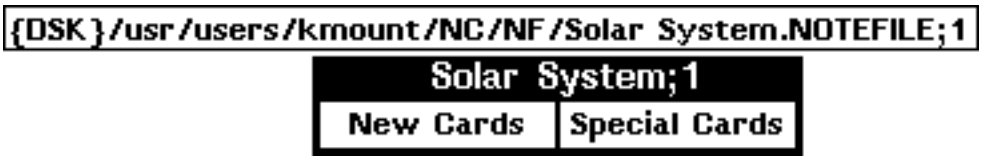


Figure 9-6. The full file name of the Solar System notefile.

Press any mouse button to continue. The window will close and you will be allowed to continue your work. This option is useful when you have two files with the same name, but different directories, on the screen and you need to be able to differentiate their Banners.

File Capacity

When the notefile is open, selecting the **File Capacity** option will attach a window to the top of the notefile Banner which will display the percentage and ratio of used cards in the notefile.

Figure 9-7. Notefile Banner with file capacity window open.

To close the file capacity window, hold the right button down in the file capacity window, not in the Banner title bar, and select the **Close** option off the window menu.

When 90% of the cards in a notefile have been used, the file capacity window will automatically open and warn you that the file is almost full and that it needs to be compacted. You must close a notefile before you can compact it. See Chapter 12,The MenuBox Icon, for instructions on compacting notefiles.

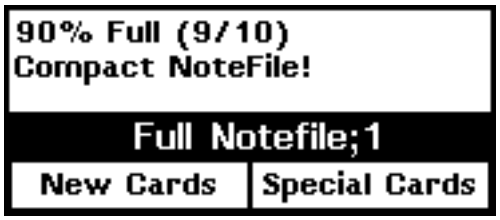


Figure 9-8. Notefile Banner with file capacity window warning to compact notefile.

If you try to create a new card when the notefile is 100% full, the system will display a message that the notefile is full and you will be asked if it is ok to checkpoint the notefile and make room for some number of new cards.

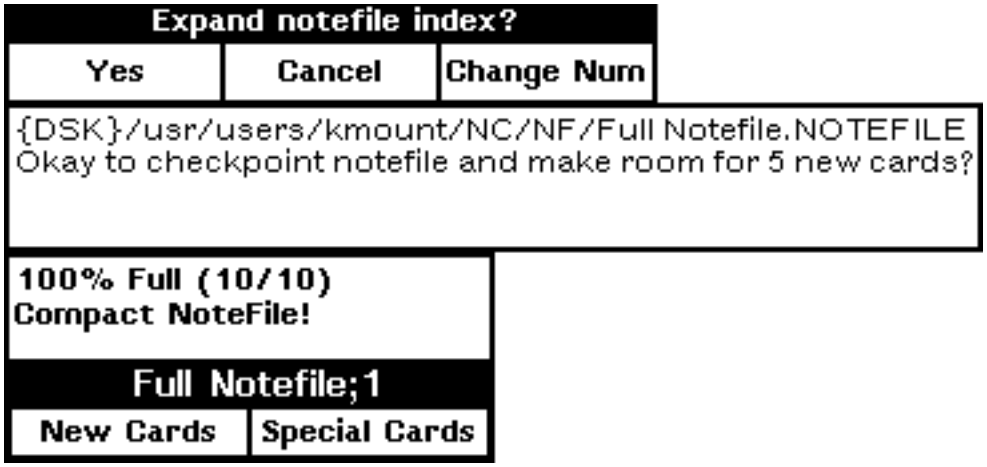


Figure 9-9. "Expand notefile index?" menu and prompt window.

If you select **Yes**, the system will save all the open cards, close the notefile, expand the notefile index by the amount you indicated, reopen the notefile, and create the card you requested. Note that the notefile is not compacted in this operation.

Cancel stops the operation. **Change Num** allows you to change the number of cards to expand the notefile by, after which you can choose the **Yes** option from the "Expand notefile index?" menu and proceed to save and expand the notefile.

Special Cards

Old notecards are accessed from the notefile Banner under the option **Special Cards**, (see Figure 9-1). Clicking the left mouse button in the **Special Cards** option brings up the "Table of Contents" FileBox card for the named notefile.

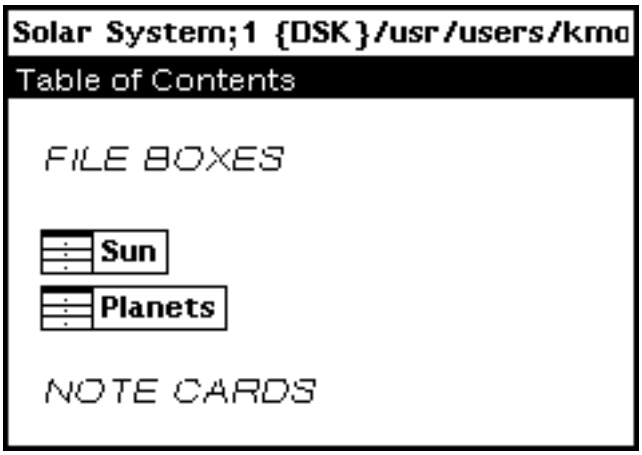


Figure 9-10. "Table of Contents" FileBox card for the "Solar System" notefile.

Holding down the middle mouse button in the **Special Cards** option brings up the "Special Cards" menu.

Figure 9-11. The "Special Cards" menu.

This menu allows you to access all of the top-level FileBoxes. Ultimately, all accessible cards are linked to one of these three FileBoxes.

Table of Contents FileBox

This is the top-level FileBox of each notefile. It is intended for storage of links to FileBoxes and other cards at the highest level of the information hierarchy.

To Be Filed FileBox

A temporary FileBox for cards that are not filed in any FileBoxes. If you close a card without specifying its FileBox, or if you close a notefile, without having specified the FileBoxes for every card, the system will place the cards without assigned FileBoxes in the "To Be Filed" FileBox.

If you later designate a FileBox for an unfiled card using the **Designate FileBoxes** or **Title/FileBoxes** commands from the card menu, this does not remove the card from the "To Be Filed" FileBox. To remove a card from the "To Be Filed" FileBox either delete its link or use the **Unfile From FileBoxes** suboption on the card menu of the newly filed card.

Figure 9-12. A card menu showing the **Unfile from FileBoxes** suboption selected.

As a general rule, it is a good idea to check the "To Be Filed" FileBox at the end of every session to verify that you have properly filed every card where you want it. You might also keep a copy of this FileBox on the screen so that you can tell at a glance if any cards have been filed there.

Orphans FileBox

A FileBox for cards whose last link from another FileBox has been removed.

Removing a link icon from the "Orphans" FileBox severs the card from the FileBox hierarchy but does not delete it from the notefile. Cards in this state are lost cards. The only means of retrieving a lost card is with a search card.

As a general rule, it is a good idea to check the "Orphans" FileBox at the end of every session to verify that you have properly filed every card where you want it. You might also keep a copy of this FileBox on the screen so that you can tell at a glance if any cards have been filed there.

New Cards

New cards are created from the notefile Banner under the option **NewCards**, (see Figure 9-1).

Clicking the left mouse button on the **NewCards** option immediately creates a specific type of card. The type of card created is determined by the value of the **Default Card Type** System Parameter. See Chapter 12, The MenuBox Icon, which describes how to set this parameter. The default value is "Text."

Pressing the middle mouse button over the **NewCards** option brings up the "Card Types" menu.

Figure 9-13. The "Card Types" menu.

Selecting one of these options creates a card of that type.

Figure 9-14 graphically represents the notecard types and their range of use. Text, Graph, and Sketch cards help you to collect and express your ideas. FileBoxes allow you to express the relationships between your ideas and organize them into coherent structures. Browser cards provide a broad range of functionality, and are a means of representing as well as retrieving information. LinkIndex cards build sorted lists of cards. Search cards perform searches on card titles. And, Document cards create a linear document from your linked cards.

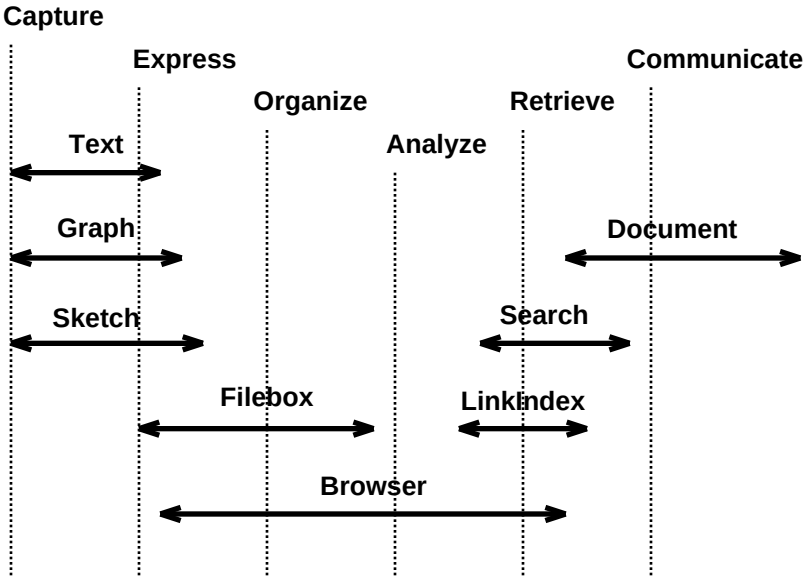


Figure 9-14. Card types and their range of use.

User Cards and System Cards

Cards can be broadly divided into two categories. User cards are those cards for which you create the contents. System cards are cards where the contents are built by the system for you. Chapter 10 discusses user cards; Chapter 11, system cards.

In Figure 9-13, the top four card types (Text, FileBox, Sketch, and Graph) are user notecards. The bottom four card types (Browser, Search, LinkIndex, and Document) are system notecards.

Text-, Sketch-, and Graph-Based Cards

All cards are based on one of three editors, the text editor TEdit, the Sketch editor Sketch, or the Graph editor Grapher.

Text, FileBox, Search, LinkIndex, and Document cards are all based on the text editor TEdit. Hence they all behave in essentially the same way and are referred to as text-based cards.

The Sketch card is the only card based on the Sketch editor. This card is referred to as a sketch-based card.

Graph and Browser cards are both based on the graph editor Grapher. These cards behave in basically the same way and are referred to as graph-based cards.

The Card Menu

All cards have a left-button title-bar menu called the card menu. This menu is the same for all cards except FileBox cards. Each card also has a middle-button title-bar menu which is frequently card-specific. These menus are discussed under their respective card types in Chapter 10, User Cards, and Chapter 11, System Cards.

The Standard Card Menu

Figure 9-15. The note card menu

This menu is found on all cards except the FileBox card which has a slight ly different version.

Edit Property List

The property list editor allows you to associate property-value pairs with cards. For example, you might want to attach a Certainty Value property (and value) to every card. The value of this property

would indicate the degree to which you believe the information contained on the card is true.

You bring up the "Edit Property List" window by selecting the **Edit property List** option off the card menu.

Figure 9-16. "Edit Property List Window"

You can only change the property list of a card interactively, from the menu shown in Figure 9-17. Holding down the left or middle mouse button in the "Edit Property List" window's title bar brings up the following edit menu for property lists. Select commands from this menu before releasing the button.

Figure 9-17. Menu from "Edit Property List" window title bar.

Add New Property Adds user-defined properties to the property list of this card. Type in the property name and the value when prompted. Next, select one of the properties already in the list. The new property will be inserted in front of the selected property. Properties are displayed in bold type and values are displayed between brackets in regular type. You can abort the add process by hitting the Stop key.

In Figure 9-16, **Source** and **Certainty Factor** are the properties and "John Smith" and ".9" are the values. You can edit the values directly with the mouse and keyboard as you would any text string.

Delete Selected Property Deletes a property from the property list of this card. After choosing this option click on the property to be deleted with the left mouse button. Properties are displayed in bold type. Once you have started the delete process there is no way to abort it, but you can hit the Undo key to undo the deletion.

Quit w/o Saving Changes Closes the display without saving any of the current changes made using **Add New Property** or **Delete Selected Property**.

Quit - Saving Changes Closes the display saving all current changes made using **Add New Property** or **Delete Selected Property**.

Show Links

Displays, in the "List of Links" window above the card, a list of all links to and from other cards. Links are represented by link icons. Selecting an icon in this window with the left mouse button displays the card referenced by that link icon.

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

For more on the **Show Links** option, see the section "Viewing Local and Global Links" in Chapter 8, Links.

Show Info

Brings up the "Card Attributes" window. This window displays the card's type, the dates each of the card parts was last changed, and a list of dates chronicling when the card was updated.

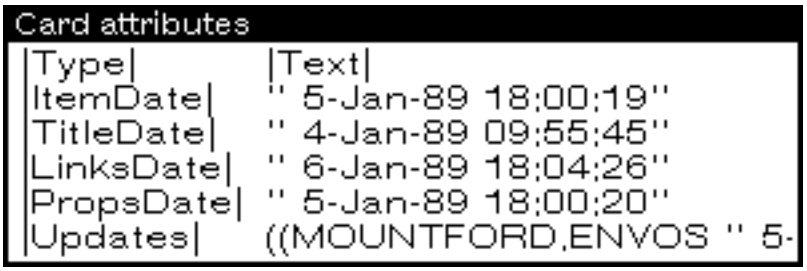


Figure 9-18. The "Card attributes" window.

The **Show Info** option also automatically brings up the notefile indicator on the card.

Indicate NoteFile This pull-across subitem on the **Show Info** option brings up the notefile indicator without bringing up the "Card Attributes" window.

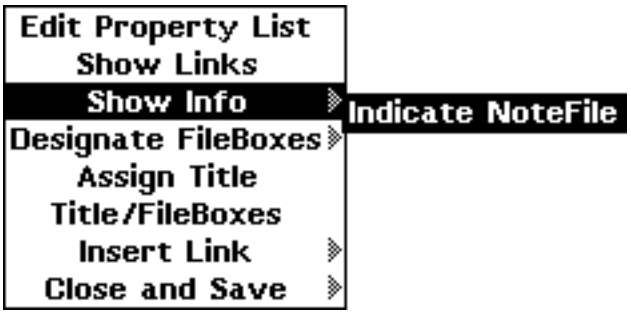


Figure 9-19. The **Indicate Notefile** submenu off **Show Info**.

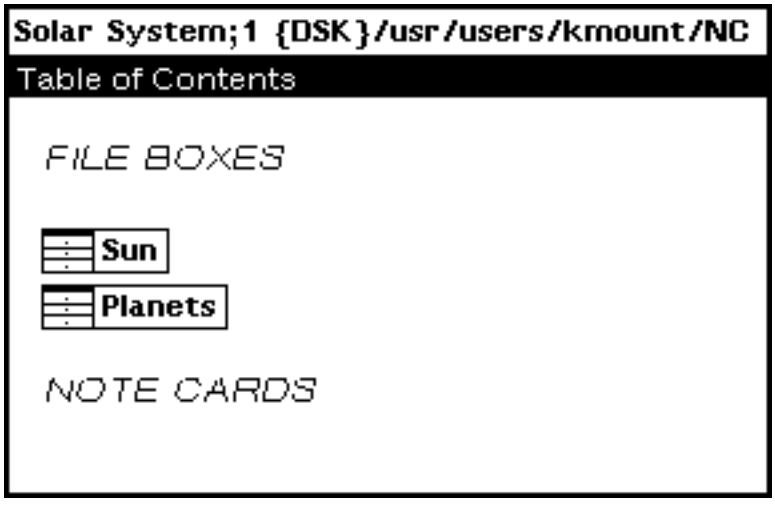


Figure 9-20 A card with its notefile indicator window open on the top of the card.

To close the "Card Attributes" window, just choose the **Close** item on the standard window title bar menu of the "Card Attributes" window. Closing the "Card Attributes" window does not close the notefile indicator window, which must be closed separately using its standard window title bar menu.

Designate FileBoxes

Pops up a prompt window above the card asking the user to file the card in one or more FileBox cards.

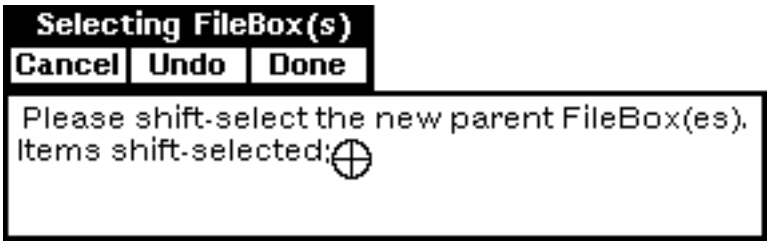


Figure 9-21. The "Selecting FileBox(es)" prompt window.

Associated with the prompt window is a three-item menu. Select **Done** from this menu or type a carriage return after shift-selecting the FileBoxes to file the card in. Select **Cancel** to abort the filing operation. Choose **Undo** to remove the last selected FileBox from the list of new parent FileBoxes.

Figure 9-22. A card menu showing the **Unfile from FileBoxes** suboption selected.

Unfile from FileBoxes

Unfiles the card from the FileBoxes you select. You unfile the card by shift-selecting the FileBox card IDs from their title bars. The system tells you when you have made an invalid selection. The prompt window menu options are the same as those for **Designate FileBoxes**, shown in Figure 9-21.

Assign Title

Assign Title allows you to assign a title to a card or edit an existing title.

You can reposition the cursor in the edit string with the mouse. You can also use the mouse to delete pieces of the string by using the left mouse button to position the cursor at the beginning of the piece you want to delete and sweeping out the portion to delete while holding down the right mouse button. When you release the right mouse button the selected text will be deleted.

To cancel the delete operation, still holding the right mouse button down, move the mouse cursor outside the prompt window and release the right mouse button.

You can undo the delete operation by hitting the Undo key.

Title/FileBoxes

Selecting **Title/FileBoxes** is identical to selecting **Assign Title** and **Designate FileBoxes** in sequence.

Insert Link

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.

Before execution of this command, select the point, in the body of the card, where you want the link icon to appear.

When you select this menu option, a menu pops up displaying a list of link types currently available in the notefile. Link types are notefile specific. Specify the type of link by selecting one from this list or select **--New Link Type--** to create a new type. This new type of link is added to the notefile and becomes a choice in its list of link types. It is not possible to assign a system-reserved type of link to a user-specified link. Select ****CANCEL**** from the menu to abort the **Insert Link** command or just click outside the menu.

After a type has been designated, a prompt window and menu pops up above the card asking the user to choose the destination card for the link. The user has the option of selecting an existing card or of creating a new card as the destination card by selecting **New Card** from the menu with the left mouse button. If this option is chosen, a menu of card types pops up from which the user selects the desired type of card. Again, selecting **Cancel** from the "Selecting Note Card" menu aborts this command.

At this point, the link icon will be inserted at the flashing caret in the body of the source card.

For more on **Insert Link** and its submenu options **Insert Links**, **Add Global Link**, and **Add Global Links**, see Chapter 8, Links.

Close and Save

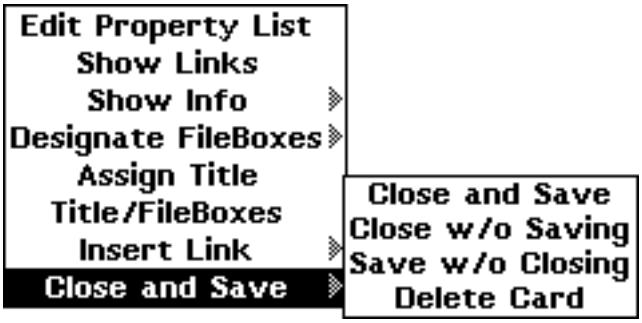


Figure 9-23. The submenu off the **Close and Save** option

Close and Save Saves the card in the notefile before closing the card. If the **Force Titles** and **Force Filing** system parameters are set to "Yes," NoteCards will request that you provide a title and a FileBox for the card before closing it. If you do not wish to provide this information,

simply type a carriage return to both requests. In this case, the card will be titled "Untitled," and be filed in the "To Be Filed" FileBox.

Close w/o Saving Closes the card without saving it, in the notefile. Any changes you have made to the card since the last save are lost. This option is useful if the card contents are mistakenly lost or scrambled while you are editing. This command may ask for confirmation. Type a carriage return or click on the **Yes** option to confirm. Type "n" or "N" and a carriage return or click on the **No** option to cancel.

Save w/o Closing Saves, to the notefile, all changes you have made to the card without closing the card. Saving updates the card in the notefile but does not update the notefile index. If there is a system crash you will have to perform an **Inspect & Repair** operation to recover the card contents.

Delete Card Permanently deletes the card from the notefile and all its links to and from other cards. Because this deletion is irreversible, the user is asked to confirm before the delete command is executed. Type a carriage return or click on the **Yes** option to confirm. Type "n" or "N" and a carriage return or click on the **No** option to cancel. Note that a deleted card cannot be retrieved with a search card as a lost card can.

The FileBox Card Menu

The FileBox card menu differs in only two items, **Add Global Link**, and **Put Cards Here**.

Add Global Link

The **Add Global Link** and **Add Global Links** options are the same global link options found on the submenu of the **Insert Link** option. **Insert Link** appears in this position on this menu for all other cards. These two options are discussed extensively in Chapter 8, Links.

Figure 9-24. The FileBox card menu with the **Add Global Link** submenu.

Put Cards Here

The **Put Cards Here** menu option is a specialization of the **Insert Links** option for FileBox cards. It allows you to file more than one card in a FileBox in a single operation. Selecting this option brings up the "Selecting cards to file" prompt window into which you can shift select the card IDs from the title bars of all the cards you want to file in this FileBox card.

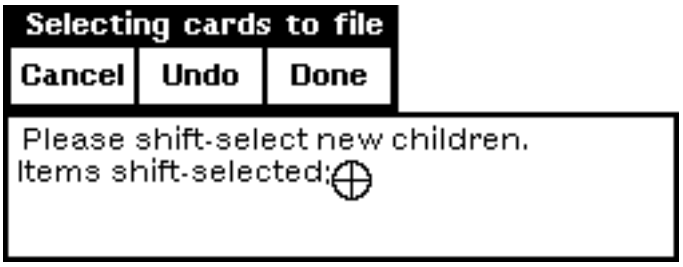


Figure 9-25. The "Selecting cards to file" prompt window.

[This page intentionally left blank]

10. USER CARDS

Cards can be broadly divided into two categories. User cards are those cards for which you create the contents. System cards are cards where the contents are built by the system for you. This chapter discusses user cards.

This chapter explains :

How to use Text cards.

How to use FileBox cards.

How to use Sketch cards.

How to use Graph cards and the graph editor.

How to use the bit map editor.

Text Cards

The Text card is based on TEdit, a versatile editor and text formatter. This entire document was produced using TEdit. For a detailed discussion of how this editor works, see *A User's Guide to TEdit*. Text cards allow you to include sketches, graphs, and bit maps. To learn how to manipulate sketches, see *A User's Guide to Sketch*. For graphs and bit maps see the sections below on the Graph card and the bit map editor.

The Text-Card Menu

The text-card menu is the same as the TEdit menu with one additional option separated from the others by a dashed line.

Figure 10-1. The text-card menu.

You use the **Restart Editor** command when the contents of the Text card are incorrectly displayed on the screen.

For all the other menu items, see *A User's Guide to TEdit*.

FileBox Cards

The FileBox card, like the Text card, is based on TEdit. For a detailed discussion of how this editor works, see *A User's Guide to TEdit*. Text-based cards allow you to include sketches, graphs, and bit maps. To learn how to manipulate sketches, see *A User's Guide to Sketch*. For graphs and bit maps see the sections below on the Graph card and the bit map editor.

A FileBox is a card that contains links to other cards including other FileBox cards. All cards, including FileBox cards, can be filed in one or more FileBoxes. Every card, including FileBox cards, except the top level Special FileBox Cards, is contained in at least one other FileBox. Whereas other cards may be linked together to form an arbitrary network, the set of FileBoxes forms a strict hierarchy This is to say that no child FileBox is allowed to have its parent FileBox as a child. In short, no circular linkages.

FileBoxes are meant to hold all cards relating to some given topic. A FileBox typically contains both links to subFileBoxes, which contain any cards relevant to the subtopics of the main topic, and links to other card types, which contain information relevant to the main topic. For example, the screen image below shows a FileBox containing both FileBoxes and other note cards.

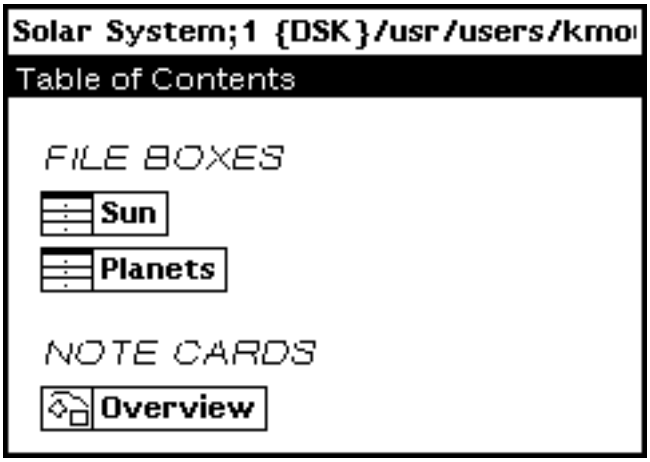


Figure 10-2. The Solar System FileBox containing subFileBoxes for its subtopics and a Sketch card dealing with the main topic, the Solar System.

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 you to represent the interconnections between various ideas or pieces of information, independent of any categorization into topic areas.

The markers FILE BOXES and NOTE CARDS help differentiate what kinds of cards are filed in the FileBox. In addition, since FileBoxes are text-based cards, anything you can do with a text card, you can also do with a FileBox. This means that you can, for example, insert your own labels or short lines of commentary to break up the links into subgroups.

Suggested FileBoxes and Note Cards

You may find it helpful to create the following types of general FileBoxes

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 cards.

Read Me

A note card in the top level FileBox giving global information about the notefile for first time browsers.

Active Cards

A FileBox kept at the top level of the FileBox hierarchy containing FileBoxes and note cards that represent work in progress and are thus frequently accessed. A Sketch card containing links to these FileBoxes and note cards is another method of organizing active cards, using spatial cues as a way of representing structure.

The FileBox-Card Menu

This menu is the same as the Text-card menu. See the section immediately above on Text cards as well as *A User's Guide to TEdit*.

System Parameters Affecting FileBoxes

FileBoxes have two system parameters associated with them, **Markers In FileBoxes** and **Alphabetized FileBox Children**. For a complete discussion of these parameters see Chapter 13, System Parameters.

Sketch Cards

The Sketch card is based on Sketch, a sophisticated graphics package. For a detailed discussion of how Sketch works, see *A User's Guide to Sketch*. Sketch cards allow you to include graphs, and bit maps. To learn how to manipulate graphs and bit maps see the sections below on the Graph card and the bit map editor.

The Sketch-Card Menu

This menu is the same as the Sketch editor menu. For a detailed discussion of this menu's functionality, see *A User's Guide to Sketch*.

Figure 10-3. The Sketch card menu.

System Parameters Affecting Sketch Cards

Sketch cards have one system parameter associated with them, **Attach Sketch Menu**. For a complete discussion of this parameter see Chapter 13, System Parameters.

Graph Cards

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

The Graph-Card Menu

The Graph card, like the Browser card, is based on the graph editor Grapher. To make the graph-card menu and terminology more consistent with the browser-card menu and terminology the

Graph card presents a slightly different menu to you than Grapher does. However, since the functionalities are virtually identical, both the graph-card and Grapher menus are discussed in parallel below. The graph-card menu-option titles are left justified while the Grapher menu-option titles are right justified.

You display the graph-card menu by depressing the right mouse button in the body of the card. Select the desired command before releasing the button.

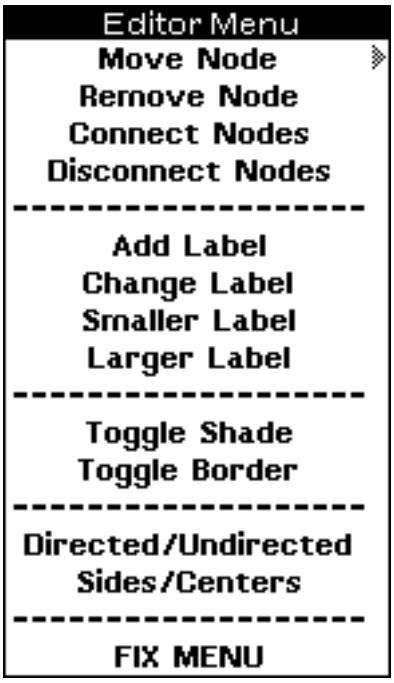


Figure 10-4. The graph-card menu.

The Grapher Menu

You will probably encounter the Grapher menu only if you save your document cards to TEdit files and edit a graph-card graph from within the TEdit document. You can safely skip over this section and still understand the Graph card.

To edit a graph from a TEdit document you must first select the **Edit graph** option from the one-item menu which appears when you hold any mouse button down in the graph region.

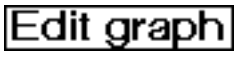


Figure 10-5. The "Edit graph" menu.

When you select this option, Grapher opens a window containing the graph. Hold the middle mouse button down in this window to make the menu shown in Figure 10-6 appear. Use the left mouse button to move nodes. When you are done editing the graph, select the **STOP** option.

While you are editing a graph, Grapher captures the type-in process and does not allow you to do anything other than mouse operations. To free the type-in process, select the **STOP** option.

There is no simple way to abort out of the Grapher editor and throw away all the changes you have made to the graph. For this reason, if you are going to edit the graph extensively, we recommend that you use shift-select to copy the original graph in place and call the Grapher editor on the copy. In a worst case scenario, you can try hitting the STOP key or typing CONTROL-E.

Figure 10-6. The Grapher menu.

Prompts for information or confirmation are given in the card prompt window for Graph cards and appear in the system prompt window for Grapher. General information is printed to the system prompt window for both Graph cards and Grapher.

The Graph-Card and Grapher Menu Options

Move Node **Move Node**

Moves a node and connections to a new position. After selecting this option, point to the node you want to move, press and hold the left mouse button, move the node to its new position, and release the mouse button.

Move Node has three options on a submenu. These same three options appear on the Grapher **Move Node** submenu.

Figure 10-7. The **Move Node** submenu.

Move Single Node Functions exactly as **Move Node** does.

Move Node & SubTree Moves a selected node and all subnodes which it is connected to. This operation does not move any super nodes of the selected node. That is, nodes which are connected to the selected node as opposed to nodes which the selected node is connected to.

Figure 10-8. A node and subtree selected to be moved.

Figure 10-8 shows what happens, when you select **Move Node & SubTree** and then hold the left mouse button down on node B. Nodes B, C, and D are selected to be moved, but not node A. Grapher keeps track of the nodes where each connection starts and ends, and in this case the connections run from A to B to C to D. So node A is not in the subtree of node B and hence is not moved. See the **Directed/Undirected** option below for more information.

Move Region Allows you to sketch out a region of the graph which you want to move. **Move Region** does not pay attention to the graph hierarchy, it only pays attention to the area you sweep out in the Graph card.

Remove Node **Delete Node**

Removes a node from the graph. Select the node to be deleted with the left mouse button. The card prompt window will prompt you for confirmation.

Connect Nodes **Add Link**

Draws a connection between two nodes. Select the "from" node and then the "to" node with the left mouse button when prompted. If a second overlapping connection is made running in the opposite direction between the same two nodes, the lines representing the connections, between those two nodes, will not be visible You can make them visible by choosing the directed display option.

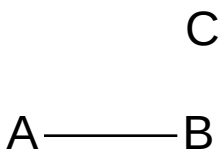


Figure 10-9. A graph, with connections from A to B to C and a third connection from C to B, displayed using the **Undirected** option.

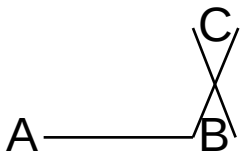


Figure 10-10. A graph, with connections from A to B to C and a third connection from C to B, displayed using the **Directed** option.

Disconnect Nodes **Delete Link**

Removes a connection from between two nodes. Select the "from" node and then the "to" node with the left mouse button when prompted.

Add Label **Add Node**

Pops up a window prompting you to type in a label name followed by a carriage return. The label will appear next to the cursor within the graph card. Position the new label by moving the cursor to where you want the label to appear. Plant the label by clicking any mouse button. This operation can be cancelled by typing a carriage return before typing any other characters to the prompt window.

Change Label **Change Label**

Allows you to change a label. **Change Label** first waits for you to select, with the left mouse button, the label you want to change. It then pops open a window prompting you to type in a new label name followed by a carriage return. The new label immediately replaces the old label, preserving font, position, and connections. You can cancel this operation by clicking outside a node, or by typing a carriage return before typing any other characters to the prompt window.

Smaller Label **label smaller**

Decreases the font size of the selected node. Repeat this command as many times as necessary to achieve the font size you want.

Larger Label **label larger**

Increases the font size of the selected node. Repeat this command as many times as necessary to achieve the font size you want.

Toggle Shade **<-> Shade**

Inverts the shade around the selected node. 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.

Toggle Border **<-> Border**

Draws a rectangular border around the selected node. Select the node to have a border drawn around it with the left mouse button. To remove a border, re-apply this option.

Directed/Undirected **<-> Directed**

A graph is stored as a directed lattice. Connections always run from one node to some other node.

The **Directed** option makes the flow of the connections explicit in the presentation of the graph. When you select the **Directed** option, connections prefer to run from the left side of the parent node to the right side of the child node when you have **Sides** selected. When you have the **Centers** option selected, connections prefer to run from the bottom center of the parent node to the top center of the child node.

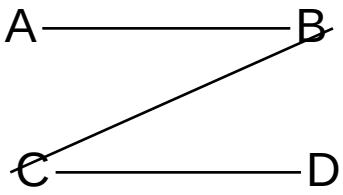


Figure 10-11. A **Directed** graph which explicitly shows the flow of connections from A to B to C to D, with the **Sides** option selected.

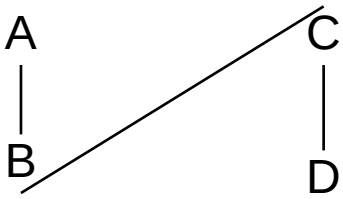


Figure 10-12. A **Directed** graph which explicitly shows the flow of connections from A to B to C to D, with the **Centers** option selected.

The **Undirected** option draws the graph without regard to the flow of connections. The lines are drawn starting and ending on the sides of the nodes closest to each other.

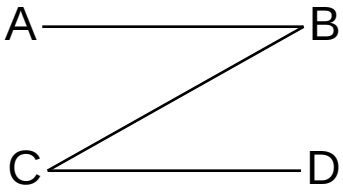


Figure 10-13. An **Undirected** version of Figure 10-11.

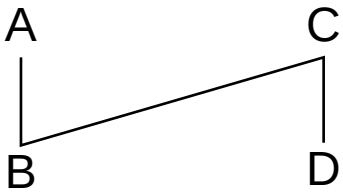


Figure 10-14. An **Undirected** version of Figure 10-12.

Sides/Centers **<-> Sides**

The sides mode predisposes the graph editor to make the left and right sides of the nodes the connection points for lines.

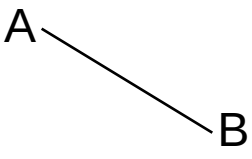


Figure 10-15. A graph drawn favoring sides.

The centers mode predisposes the graph editor to make the top and bottom centers of the nodes the connection points for lines.

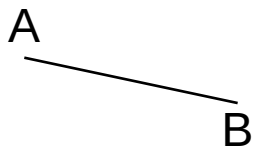


Figure 10-16. A graph drawn favoring centers.

FIX MENU

Attaches the graph-card menu to the right side of the Graph card.

STOP

Only appears on the Grapher menu, not on the Graph card menu. **STOP** exits the Grapher editor saving all your changes.

The Bit Map Editor

The bit map editor allows you to manipulate bit maps that have been inserted in Text or Sketch cards as well as TEdit and Sketch files. It is automatically invoked when the bit map area is selected.

Inserting Bit Maps into Cards

The method for inserting bit maps into text-based cards differs slightly from that for inserting them into sketch-based cards. Each procedure is discussed below.

You cannot insert bit maps into graph-based cards.

Text-based Cards

Inserting a bit map into a text-based card involves several steps. First, position the caret cursor where you want the bit map to appear in the destination card, or TEdit window, by clicking at that position with the left mouse button. Second, depressing the Copy key or either of the Shift keys, hold the right mouse button down somewhere in the background and select the **Snap** option from the single item menu which will appear.

Figure 10-17. The single-item "Snap" menu.

At this point the mouse cursor changes to look like this, . This is the prompt asking you to sweep out an area of the screen to be made into a bit map. Third, press and hold the left mouse button while you sweep out a region of the screen. When you release the left mouse button, the bit map will be transferred to the designated card or edit window.

If you need to adjust the area you are sweeping out, do the following. Hold down the right mouse button, in addition to the left mouse button, to bring up the forceps prompt, This prompt

allows you to change corners so that you can adjust the size of your bit map in all directions.

Sketch-based Cards

The procedure for inserting bit maps into sketch-based cards differs in only one respect from that for text-based cards. The sketch-based card must be the active card, which is to say you must click in it so that it has the type-in process to mark it as the destination for the bit map. However, when you do this, you are not indicating the insertion point for the bit map. In sketch-based cards, positioning the bit map is done last. After you have swept out a region of the screen to include as a bit map in the sketch, move the mouse cursor back into the Sketch card. When you enter the Sketch card, the snapped bit map will appear attached to your mouse cursor, and you can position it by clicking the left mouse button.

Bit Map Operations

Moving the mouse cursor into a bit map and holding down the left or middle mouse button brings up the "Operations on bitmaps" menu.

Figure 10-18. The "Operations on bitmaps" menu.

Change Scale

Changes the scale or size of the bit map. Giving a scale of 2 doubles the size of the bit map; a scale of .5 halves the size of the bit map. You achieve the best results shrinking or enlarging a bit map when you change the scale by evenly divisible amounts. For example, 4, 2, 1, .5, or .25.

Hand Edit

Invokes the bit map editor on the bit map. The bit map editor is described in detail below.

Trim

Trims the white columns and rows from all four edges of the bitmap. This is a very useful operation to remove any extraneous white space from around the bit map. Position the image that you are taking a snap of on a white background to take the greatest advantage of this option.

Reflect Left-to-right

Flips the bitmap about its vertical centerline.

Reflect Top-to-bottom

Flips the bitmap about its horizontal centerline.

Reflect Diagonally

Flips the bitmap about its X=Y diagonal so that the resulting bit map is reversed and lying on its right side. The same effect can be achieved by performing a **Reflect Left-to-Right** followed by a **Rotate Right**.

Rotate Left

Rotates the bit map by 90 degrees in a counterclockwise direction so that the resulting bit map is lying on its left side.

Rotate Right

Rotates the bit map by 90 degrees in a clockwise direction so that the resulting bit map is lying on its right side.

Expand on Right

Adds white space to the right of the bit map. You specify the width of the white space in pixels using the number pad. Select **ok** when you are done.

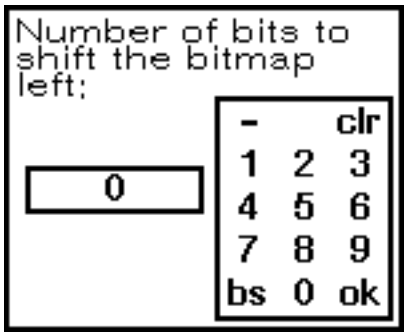


Figure 10-19. The number pad.

The number pad is used much like a simple calculator to enter numbers. The **ok** button returns the number to the system. **bs** deletes the last digit you entered. **clr** resets the input to "0." You abort the operation by setting the input value to"0" and selecting **ok**.

You enter a negative number by first entering the digits and then selecting the minus sign. Entering a negative number removes that many pixels from the right side of the bit map.

Expand on Left

Adds white space to the left of the bit map. You specify the width of the white space in pixels using the number pad. Select **ok** when you are done. See the **Expand on Right** option for more detail.

Expand on Bottom

Adds white space to the bottom of the bit map. You specify the width of the white space in pixels using the number pad. Select **ok** when you are done. See the **Expand on Right** option for more detail.

Expand on Top

Adds white space to the top of the bit map. You specify the width of the white space in pixels using the number pad. Select **ok** when you are done. See the **Expand on Right** option for more detail.

Switch Black & White

Inverts all of the pixels in the bit map; exchanges black for white and white for black.

Add Border

Adds a border to the bit map. The system prompts you for the width of the border using the number pad described above. It then prompts you for the texture of the border with the texture bit map editor.

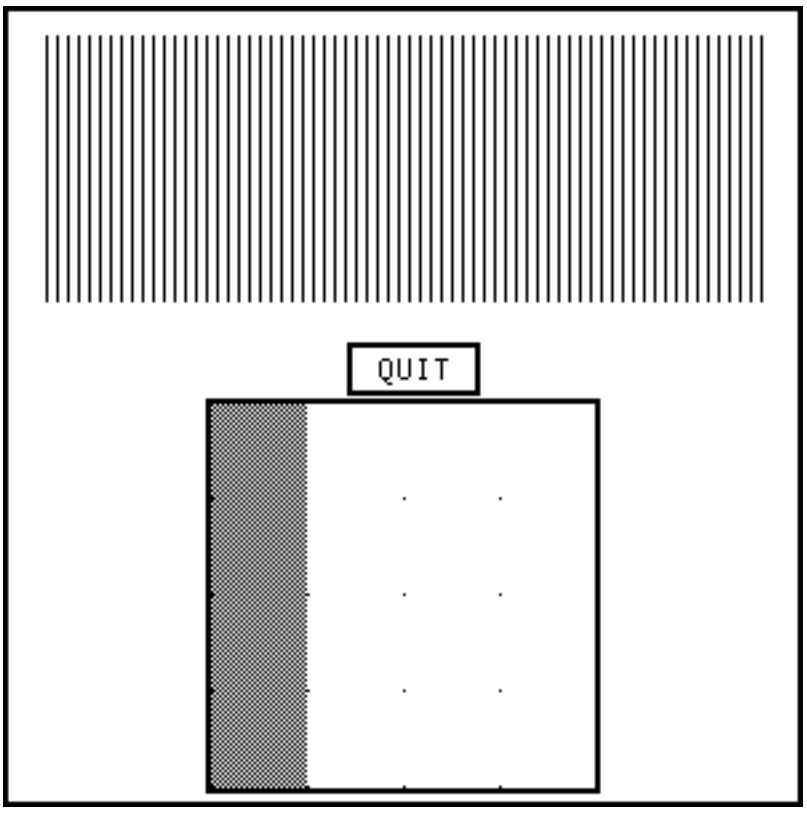


Figure 10-20 The texture bit map editor.

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

edit array. Clicking the left mouse button in the edit grid turns a pixel on; clicking the middle button turns a pixel off. Select the **Quit** option when the texture looks the way you want it to. The texture will then appear as the border around the bit map.

To abort this operation, select **clr** followed by **ok**, on the number pad.

There is no simple way to abort this operation once you have brought up the texture bit map editor. However, you can turn all the pixels off (set them to white space) and select **Quit**. Then selecting **Trim** from the "Operations on bitmaps" menu should return the bit map to its previous condition.

The Bitmap Editor

The editing window has three active areas, a grid edit area in the lower part of the window, a display area in the upper left part, and a gray bar in the upper right.

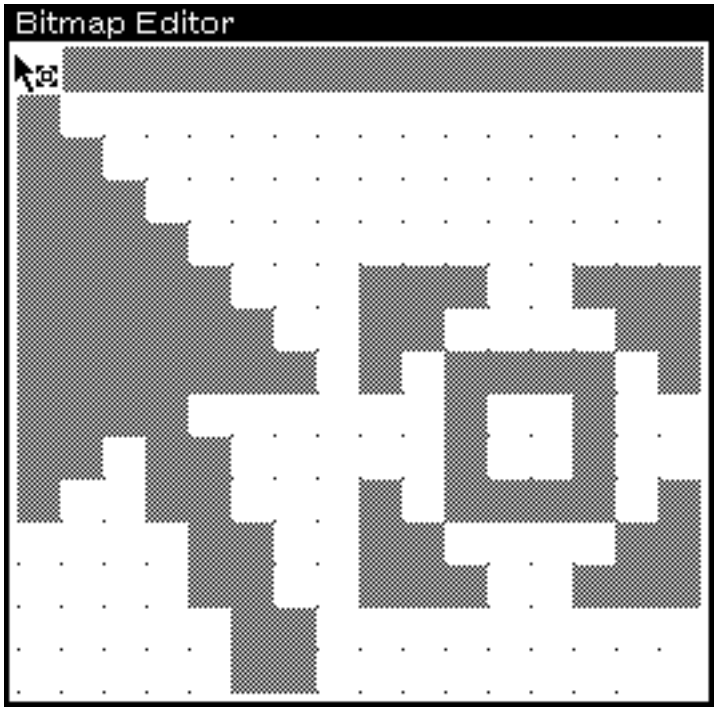


Figure 10-21. The "Bitmap Editor" display.

In the edit area, the left button adds points and the middle button erases points. The display area shows the actual size and form of the bit map. The gray bar provides access to the "Bitmap Editor" menu.

The right mouse button brings up the normal window menu in all areas of the window.

If the bit map is too large to fit in the edit area, you can change the portion which can be edited by scrolling up and down in the left margin, and left and right in the bottom margin. Pressing the middle mouse button while in the display area brings up a menu that allows you to make a global placement of the portion of the bit map which can be edited. If you want to see more of the bit map

you are editing, you can reshape the window to make it larger, or you can use the **GridSize** command, described below, to reduce the bit size in the edit area.

Whenever you press the left or middle mouse button down with the cursor inside the display area or the gray bar, the section of the bit map that is currently in the edit area is shown in reverse video. Pressing the left button while in the gray bar puts the lower left 16 x 16 bit section of the bit map into the mouse cursor for as long as the left button is held down.

Pressing the middle button while in the grey bar or in the title bar brings up the "Bitmap Editor" menu.

Figure 10-22. The "Bitmap Editor" menu.

Holding the middle button down over a command results in an explanatory message being printed in the system prompt window.

Paint

Puts the current bit map into a window and calls the paint command on the bit map. You use the left mouse button for drawing and the right for erasing. The paint command implements drawing with various brush sizes and shapes but only on an actual sized bit map. You set brush characteristics and exit paint by pressing the right mouse button and selecting the appropriate command from the paint command menu. When you exit, you will be asked whether or not the changes you made while in Paint mode should be placed in the current bit map. **Paint** is particularly useful for erasing or filling in large regions in bit maps. See the section "The Window Menu" in Chapter 7, The User Interface for a detailed discussion of all the paint menu options.

Figure 10-23. Paint command menu.

ShowAsTile

Tesselates the current bit map in the gray bar. This is useful for determining how a bit map will look if it were made the display background. The tiled display does not automatically change as

the bit map changes. To update it, use the ShowAsTile command again.

Grid On/Off

Turns the editing grid display on or off.

GridSize

Allows you to specify the size of the editing grid. When you select this option, a number menu appears, giving you a choice of several point sizes for the grid.

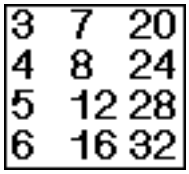


Figure 10-24. The number menu.

When you select a size, the editing portion of the bit map editor is redrawn. A smaller size allows you to edit more of the bit map without scrolling, while a larger size makes it easier for you to turn individual bits on and off. The original size is chosen heuristically. It is typically about 8. Clicking outside the number menu aborts this operation.

Reset

Sets all or part of the bit map to the contents it had when you originally called the bit map editor. When you select this option, a second menu appears giving you a choice between resetting the entire bit map or just the portion that is in the edit area.

Figure 10-25. The "RESET how much?" menu.

This second menu also acts as a confirmation, since clicking outside of this menu results in no action being taken. Note that if the entire bit map appears in the edit area the menu only has the **WholeBitmap** option.

Clear

Sets all or part of the bit map to white space. As with the Reset command, a second menu gives you a choice between clearing the entire bit map or just the portion that is in the edit area.

Figure 10-26. The "CLEAR how much?" menu.

Cursor

Sets the cursor to the contents of the lower left part of the bit map. This operation next prompts you to specify the new cursor's active pixel. You do this by clicking somewhere in the lower left 16 x 16 portion of the grid. Cursors created this way are typically very short lived. This option is intended for people extending the NoteCards environment. We recommend that non-programmers do not use this option.

OK

Copies the edited bit map image into the original bit map, exits the bit map editor, and closes its edit window. The image you modify using the editor is a copy of the original bit map. Unless you exit the bit map editor via **OK**, no changes are made to the original bit map.

Abort

Exits the bit map editor without making any changes to the original bit map. Contrast with **OK**.

[This page intentionally left blank]

12. THE MENUBOX ICON

The MenuBox Icon is the main interface to NoteCards. Through this icon you access all the system NoteCards menus.

This Chapter explains:

How to create notefiles.

How to perform basic operations on notefiles.

How to recover notefiles when they are damaged.

How to perform structure operations on notefiles.

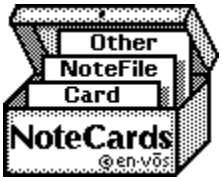


Figure 12-1. The MenuBox Icon.

Notefile Options

It is through the Notefile-Operations menu that you interact with notefiles. The word "Operations" is usually abbreviated as "Ops" and, as a result, this menu is usually refered to as the "Notefile Ops" menu.

You access the "Notefile Ops" menu by holding the left mouse button down in the **Notefile** option of the MenuBox Icon. The menu has three regions, separated by dashed lines. The top region contains operations you can only apply to closed notefiles. The middle region contains commands you can only apply to open notefiles. The bottom item, NC FileBrowser, does not share these limitations. It provides you with a notefile interface.

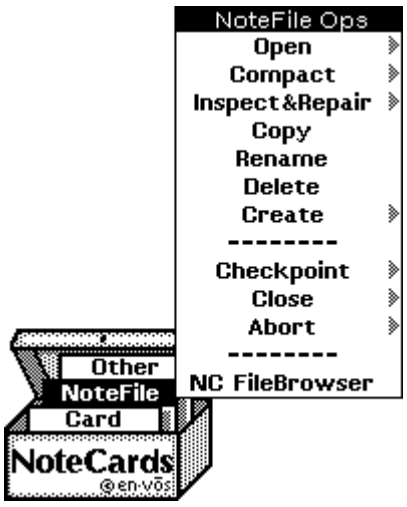


Figure 12-2. The MenuBox Icon's "notefile Ops" menu.

Open

Opens an existing notefile. If you have not previously opened any files, the system prompts you to type in a file name. The file name is not case sensitive. If you provide just a file name with no path, the system looks in the connected directory for the notefile. If NoteCards does not find the file there, it asks you if you want to create the file.

The system maintains a list of all notefiles you open. Subsequently, when you select **Open**, these files will be presented to you in menu format. If the file you want to open is not on the menu, select the **--Other Notefile--** option.

There are two submenu options for Open, **Open Read/Write** and **Open Read-only**.

Figure 12-3. The **Open** submenu.

Open Read/Write

Is the same as selecting **Open** from the top-level menu.

Open Read-only

Opens the file but does not allow you to make any changes to the file. When you open a file read-only, the file name appears in its Banner with a read-only prefix (RO:).

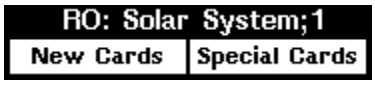


Figure 12-4. A notefile opened read-only with its Banner showing the read-only prefix "RO:"

If the notefile is damaged the following menu appears.

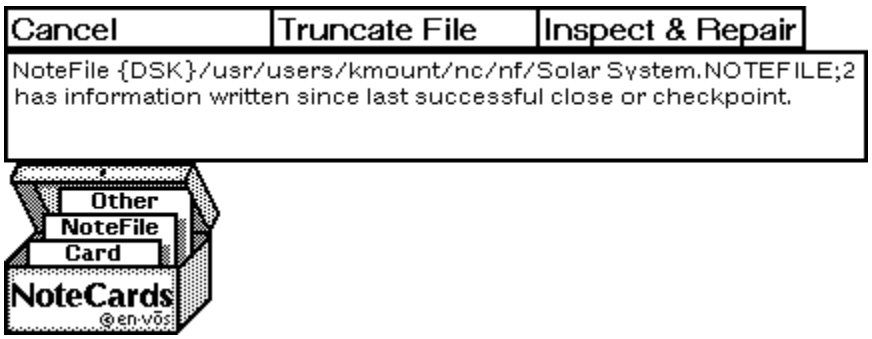


Figure 12-5. The menu which appears when you open a damaged notefile.

Cancel causes the open notefile operation to be aborted.

Truncate File deletes everything after the checkpoint pointer.

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. For more details, see the

Inspect & Repair and Appendix A, Notefile Concepts and Appendix B, The Notefile Inspector.

Compact

Compresses a notefile by deleting old information. NoteCards never actually overwrites any information in the data area of a notefile. When cards are revised, old information remains in the notefile. Compacting creates a new, compressed version of the notefile by copying into the new version only the latest information from the old version. To understand the structure of notefiles and compacting, see Appendix A, Notefile Concepts.

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

There are two submenu options off **Compact**, **Compact To New File** and **Compact in Place**.

Figure 12-6. The **Compact** submenu.

Compact To New File

Is the same as selecting **Compact** from the main menu. NoteCards prompts you to type in a new file name. The default is the same file name with a higher version number.

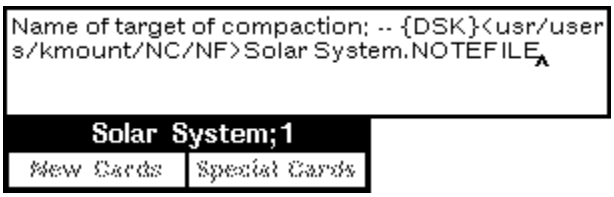


Figure 12-7. NoteCards prompting for the output file name for the compact operation.

Figure 12-8. The notefile Banner after compacting to the same file name. The old version is "Solar System;1." The new version is "Solar System;2."

Compact In Place

Does not build a new copy of your notefile, rather the contents of the old file are rearranged in such a way that old versions of cards are written over. This is useful when your notefile is large and there isn't room to store another version. However, be aware that when you use this option you are not able to back up cards to previous versions by using the **Inspect & Repair** utility. We recommend

compacting to a target file and saving the uncompacted version until you are sure that the older versions of cards are not needed.

Inspect & Repair

Reads the data area of your notefile, looking for good card parts, including outdated and deleted versions. It then allows you to delete or back up card parts to earlier versions through its menu interface. Only when the notefile is considered healthy are you allowed to perform the link rebuilding.

Inspect & Repair has two submenu options, From **Links** and **From Contents**.

Figure 12-9. The **Inspect & Repair** submenu.

From Links

Rebuilds the notefile index using only the links part of the notefile.

From Contents

Rebuilds the notefile index using the links part of the notefile as well as the links stored in the bodies of individual note cards. **From Contents** takes longer to rebuild a notefile index.

For a detailed explanation of the **Inspect & Repair** utility, see Appendix B, The Notefile Inspector.

Copy

Allows you to copy a notefile.

Prompts you for a new name and path for the notefile to be copied and copies it. This is the same as doing a file copy.

Rename

Allows you to rename a notefile.

Prompts you for a new name and path for the notefile to be renamed and renames it. This is the same as executing a file rename.

Delete

Deletes a notefile.

NoteCards asks you to confirm the deletion by typing a carriage return for "No" or a `Y` and a carriage return for "Yes." If your response is "Yes," a second confirmation is required a second time to insure against unintentional deletion.

Create

Creates a new empty notefile. When you use **Create**, it is possible to assign an already existing name to the new notefile, thereby creating more than one version of the notefile.

Create has two submenu options, **Create without Open** and **Create and Open**.

Figure 12-10. The **Create** submenu.

Create without Open

Is the default action. It prompts you for a new notefile name, creates a new notefile, and place the new notefile's Banner on the screen. If you supply only a name and no path, the notefile is created in your connected directory.

Create and Open

Does all of the above, and opens the new notefile.

Checkpoint

Saves the contents of any active cards to the notefile without closing the cards or the notefile. The index array is written out to the file and the checkpoint pointer is reset to the end of the file.

A notefile consists of two parts, an index area and a data area. For each card, the index contains four pointers into the data area. There are pointers for the notecard's contents, title, property list, and list of links. When, for example, you change a card's title, NoteCards writes the new title at the end of the file, and the card's title pointer is changed to point to the new title. To increase access speed, index modifications are written in memory and are not written to the file till checkpoint, or close time. There is a checkpoint pointer that is updated to point to the end of file at 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, due to a power failure or some other mishap, and later you reopen the notefile, NoteCards can notice the extra data beyond the checkpoint pointer. Each time a notefile is opened, the checkpoint pointer is compared with the end of file pointer. If they don't agree, then a message that something is wrong is printed out. At this point, you must choose one of three options from a menu: **Cancel**, **Truncate File,** or **Inspect & Repair**.

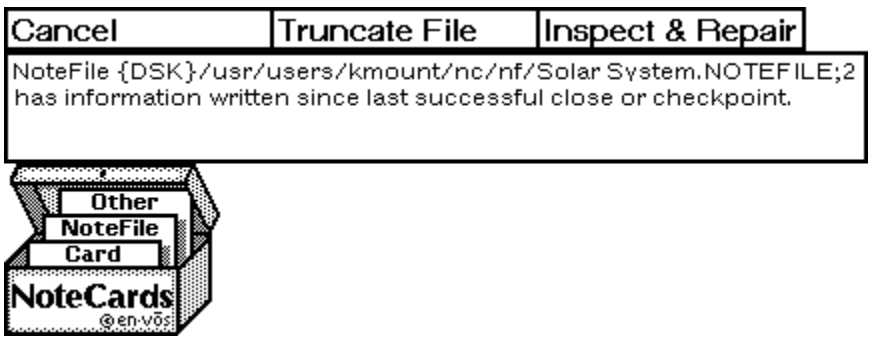


Figure 12-11. The menu which appears when you open a damaged notefile.

Cancel causes the open notefile operation to be aborted.

Truncate File deletes everything after the checkpoint pointer.

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. For more details, see Appendix A, Notefile Concepts and Appendix B, The Notefile Inspector.

Figure 12-12. The **Checkpoint** submenu.

Checkpoint and **Checkpoint Notefile** pop up a menu of known notefiles which you can checkpoint.

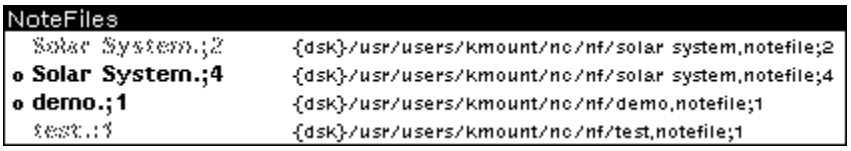


Figure 12-13. A menu of the list on known notefiles.

The grayed over notefiles are closed and cannot be checkpointed. Selecting one of the open notefiles causes that notefile to be checkpointed.

Checkpoint All checkpoints all open notefiles.

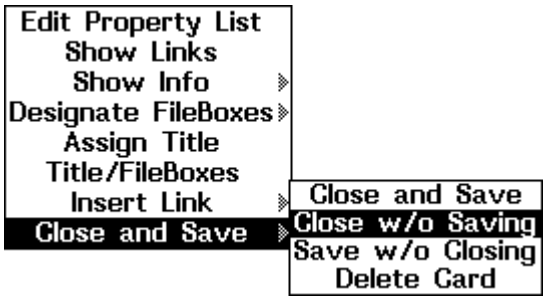


Figure 12-14. The card menu with the Close w/o Saving menu option selected.

In contrast to **Checkpoint**, when you choose **Close and Save** or **Save w/o Closing** from the card menu, the card's contents are written to the notefile but the notefile index is not written to the notefile. If you are anticipating a crash, you should checkpoint you notefile frequently.

Close

Allows you to close a notefile.

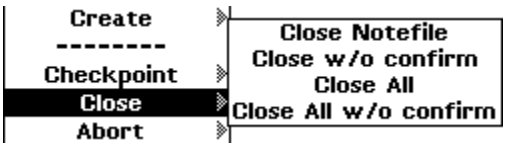


Figure 12-15. The **Close** submenu.

Close and Close Notefile

Pop up a menu of known notefiles like that shown in Figure 12-13. Selecting one of these notefiles closes that notefile. The grayed over notefiles are already closed. Selecting one of these has no effect.

Close w/o confirm

Closes a notefile without asking you about open cards on the screen. Open cards are closed and saved to either their designated FileBox or the "To Be Filed" FileBox.

Close All

Closes all open notefiles.

Close All w/o confirm

Closes all open notefiles without asking you about open cards on the screen. Open cards are closed and saved to either their designated FileBox or the "To Be Filed" FileBox.

Abort

Truncates and closes a notefile.

If you discover that you have made changes to a notefile that you don't want to keep, choose this option to truncate your file. This should only be done if you feel that the post-checkpoint work is not worth saving.

Figure 12-16. The **Abort** submenu.

Abort and Abort Notefile

Pop up a menu of known notefiles like that shown in Figure 12-13. Selecting one of these notefiles aborts that notefile. The grayed over notefiles are already closed. Selecting one of these has no effect.

Abort All

Aborts all open notefiles.

NC FileBrowser

Opens up a FileBrowser specialized for NoteCards.

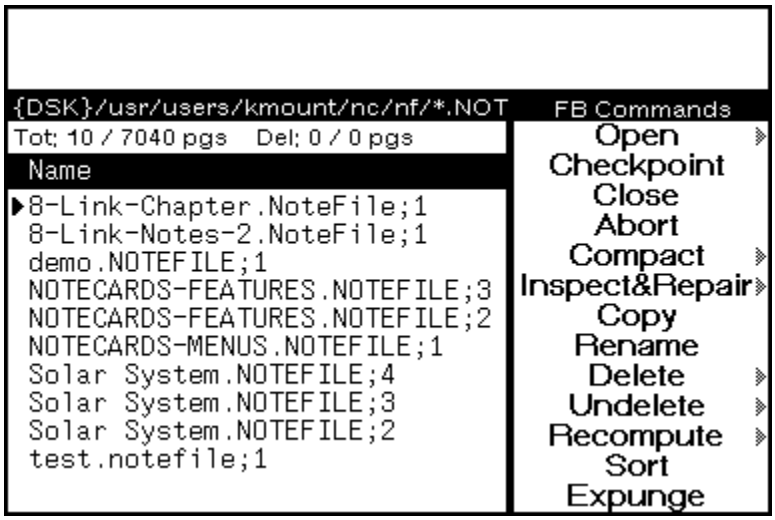


Figure 12-17. The NC FileBrowser.

The commands: **Open**, **Checkpoint**, **Close**, **Abort**, **Compact**, and **Inspect & Repair** can be looked up in this chapter.

The commands: **Copy**, **Rename**, **Delete**, **Undelete**, **Recompute**, **Sort**, and **Expunge** should be looked up in Chapter 14, The FileBrowser.

Card Options

The Card options allow you to perform a **Close**, **Delete**, **Copy**, or **Move**, on a series of cards or a card structure.

Figure 12-18. The "Card Ops" menu.

Close

Enables you to close any number of currently open cards by simply shift-selecting each one's card-ID into the "Selecting Cards" prompt window.

After you have specified your choices, select **Done** from the menu on top of the "Selecting Cards" prompt window, or type a carriage return. **Cancel** aborts this command. **Undo** removes the last selected card from the list.

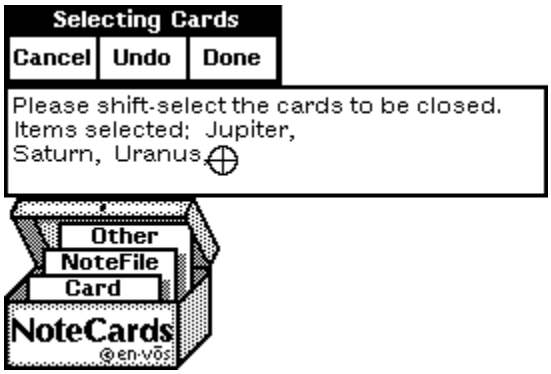


Figure 12-19. The "Selecting Cards" prompt window and menu.

The **Close** option has two submenu options.

Close and Close NoteCards

Perform the same action as **Close**.

Close Structure

Closes a linked list of cards. This operation follows both local and global links.

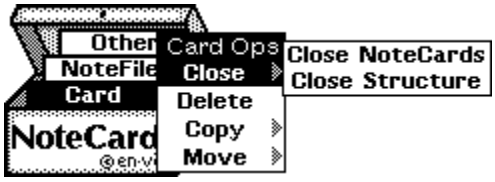


Figure 12-20. The **Close** submenu.

Once you have selected the cards from which to start the **Close Structure** operation, a menu opens asking you what types of links you want to follow. When you are done, select **Done**. **Reset** sets the menu to the values it had when it came up. **Abort**, terminates the operation.

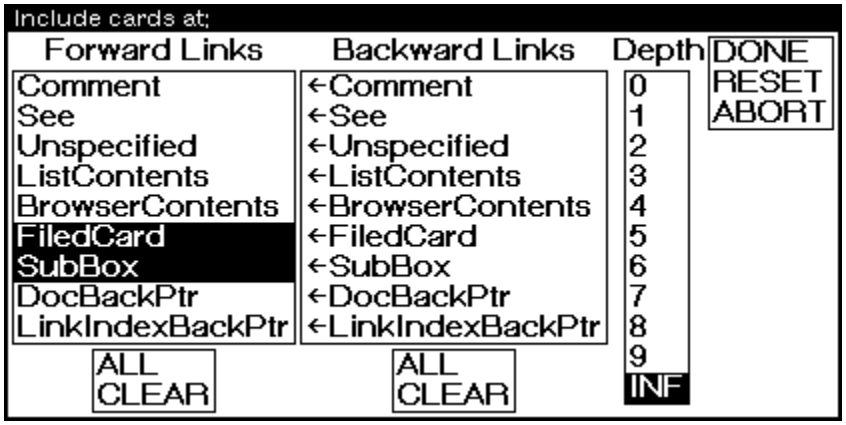


Figure 12-21. The "Include cards at:" menu.

For an explanation of Forward and Backward links see Chapter 8, Links. The **Depth** variable determines how many link levels are traversed in carrying out the operation. If set to a depth of 2, only cards two levels down are closed. For example, if you had cards 1,

2, 3, and 4 on the screen, each forming a sequence (card 1 only pointing to card 2, and card 2 only pointing to card 3, etc.), then if the depth were set to two and you started the close operation from card 1, only cards 1, 2, and 3 would be closed, because only two levels of links would be traversed. Card 4 would remain open on the screen after the **Close Structure** operation.

Delete

Works just like **Close**.

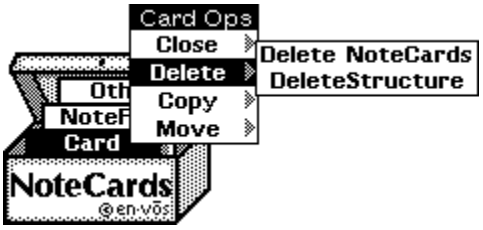


Figure 12-22. The **Delete** submenu.

We caution you to be very careful with the **Delete Structure** command as it is very easy to unintentionally delete large numbers of cards. This command is most useful in conjunction with **Copy Structure** as a way of performing a safe **Move Structure**. First you copy the structure you think you want and then once you have verified that you have gotten the structure you wanted, you use **Delete Structure** to remove it.

Copy

Works just like **Close**. Allows you to copy a group of cards or a card structure to another FileBox which may be in a different notefile.

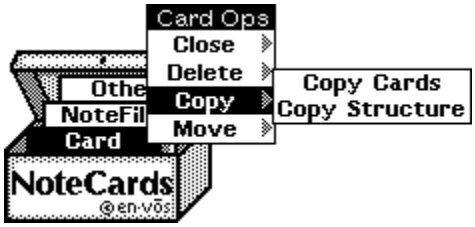


Figure 12-23. The **Copy** submenu.

Move

Works just like **Close**. Allows you to move a group of cards or a card structure to another FileBox which may be in a different notefile.

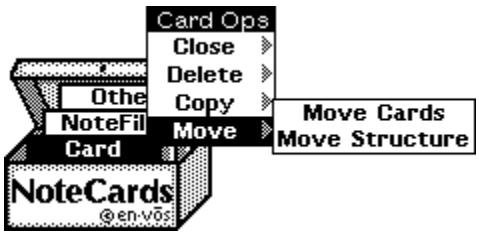


Figure 12-24. The **Move** submenu.

Other Options

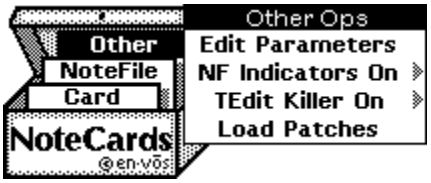


Figure 12-25. The "Other Ops" menu.

Edit Parameters

The **Edit Parameters** option brings up the System Parameters menu. This menu is discussed in detail in Chapter 13, System Parameters.

NF Indicators On

NF Indicators On and the submenu option **Indicators On** turn on notefile indicators for all the cards on the screen.

Indicators Off

Turns off all notefile indicator windows for all the cards on the screen. Note that whether or not cards come up with notefile indicators is determined by the system parameter **Show Notefile On Cards**. To learn more about this parameter and changing system parameters, see Chapter 13, System Parameters.

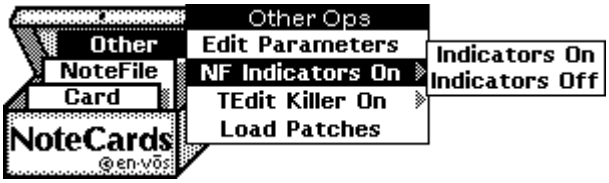


Figure 12-26. The "NF Indicators On" submenu.

Notefile indicators are small windows on the tops of cards which show the name and path of the card's notefile.

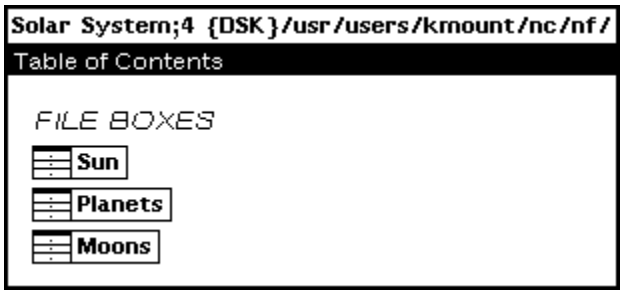


Figure 12-27. A notefile indicator window attached to the "Table of Contents" card from the Solar System notefile.

TEdit Killer On

Kills TEdit processes after they have been around for a specified period of time.

Figure 12-28. The "TEdit Killer On" submenu.

Since every text-based card and every TEdit window, including TEdit command menus, is a separate TEdit process it is very easy to get twenty or thirty TEdit processes all vying for machine time. This can greatly slow down your machine's response time. If you like to work with many cards and windows open on the screen at one time turning the TEdit killer on will help improve your machine's response time. Note that TEdit killer only affects text-based cards. The processes for sketch-based cards are unaffected. To restart a TEdit process that has been killed in this manner, you only have to click in the TEdit window.

TEdit Killer On

Turns on the TEdit killer process.

TEdit Killer Off

Turns off the TEdit killer process.

New Process Limit

Sets the number of TEdit process which are allowed to exist. Selecting this option brings up the following menu.

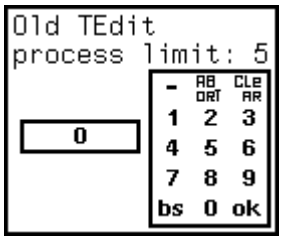


Figure 12-29. The TEdit process limit number pad.

A response of "0" allows you to have one TEdit process running and not none.

The default value is "5."

New Wait Time

Sets the number of second which a TEdit process can be idle before it will be killed. Selecting this option brings up the following menu.

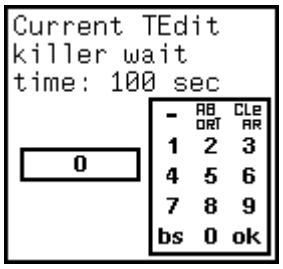


Figure 12-30. The TEdit killer wait time number pad.

Note: If both the TEdit process limit and the TEdit killer wait time are set to "0," it can freeze your system.

The default value is "100."

Load Patches

Forces the loading of any patches which were distributed with the NoteCards sysout.

[This page intentionally left blank]

13. SYSTEM PARAMETERS

System parameters allow you to customize the NoteCards environment to your particular needs and tastes.

This chapter explains:

How to set system parameters.

What each system parameter does.

Accessing System Parameters

System parameters are set from the "NoteCards Parameters" menu. To access this menu, hold the left mouse button down on the **Other** option on the MenuBox Icon. When the "Other Ops" menu comes up, still holding the mouse button down, slide off and select the **Edit Parameters** option. At this point, you release the mouse button. You can also execute a single rapid left-button click on the Other menu option and the "Other Ops" menu will come up and stay up. To select the option you want click on it.

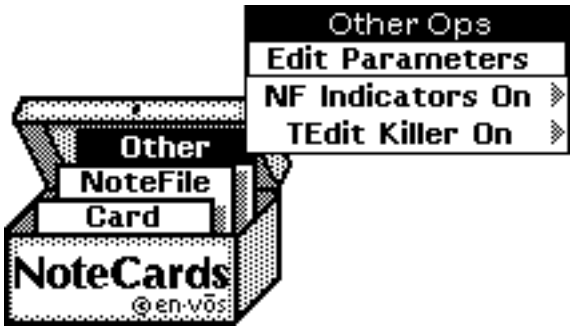


Figure 13-1. The MenuBox Icon with the **Edit Parameters** option on the **Other** menu option selected.

This action brings up the "NoteCards Parameters" permanent menu, shown below in Figure 13-2.

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, between "Yes" and "No" if binary, and allow selection from an appropriate menu otherwise.

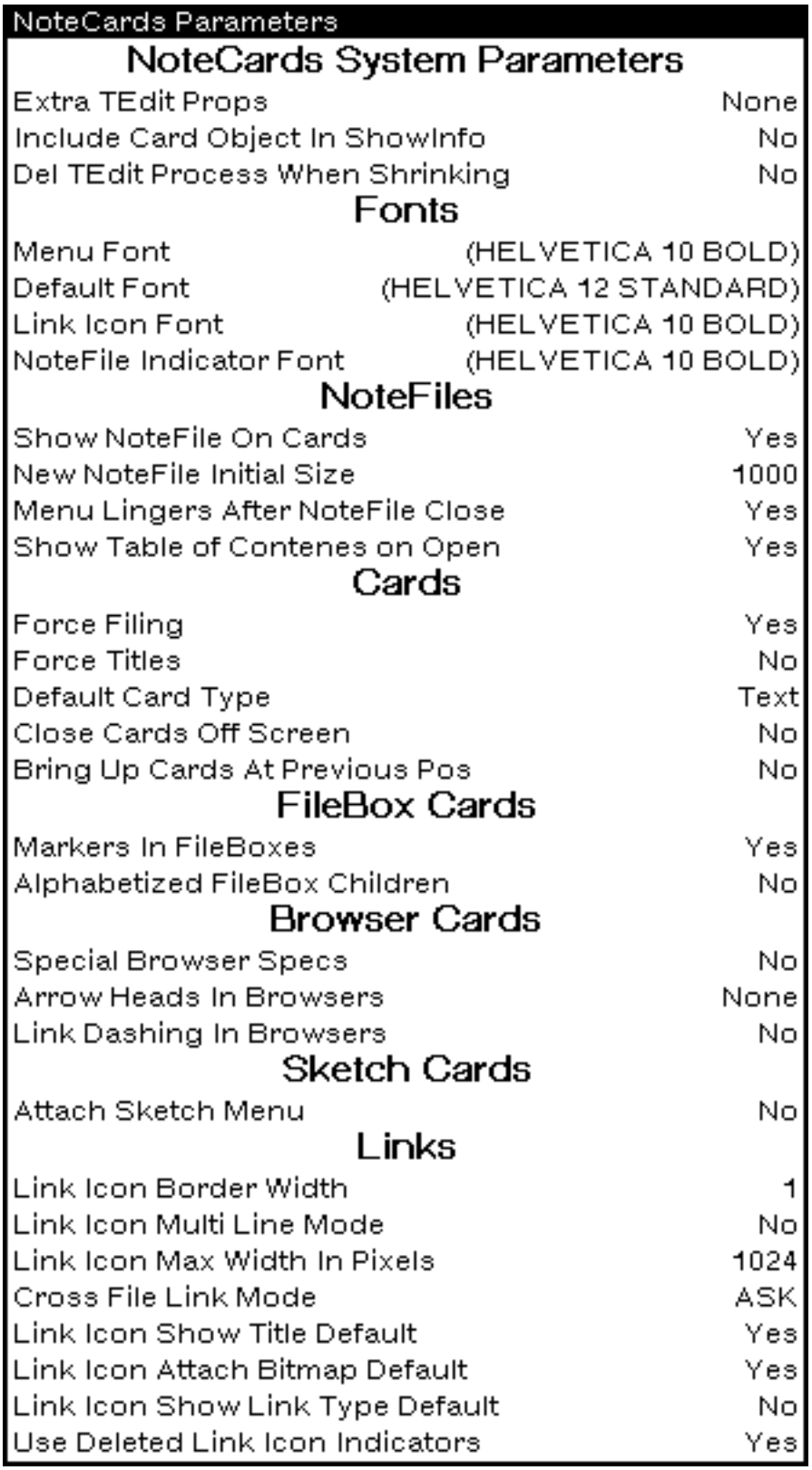


Figure 13-2. The "NoteCards Parameters" menu.

NoteCards System Parameters

Extra TEdit Props

Does not initially appear on the list of parameters for the user version of NoteCards. It is optained by going into the **Low Level Tools** option on the background menu. This parameter is to be used by people extending the NoteCards environment. Users of NoteCards should not add this parameter to their "NoteCards Parameters" menu.

The value of this parameter is a property list that is appended to the property list passed as the PROPS argument to every call to TEdit. This parameter allows you to set up the initial TEdit props for all new text-based cards.

Selecting this property brings up a type-in window above the parameter editor window. To set the parameter value, type in a sequence of zero or more attribute/value pairs followed by carriage return. The possibilities for attributes and values are described in *A User's Guide to TEdit*, Chapter 8, The Programmer's Interface to Tedit, in the section "Using the Top-level TEdit function".

For example, to set the default line leading to 2 you would type:

PARALOOKS (LINELEADING 2)

You should be very careful when setting other than LOOKS-related properties. In particular, NoteCards sets properties like QUITFN and PROMPTWINDOW itself, overriding any alternative values for these properties that you may have typed in.

Special note: If for some reason, you need to have an expression EVAL'ed in the typein window, precede it with ctrl-y.

The default value is "None."

Include Card Object In ShowInfo

When the value of this parameter is Yes, the Show Info inspector will include the Card Object. This option is for people extending or modifying the NoteCards system.

The default value is "No."

Del TEdit Process When Shrinking

If the value of this parameter is "Yes," when you shrink any text-based card, NoteCards will kill the TEdit process behind it. If the value is "No," the TEdit process remains running even when the card is shrunk to an icon.

Expanding an icon for a card whose TEdit process has been killed will not automatically restart the TEdit process. Instead, the user has to click the MIDDLE button in the substance of the card, and select the **New Edit Process** item from the singleton menu that appears.

Running TEdit processes use up limited resources. As a result, only about 25 to 30 TEdit processes can be active simultaneously. Setting the value of this parameter to "Yes," eliminates

unnecessary use of these limited resources when text-based cards are shrunk to icons. The cost is the extra effort needed to restart the process when the card is expanded.

The default value is "No."

Font Parameters

When you select any of the parameters in the Fonts section the system displays the "Please select a font:" menu.

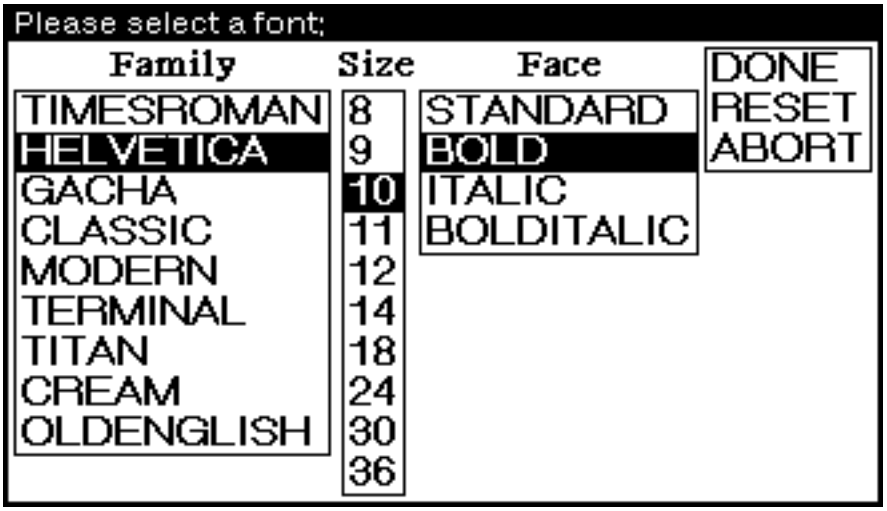


Figure 13-3. The "Please select a font" menu.

To change the link icon font, select the values you want for each of the parameters with the mouse and click on **DONE**. **RESET** returns the values to their original settings. **ABORT** returns the values to their original settings and exits.

The value for each font parameter is three-element list which includes the font family, the font size in points and the font face.

Menu Font

The value of this parameter is a description of the font in which the text in all menus will be printed.

The default value is "(HELVETICA 10 BOLD)."

Default Font

This parameter controls the font all new text-based cards will use.

The default value is "(HELVETICA 12 STANDARD)."

Link Icon Font

The value of this parameter is a description of the font which is used to print the card name and link type that appear inside the links.

The default value is "(HELVETICA 10 BOLD)."

Notefile Parameters

Show Notefile On Cards

In order to associate a card with its notefile, NoteCards provides a notefile indicator window which attaches to the top of the card's title bar.

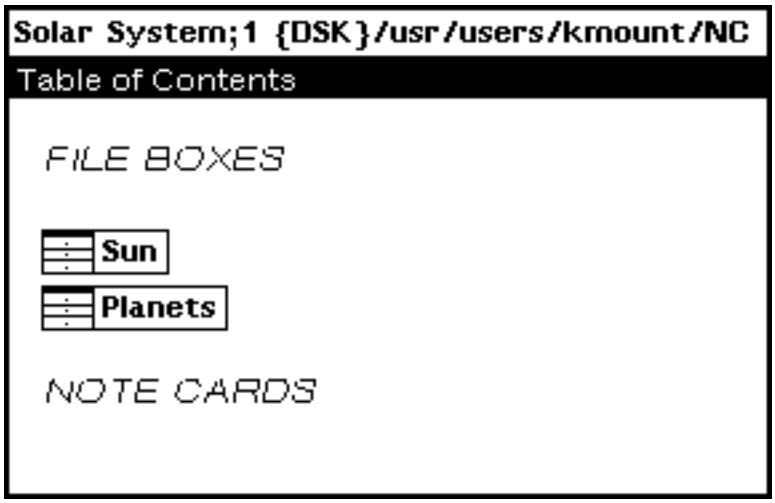


Figure 13-4. A card with its notefile indicator window attached to the top of the card.

The notefile indicator is constructed by concatenating the root file name, the version number, and the full file name. The indicator is left justified. If the card is too narrow to show the entire Indicator string, the full file name is truncated before the root name and version number as shown.

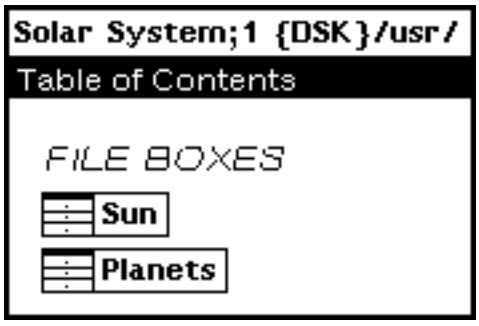


Figure 13-5. A card with its notefile indicator window attached to the top of the card, showing a truncated full file name.

If the NoteCards system parameter **Show Notefile On Cards** has the value "Yes," then a notefile indicator is attached to every card as it is brought up on the screen.

Occasionally, after you have reshaped a card, a piece of the text in the notefile indicator window will be garbled or broken. To redisplay the text in this window, place the point of the mouse cursor in the notefile indicator window, hold down the right mouse button, and select the **Redisplay** option from the window menu which pops up.

The default value is "No."

New Notefile Initial Size

When you create a new notefile, the system consults the value of this parameter to determine the initial size of the notefile index. The notefile-index size is equal to the number of cards the notefile can hold. Larger notefiles take a little longer to create. Notefiles can be enlarged later on, using the **Inspect & Repair** option.

The default value is "1000."

Menu Lingers After Notefile Close

If the value of this parameter is "No," the notefile Banner is removed from the screen when the notefile is closed. If the value is set to "Yes," then the Banner remains on the screen with its menu items greyed.

The default value is "Yes."

Card Parameters

Force Filing

If you set **Force Filing** to "Yes," the system will require you to file every card in 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 the "To Be Filed" FileBox if the card has no parent FileBox when you close it.

The default value is "Yes."

Force Titles

If you set **Force Titles** to "Yes," this parameter will require you to title a card before it can be closed. Setting the value to "No" eliminates 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, however, has a unique identifier for each card and will continue to regard them as unambiguous entities.

The default value is "Yes."

Default Card Type

When you select **the New Cards** option from the notefile Banner with the left mouse button, a card of the default type is immediately created and displayed. This parameter allows you to change this default to suit your needs. Select the new default card type from the list of choices provided.

The default value is "Text."

Close Cards Off Screen

If set to "Yes," then when a card is closed, it is first moved off the screen so that the close happens invisibly.

The default value is "No."

Bring Up Cards At Previous Pos

If set to "Yes," cards are brought up on the screen in the position they occupied when you last closed them. You are not asked to position a frame image.

The default value is "No."

FileBox Card Parameters

Markers In FileBoxes

If set to "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 set to "No," then new fileboxes come up without markers and new children are inserted at the current cursor position . Note that regardless of the **Markers In FileBoxes** setting, if a filebox has no markers (because you've deleted them) then new children are inserted at the cursor position.

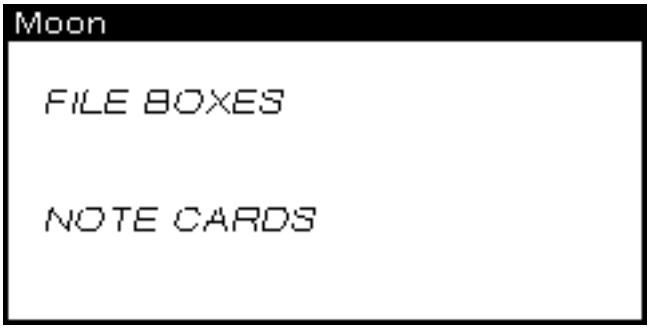


Figure 13-6. A FileBox card showing the markers "FILE BOXES" and "NOTE CARDS."

The default value is "Yes."

Alphabetized FileBox Children

If set to "Yes," then FileBoxes created while this parameter is set to "Yes" will automatically alphabetize any new cards filed in them.

The default value is "No."

Browser Card Parameters

Special Browser Specs

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 the "Browser Cards" section of Chapter 11, System Cards.

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 Special Browser Spec to retain its default value.

The default value is "No."

Arrow Heads In Browsers

This dictates whether arrow heads are drawn on browser links. The variable can be set to either AtMidpoint, AtEndpoint, or None. See the Browser card section of Chapter 11, System Cards for details.

The default value is "None."

Link Dashing In Browser

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.

There may be problems printing Browsers with dashed lines.

The default value is "No."

Sketch Card Parameters

Attach Sketch Menu

If set to "Yes," the Sketch menu is automatically attached to the right-hand side of each Sketch card when it is opened.

The default value is "No."

Link Parameters

Link Icon Border Width

Changes the border size of a link icon.

Figure 13-7. A link icon with a border size of 3.

The link icon border width can also be set to zero for link icons that blend more smoothly with text.

Figure 13-8. A link icon with a border size of 0.

The default value is "1."

Link Icon Multi Line Mode

This parameter works in conjunction with **Link Icon Max Width in Pixels**. Multi line mode determines whether the link icon will allow multiple text lines or whether all the text will be forced to appear on a single line. If **Link Icon Multi Line Mode** is set to "Yes," all link icons will display in multiple lines, with each line fitting withing the width specified by **Link Icon Max Width in Pixels**. Line breaks occur at word boundaries.

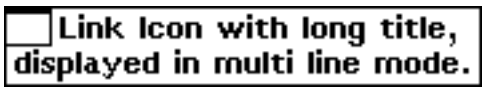


Figure 13-9. A link icon in multi line mode.

The default value is "No."

Cross File Link Mode

All links in NoteCards are bidirectional or two-way with one possible exception. When you are creating cross-file links, these links can be unidirectional or one-way. A one-way link means that a source card contains a to-link pointing to the destination card, but the destination card does not contain a corresponding from-link pointing back to the source card. This allows you to update a source-card file without being forced to open any corresponding destination-card files.

One-Way Creates unidirectional links between cards in different notefiles.

Two-Way Creates the usual bidirectional links between cards in different notefiles.

Ask The system asks you whether you want One-Way or Two-Way links each time you create a cross-file link.

The default value is "Ask."

Link Icon Max Width in Pixels

Sets the maximum width in pixels of a link icon. This value includes the card-type bitmap, if it is displayed. All links are truncated at the width specified by this parameter.

Figure 13-10. A truncated link icon in single line mode.

The default value is "1024" pixels.

Link Icon Show Title Default

This parameter and the following two determine the manner in which link icons are displayed if these parameters are not currently specified in the icon. Which is to say, if the link icon display mode values are set to "System Default."

If the title field in a link icon is set to "System Default" then the **Link Icon Show Title Default** parameter is consulted to determine whether the title will be shown inside the link icon. If it is set to "Yes" then the title will be shown.

The default value is "Yes."

Link Icon Attach Bitmap Default

If the bitmap field in a link icon is set to "System Default" then the **Link Icon Attach Bitmap Default** parameter is consulted to determine whether the card-type bitmap is shown next to the link icon. If it is set to "Yes" then the card-type bitmap is shown.

The default value is "Yes."

Link Icon Show Link Type Default

If the link type field in a link icon is set to "System Default" then the **Link Icon Show Link Type Default** parameter is consulted to determine whether the link type will be shown inside the link icon. If it is set to "Yes" then the link type will be shown.

The default value is "No."

Use Deleted Link Icon Indicators

If the value of this parameter is "Yes," when you delete a card, all link icons which point to that card are replaced with a deleted icon.

Figure 13-11. A deleted icon.

If the value of this parameter is "No," the links to the card are simply removed, leaving no trace behind.

Delete icons are useful when you may be deleting cards where you have link icons pointing to them and these link icons are refered to in the text.

Eg. "This link 

takes you to the next card in the sequence." If the "Next" card were subsequently deleted with this parameter set to "Yes," this would become, "This link Deleted takes you to the next card in the sequence." If this parameter were set to "No," the sentence would become the following more obscure version, "This link takes you to the next card in the sequence." with no visible link.

The default value is "Yes."

[This page intentionally left blank]

14. THE FILEBROWSER

The FileBrowser provides a convenient user interface for manipulating files stored on a workstation or file server. It enables you to see, edit, delete, print, load, copy, move, rename, compile, sort, and get several types of information about groups of files.

User Interface

Starting FileBrowser

To open a browser on a set of files select the FileBrowser command from the background menu.

FileBrowser will prompt you to create a window by presenting you with a dashed rectangle with the mouse cursor and a small geometric design at the lower right corner.

1. Move your mouse until the upper left corner of the rectangle is where you want it on the screen.
2. Hold down the left mouse button and move your mouse down and to the right, thus expanding the window diagonally, until the window is the right size.
3. Release the mouse button. This creates a window group on your screen in the outlined area.

Next, FileBrowser prompts you for a file pattern. Type a pattern, as described in the section "Specifying What Files to Browse," below.

FileBrowser enumerates the set of files matching the pattern you requested to see. While the enumeration is in progress, the **Recompute** command is grayed out. When the enumeration is finished, you may select files and issue commands. You can scroll the window at any time, even while the browser is busy.

If FileBrowser can't find any files matching the pattern you specified, or you decide you specified the wrong pattern and want to try again, you can specify a new file name pattern from within the browser using the **New Pattern** command; see **Recompute** in the section "FileBrowser Commands," below.

You can have as many active FileBrowsers open at once as you like.

Specifying What Files to Browse

A full file name in NoteCards consists of a device or host (such as your local disk, a file server), a principal directory and zero or more subdirectories, a file name (possibly including an extension), and a version number. These fields are put together in the form

```
{HOST}<DIRECTORY>SUBDIRECTORY>FILENAME.EXTENSION;VERSION
```

A file name pattern, as specified to FileBrowser, consists of a file name with one or more pieces omitted or filled with wild cards (*).

All the files matching the pattern are listed by FileBrowser. Thus, you can browse all the files in a particular directory, all the files in a subdirectory of that directory, all the files in a directory with a particular extension, and so forth. The wild card * can be used to stand for zero or more consecutive characters in the file name. You can use as many wild cards in a pattern as you wish.

If you leave out some of the fields in a file name pattern, the missing fields are defaulted by the system. Omitted fields in the front of the pattern, i.e., the host, device, or directory fields, are filled in by consulting your connected directory. Other omitted fields are filled in with wild cards unless they are explicitly omitted; i.e., the field is empty, but the preceding punctuation is still present. In more detail, some of the cases are as follows:

If you leave out the name of the host/device, specifying *<DIRECTORY>FILENAME*, FileBrowser will use the name of the host/device for the directory to which you are currently connected.

If you leave out both the device and directory names, specifying *FILENAME*, FileBrowser will use the device and directory to which you are currently connected.

If you do not specify a file name, FileBrowser lists all the files in the specified directory (or the connected directory if you also omitted the host and directory).

If you leave out the extension of a file name, FileBrowser lists all the files with the specified file name and any extension. If you omit the extension but include the period that usually precedes the extension, FileBrowser lists only the files with the specified name and *no* extension.

If you omit the version number of the file name, FileBrowser lists all versions of the matching files. If you omit the version number but include the semicolon that usually precedes the version, FileBrowser lists only the highest version of the matching files.

Thus, the minimal pattern you can type is * (asterisk—enumerate all files in the connected directory) or ; (semicolon—enumerate just the highest version of all files). If you press the RETURN key without giving a pattern, FileBrowser aborts the prompt for a pattern, leaving you with an empty browser in which the only things you can do are change some FileBrowser parameters (see the subcommands of **Recompute** in the section "FileBrowser Commands," below) and then use the **Recompute** command to be prompted for a pattern again.

Examples

The pattern * specifies all files in the connected directory. It is equivalent to *.* or *.*;*.

The pattern <NoteCards>Demo specifies all files in directory NoteCards with name Demo and any extension. It is equivalent to <NoteCards>Demo.*;*.

The pattern <NoteCards>Demo. specifies all files in directory NoteCards with name Demo and *no* extension. It is equivalent to <NoteCards>Demo.;*.

The pattern *.TEdit specifies all files in the connected directory with the extension TEdit. It is equivalent to *.TEdit;*.

The pattern *.TEdit; specifies only the newest version of all files in the connected directory with the extension .TEdit.

The pattern <NoteCards>A*E specifies all files in directory NoteCards whose names begin with A and end with E and have any extension.

The pattern {DSK}<NoteCards>*MY* specifies all files in directory {DSK}<NoteCards> whose names contain the substring MY and any extension.

Using the FileBrowser Window

The FileBrowser window has six major subwindows, which from top to bottom are as follows.

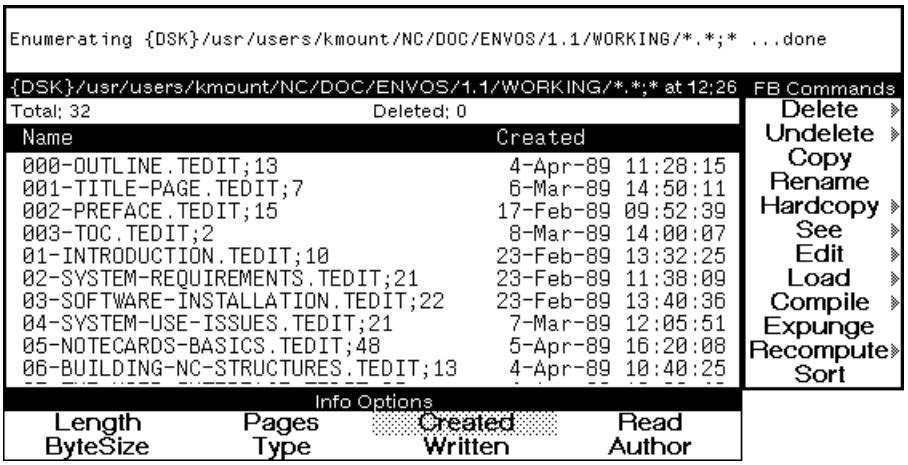


Figure 14-1. The FileBrowser display.

Prompt window This topmost subwindow is where FileBrowser prints messages about what it is doing and receives input from you. Its contents are cleared before every command.

Tally window This subwindow immediately below the prompt window keeps a running tally of the total number of files listed in the window and the number of files that you have marked for deletion. In addition, if one of the attributes you are displaying is a size attribute (Pages or Length, as in the INFO menu, described below), this window maintains a tally of the total number of pages in the files listed and the files marked for deletion.

This window also has a title bar across the top identifying the pattern you specified and the time at which the directory enumeration was performed.

The window is blank while the files are being enumerated.

Browser window This is the principal subwindow, in which the files matching the specified pattern are listed. Each file's name appears at the left, and various attributes of the file are displayed in columns to the right. A title bar across the top of the browser window identifies the contents of each column (e.g., Name, Pages, Created). The files are listed in alphabetical order, with multiple versions of the same

file listed in decreasing version order; i.e., the newest version appears first. The width of the column listing the file names is initially chosen to be appropriate for average-sized file names. If the files you asked to browse have particularly long names, then when FileBrowser has finished listing all the files it may choose to redraw the browser window with the attribute columns moved farther to the right to accommodate the longer file names.

Command menu This menu appears vertically along the right side of FileBrowser window (under a title bar "FB Commands") and lists the commands that you may select to perform operations on the files in the browser, or to change the appearance of the browser. Most of the commands operate on the set of currently selected files (see the section "Selecting Files" below). Some commands have subcommands, as indicated by the small triangle alongside them, which can be selected by holding down the left mouse button and sliding the mouse to the right over the triangle.

Info menu An additional subwindow, the Info menu, is not normally displayed. It is used to change the set of file information (attributes) displayed in the browser (see the section "Getting Information About Files," below).

Scroll bar If there are more files in the listing than fit at one time in the browser window, you can scroll the browser window to view more files. Slide the mouse cursor out the left side of the browser window to get the scroll bar and press the left mouse button to scroll the region up and the right mouse button to scroll it down. Pressing a mouse button when the cursor is near the bottom of the scroll bar will scroll the region by larger increments than when the cursor is at the top.

You can also press the middle mouse button in the scroll bar to move the listing to the place that corresponds to that position in the scroll bar. For example, pressing the middle mouse button when the cursor is at the bottom of the scroll bar will display the end of the listing. This quick-scrolling technique is called thumbing. The gray box in the scroll region indicates where the currently displayed contents are, relative to the entire contents of the browser.

Similarly, if there is more attribute information than fits in the browser window, you can scroll the browser window horizontally to view the rest of the attribute information. To do this, slide the mouse out the bottom of the browser window to get the horizontal scroll bar. The left button scrolls to the left, the right button to the right.

Selecting Files

Most FileBrowser operations are performed by selecting a single file or set of files, then giving a command that specifies what you want to do with the selected files. The current selection is indicated by a small triangle in the left margin of the browser next to each selected file.

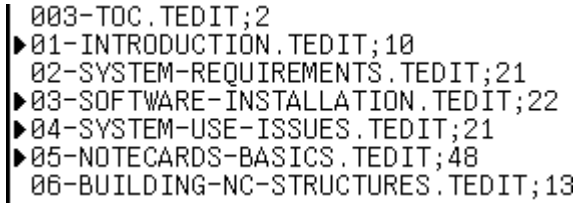


Figure 14-2. Four selected files marked by triangles.

To select one file, point to any part of the line (which lists the file name and its attributes) and press the left mouse button. If other files are already selected, this unselects them; thus, a file selected with the left mouse button is always the only selection.

To add a single file to the current selection, press the middle mouse button at any place in the line. The file is selected without unselecting any other file.

To remove a single file from the current selection, hold down the control key and press the middle mouse button at any place in the line. The file is unselected without affecting any other file.

To extend the selection to include a group of contiguous files, that is, to select all the files between a file and the nearest already selected file, press the right mouse button on any part of the line. You can only extend the selection from the first selected file upward, or the last selected file downward. In addition, files marked for deletion are not normally selected when you extend.

If you want to include all files, both deleted and undeleted, hold down the control key while extending the selection.

Some lines in a FileBrowser display are directory-only lines. These lines are slightly indented and name the directory and subdirectory to which the files listed below that line belong. You cannot select in these lines, though you can copy-select them (see the section "Copy-Selecting Files," below).

Commands that Require Input

Some FileBrowser commands require input from you. For example, the **Copy** command requires that you supply a destination file name. When a command requires input, FileBrowser prints a prompt message in its prompt window. This is usually followed by a default answer. If you want the default answer, you can just press the carriage return to finish the input. If you want to specify a different answer, simply start typing it; the default answer is erased and your answer replaces it.

Alternatively, you can modify the default answer by backspacing over individual letters, or typing control-W to back up over complete words. Typing CONTROL-Q erases the entire answer. You can also use the mouse to edit your answer, using the same rules as followed by the Executive (see the documentation of TTYIN). Briefly, the left mouse button positions the caret at a character boundary; the middle mouse button positions the caret at the nearest word boundary; and the right mouse button deletes the characters between the caret and the mouse.

When you have finished, position the caret at the end of your answer, if it isn't there already, and press the carriage return. You can also type CONTROL-X to finish your answer even if the caret isn't at the end.

If you change your mind and want to abort the command, supply an empty input; i.e., if there is an answer in progress, backspace over it or type CONTROL-Q to erase it, then press <RETURN>. FileBrowser prints "aborted" and aborts the command. In most situations, the CONTROL-E interrupt can also be used to abort your answer.

While you are typing an answer, you can copy-select file names out of the browser (or any other browser), as described below in the section "Copy-Selecting Files". This can be useful, for example, if you wish to rename a file to a similar name in the same directory, or move a file into a subdirectory listed in the browser.

Aborting Commands

During commands of indefinite duration, such as **Recompute** or **Copy**, FileBrowser adds another command to the browser, **Abort**.

Figure 14-3. The FileBrowser menu with the Abort option attached.

Clicking on the **Abort** command will immediately abort the current operation. Aborting some commands can take a little while, as FileBrowser may need to do some cleaning up, so the **Abort** command is greyed out during this time to show you that it is doing something.

Quitting the FileBrowser

To quit a FileBrowser, simply close the browser window. If any files have been deleted but not expunged, a small menu will pop up listing two options: **Expunge deleted files** and **Don't Expunge**. If you choose **Expunge deleted files**, the files will be expunged before the window closes. If you choose **Don't Expunge**, your deletions are ignored. If you click outside the menu, no action is taken, and the **Close** command is aborted.

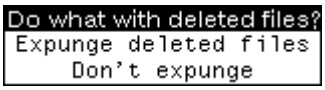


Figure 14-4. The "Do what with deleted files?" menu.

If you have finished with FileBrowser only temporarily and want to put it aside to work on later, you can shrink the browser by selecting the **Shrink** command from the right-button background menu. The browser shrinks to an icon which displays the file pattern inside the browser. If any files are marked for deletion, you will be prompted with the same menu of expunge options as when you close a browser.

Figure 14-5. A FileBrowser shrink icon.

Copy-Selecting Files

You can copy-select file names from a FileBrowser into other windows, such as TEdit windows, by holding down the COPY or SHIFT key while selecting a name in the window. The full name of the file is inserted as if you had typed it where the input caret is flashing. You can also copy-select in a directory-only line, in which case the full directory name is inserted in your type-in.

Getting Hardcopy Directory Listings

You can get a hardcopy listing of the directory displayed in a FileBrowser by using the regular window **Hardcopy** command. Press the right button in FileBrowser's prompt window or tally window and select **Hardcopy** from the menu. FileBrowser will produce a hardcopy listing of the files and the attributes displayed in the browser.

If the browser displays a large number of attributes, or your default printer font is too large, the listing may not accommodate all the attributes on one line, making the listing less readable. You may want to make the listing with fewer attributes, or use a smaller font for the listing.

FileBrowser Commands

Delete, Undelete

Removing a file from the file system using FileBrowser is a two-step process. First, you mark the file or files for deletion. Then you issue the **Expunge** command. Any time between the deletion and the expunge you can change your mind and undelete any of the files.

To mark a file or files for deletion, select them, then choose the **Delete** command. FileBrowser draws a line through the deleted files. It also adjusts the numbers in the tally window to show how many files are marked deleted and how many pages they contain. It is thus easy to see how much file space you will regain when you issue the **Expunge** command.

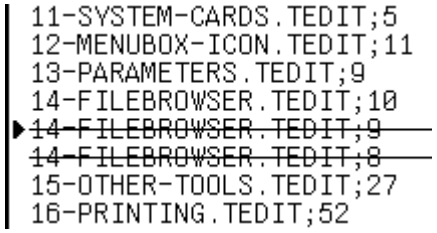


Figure 14.6 Two files marked for deletion.

To undelete a file or files (i.e., to remove the deletion mark), select them, then choose the **Undelete** command. The lines through the files are removed, and the tally of deleted files is updated. The **Undelete** command has a single subcommand, **Undelete ALL Files**, which undeletes all the files in the browser, independently of whether they are selected. This is useful if you completely change your mind about deleting any files.

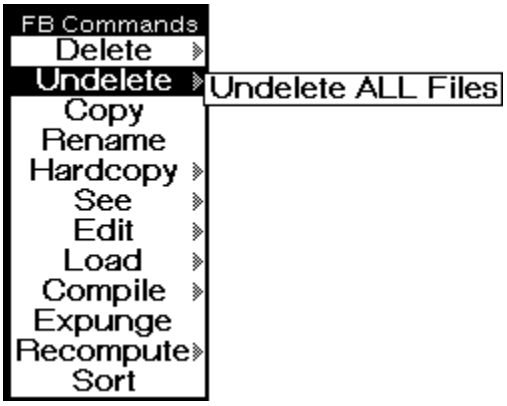


Figure 14-7. The **Undelete** submenu.

The **Delete** command has a useful subcommand, **Delete Old Versions**. When you have been editing a file in the text editor and performing repeated **Put** commands, multiple versions of the file accumulate, each more recent version denoted by a higher version number. The **Delete Old Versions** command is used to delete excess versions of the files displayed in the browser.

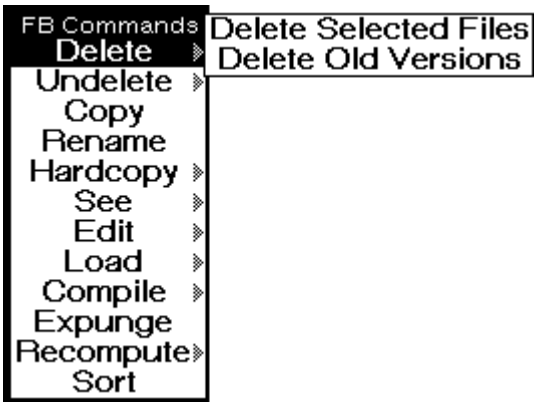


Figure 14-8. The **Delete** submenu.

To use this command, press the mouse down on the **Delete** command and slide the cursor out to the right, choosing the **Delete Old Versions** command. Unlike the **Delete** command (or **Delete Selected Files**, the equivalent subcommand), the **Delete Old Versions** command operates on all the files in the browser. FileBrowser prompts you for the number of versions of each file that you wish to retain. It offers the default of one version. You can accept the default or you can type a different number of your choosing, followed by a carriage return. FileBrowser then marks for deletion all but the most recent n versions of all the files in the browser, where n is the number you specified. Before issuing the **Expunge** command, you can, if you wish, scroll through the browser, undeleting any particular files for which you wish to retain more versions than you specified.

The **Delete Old Versions** command is sometimes useful even when you are not planning to actually expunge the files. This is because of the way extending the selection avoids deleted files (see the section "Selecting Files," above).

For example, if you wanted to copy only the most recent version of all the files in the browser to another location, you could do the following:

1. Use the **Delete Old Versions** command, retaining just one version. This marks deleted all files but the newest version of each.

2. Go to the start of the browser and select the first file, then scroll to the end of the browser and press the right mouse button to extend the selection to the end of the browser. You have selected exactly the newest version of each file.

3. Use the **Copy** command to copy those files.

4. Finally, use the **Undelete ALL Files** command to undelete all the old versions.

Copy

The **Copy** command is used to copy an entire file or set of files to another file system location; for example, from your disk to a file

server. Select the file(s) you wish to copy, then select the **Copy** command. FileBrowser prompts you to supply a destination.

If you selected just one file, FileBrowser prints the old name and offers a default, which consists of the same file name and either the same directory that was last used in a **Copy** or **Rename** in this FileBrowser, or the connected directory if this is the first use of **Copy** in this FileBrowser. You can accept the default or supply your own destination file name. If you supply just a directory specification, e.g., *{SERVER}<DIRECTORY>*, the file is copied to that directory under its current name. If you supply a complete name, the file is copied to that exact name.

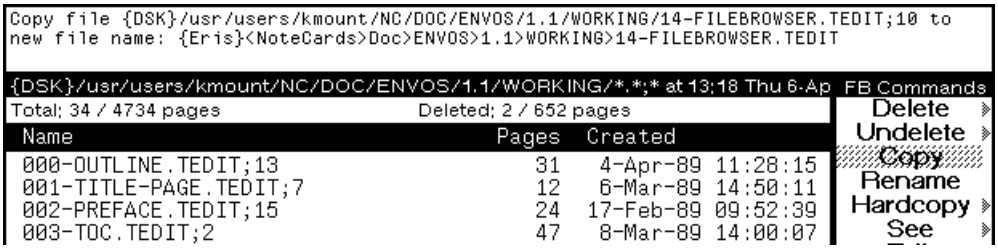


Figure 14-9. The Copy command comfirming the name of the file to copy to.

Note: Unless you specify a version number in the destination file name, the version number of the new file will be 1, or one higher than the highest existing version of the file in the destination directory, independent of the version number of the old name.

Even files marked for deletion can be copied.

If you selected several files, FileBrowser notes how many files you wish to copy and offers as a default destination the connected directory. You can accept the default or supply a different directory. All the files are copied to that directory under the names they currently have.

You must supply a directory specification, e.g., *{SERVER}<YOURDIRECTORY>*, rather than a complete file name, since you can't copy multiple files to the same name. If you mistakenly type a file name, rather than a directory specification, FileBrowser will complain and abort the command.

If you want to copy files from different subdirectories, FileBrowser will ask, via a message in its prompt window, if you want to preserve the subdirectory structure at the destination. If you answer **Yes**, then the names at the destination will include not just the root name of each source file, but also all the subdirectory names below the greatest subdirectory prefix common to all the selected files (this common prefix is displayed as part of the question). If you answer **No**, then the names at the destination are formed solely from the root name of each file (the name displayed in the browser), ignoring any directory information each name might have. This can cause multiple files with the same root name to be copied into the same destination name (but with different version numbers, of course).

When copying (or renaming) multiple versions of the same file, FileBrowser does the copying in order of increasing version number, so that the versions at the destination are in the same relative order as at the source.

As each file is copied, FileBrowser prints a message giving the full name of the new file. If a file with the chosen name already exists, the new file's version number will be one higher; otherwise it will be version 1 (one). The new file will have the same creation date as the original file. If the destination file happens to be one that matches the pattern of the files in the browser, the new file is inserted in the appropriate place in the browser display. However, if it matches the pattern of some other FileBrowser, it is not inserted in that other browser's display (in other words, FileBrowsers do not know about each other). You would have to **Recompute** the destination FileBrowser to see that the file was copied into it.

Rename

The **Rename** command is used for changing the name of a file or group of files, or for moving a file or group of files to a different directory.

The **Rename** command is used in exactly the same way as the **Copy** command. If you rename a single file, you can supply a complete new name or just a directory; if you rename several files, you must specify a directory. As each file is renamed, FileBrowser prints a message giving the file's new name and removes the file from the browser display. If the new name belongs in the same browser, it is inserted in the appropriate place. If for some reason a file could not be renamed, this is noted in the FileBrowser prompt window. The reasons for the failure of a renaming operation are roughly the same as for the failure of an **Expunge**; the file is open, or you do not have the access rights needed to rename the file.

Note: If the destination of the rename is on a different file system than the original file, changing its name is equivalent to copying the file to its new name and then deleting the original file.

Hardcopy

You can print text files, TEdit, Sketch, Interpress, and PostScript files from FileBrowser. Select the appropriate file or files, then select the **Hardcopy** command. The **Hardcopy** command will determine what type of file you are printing and call the appropriate function for printing that file. Then the files will be printed one at a time on your default printer. The prompt window will display status messages telling you when files are being printed and when they are done (if your printer is one that provides this status service).

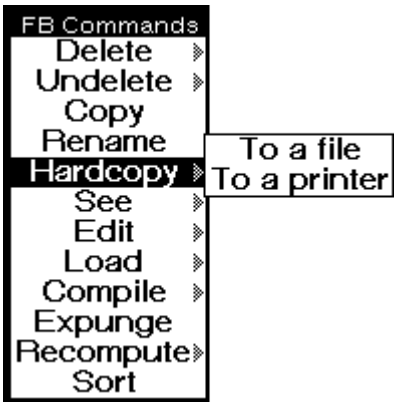


Figure 14-10. The **Hardcopy** submenu.

You may specify printing to a file or to a printer other than the default printer by means of a submenu from the **Hardcopy** command. This menu is the same as the one on the **Hardcopy** command in the background menu. Selecting **To a printer** presents you with a choice of printers from a menu. Selecting **To a file** prompts you to supply a file name. If you selected a single file, you must specify a single hardcopy file name (or accept the offered default). If you selected multiple files, then you must specify a pattern with a single asterisk somewhere in the "name" field, for example, *.INTERPRESS or Hardcopy-*.IP. The output file names are constructed by merging the pattern with each selected file name. If the name includes an extension that implies the type of print format (e.g., .IP or .INTERPRESS implies the Interpress print format), then a file of the specified type is made automatically. Otherwise, you are prompted to supply a print format type.

Note: For files stored on servers not supporting random access, FileBrowser is currently unable to determine that a file is in TEdit format unless the file has the extension .TEDIT. Therefore you should use TEdit to hardcopy TEdit files with other extensions. Use FileBrowser's **Edit** command (to call TEdit), then the **Hardcopy** command either from the TEdit Expanded Menu or from the right-button menu.

Note: To obtain a hardcopy of the directory itself, use the **Hardcopy** command from the right-button window menu. See the section "Getting Hardcopy Directory Listings".

See

When you browse a directory you sometimes want to see a file before printing or performing some other operation on it. To do this, select the file, then select the **See** command from the command menu. FileBrowser will prompt you to open a window by presenting you with a dashed rectangle and printing a message in the system prompt window. The window will be blank until FileBrowser starts printing the contents of the file in it.

There are actually four different **See** commands, as shown in the submenu for **See**. The **Fast SEE Pretty** and **Fast SEE Unformatted** commands are provided to let you quickly see the contents of a file, but not do anything fancy, such as scroll around

at random in the file. The slower **Scrollable & Pretty** command does let you scroll, and if the file contains formatting information of a kind that FileBrowser knows about (via the editors you have loaded), you will see the file formatted. However, this command does much more work, and may take a bit longer to show you even the first line of the file. The **Filebrowse** command is for use on files that are directories; it is described in the next section.

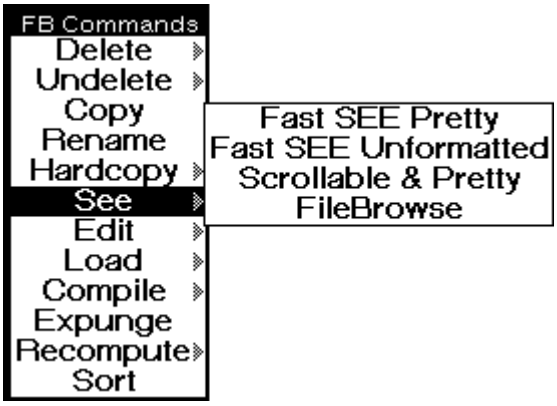


Figure 14-11. The **See** submenu.

The **Fast SEE Pretty** and **Fast SEE Unformatted** commands display the selected file in the display window one windowfull at a time. When the file fills the window, a small menu appears at the bottom-left corner of the window (or top-left if your display window is at the bottom of the screen) giving you the option of seeing more of the file or aborting the **See** command. If you issued the command with more than one file selected, you also have the choice of aborting just the display of this file or the entire **See** command.

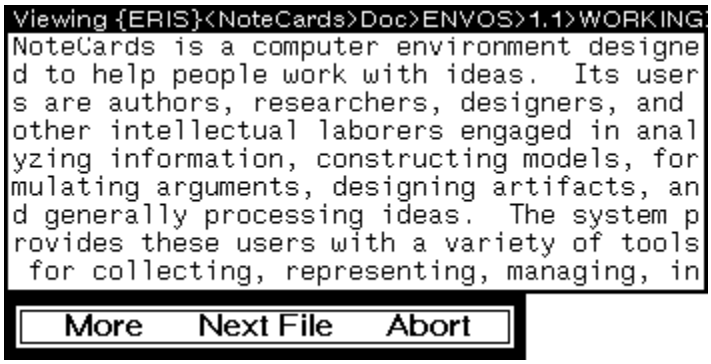


Figure 14-12. The **See** window control menu.

If you select **More**, the **See** command displays another windowful of the file. If you select **Next File**, the **See** command closes this file and goes on to display the next file in the current selection. If you select **Abort**, the entire **See** command is aborted. You can also abort the **See** command by closing the display window.

The next time you give a **Fast See Pretty** or **Fast SEE Unformatted** command, the same window will be reused.

The only difference between the **Fast See Pretty** and the **Fast SEE Unformatted** commands is the manner in which the characters of the file are processed as they are displayed.

The pretty (formatted) version interprets certain control characters found in source files to be font change commands, and interprets certain multibyte sequences as representing characters in the Xerox extended character set. It also squeezes out blank lines and shrinks the indentation of indented lines in order to better fit the text in a window that is generally much narrower than the standard file width. The formatted version is thus most appropriate for viewing source files and files containing plain text.

The unformatted version of the **See** command does no special processing on the characters whatsoever. It simply displays each eight-bit byte as a single character, uninterpreted. This means that bytes that do not represent normal printing characters may be displayed as black boxes, in the form ^x or #x, or as a flashing of the window (for the byte that represents the ASCII "bell'' character). The Unformatted version is thus most appropriate for viewing binary files that also contain text portions that might be worth seeing; e.g., compiled files (those with extension .LCOM) or Interpress masters (extension .IP or .INTERPRESS).

The **Scrollable & Pretty** command views a file in a different way. This command brings up a new read-only TEdit window for viewing a file (only if TEdit is loaded in your system; otherwise, **Scrollable & Pretty** reverts to fast format). You can scroll and copy-select the file's contents at will, as with any TEdit window. If the file is a NoteCards source file, its contents are first formatted into a TEdit document, so that all the font information is retained. This formatting, however, can take a long time for a large file. For other kinds of files, the **Scrollable & Pretty** command is exactly like viewing the file in a regular TEdit window, except that you can't edit it. If you want to edit a file, use the Edit command instead of the See command.

You can keep the display window used by the **Scrollable & Pretty** command open as long as you like. The command uses a different window for each file you select. Simply close the window with the standard right-button window menu when you are finished with it.

Filebrowse

The **Filebrowse** command is a subcommand of **See** used to view a subdirectory in its own FileBrowser window. The selected file must be a (sub)directory. Subdirectory files appear in browsers on XNS file servers when the depth is finite (see the **Set Depth** command), and their names always end in ">". On Sun servers subdirectory file names end in "/."

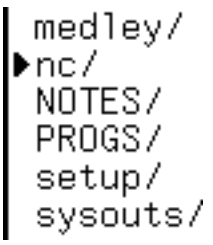


Figure 14-13. Some subdirectories on a Sun server, indicated by trailing slashes.

The **Filebrowse** command prompts you for a region for a new FileBrowser window group, in which it proceeds to enumerate the contents of the selected subdirectory, to the same depth as the main browser used, if any.

Edit

The **Edit** command invokes an editor on the selected file. To specify an editor explicitly, use one of the commands on the submenu.

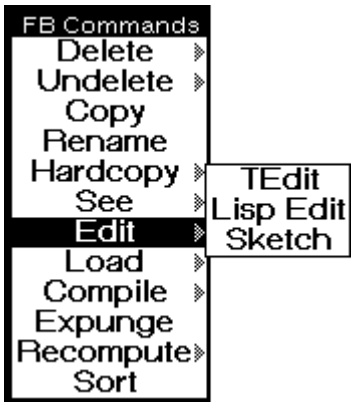


Figure 14-14. The **Edit** submenu.

To start up a TEdit editor on a selected text file, select **Edit** with the left mouse button. If you have recently closed a TEdit window, then TEdit will probably reuse that window; otherwise, you will be prompted to create an editor window. TEdit only remembers the most recently abandoned window, however, so if you issue the **Edit** command when you have several files selected, you will be prompted to create windows for all but the first file.

The subcommand **Lisp Edit** is for people extending the NoteCards environment. Do not use this command unless you are extending the NoteCards system.

If you select the main **Edit** command, without sliding off to the submenu, then FileBrowser's default editor, TEdit, is called.

Load

The **Load** command is for people extending the NoteCards system. Do not use this command unless you are extending the NoteCards system.

Compile

The **Compile** command is for people extending the NoteCards system. Do not use this command unless you are extending the NoteCards system.

Expunge

If you are sure you want to delete files permanently, choose the **Expunge** command. The **Expunge** command is grayed while FileBrowser expunges the files that were marked for deletion by the

Delete command. As each file is removed from the system, it is removed as well from the browser display, and the tally of total number of files and number of deleted files is updated, so you can see the progress of the command.

If for some reason a file can not be expunged, FileBrowser prints a message saying so in its prompt window, but continues to expunge the other files. The main reasons that prevent a file from being expunged are its being opened, either by you or some other user, or your not having the access rights required to delete it (if it is on a file server). See the section "Troubleshooting Problems with FileBrowser," below.

Note: The **Expunge** command is not affected by the current selection; it operates only on files marked for deletion, whether currently selected or not.

Recompute

FileBrowser's display shows those files that existed and matched the specified pattern at the time you created the browser. If you want the browser to reflect the latest state of the file system, use the **Recompute** command.

For example, if you open a FileBrowser on your directory, then save several versions of a TEdit file on that directory, the file listing will not display the new versions until you **Recompute**.

The **Recompute** command operates exactly as when you started up FileBrowser initially: it clears the display and tally windows, then enumerates the files matching the pattern. The **Recompute** command in the menu is grayed until the enumeration is finished. During this time you cannot scroll or perform any other operations on the browser. However, you can close the window if you want to abort the command and throw away the browser.

If any files are marked for deletion at the time you request a **Recompute**, FileBrowser will present the choice of expunging or undeleting the files, just as it does when you want to quit the browser (see the section "Quitting the FileBrowser," above).

The **Recompute** command also has a menu of subcommands that allow you to list different files or different information for the same set of files.

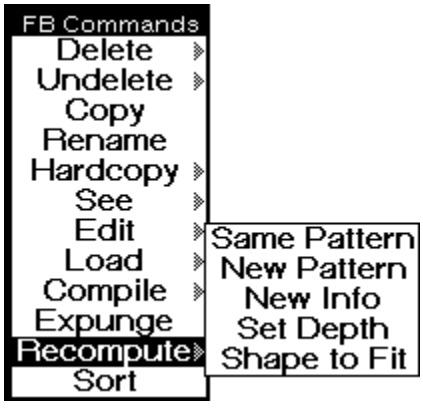


Figure 14-15. The **Recompute** submenu.

Same Pattern Is the same as the main **Recompute** command, i.e., it enumerates the files matching the same pattern as before.

New Pattern Lets you change the pattern, i.e., browse a new set of files. FileBrowser prompts you to supply a new file name pattern and offers the old pattern as an initial default. You can either type an entirely new pattern, replacing the one offered, or delete the old pattern one character at a time by backspacing. Press the carriage return when you have finished specifying the pattern. FileBrowser then enumerates the files matching this pattern, just as with the **Recompute** command. You can abort the command with the Abort button, or by erasing the whole pattern (by backspacing or using CONTROL-Q) and then pressing the <RETURN>.

New Info Lets you change which attributes the browser displays. It is described in the next section.

Set Depth Lets you change the depth to which FileBrowser enumerates a directory on an XNS file server. It is described in the section "Set Depth."

Shape to Fit Reshapes the FileBrowser window so that all the attributes in the display are visible at once, eliminating the need to horizontally scroll the window to get at all the information.

New Info

FileBrowser displays some file attributes, or information about the file, alongside each file in the browser display. Ordinarily, the attributes displayed are the size of the file in pages, its creation date, and its author. You can change the attributes displayed in a particular browser window by using the **New Info** command.

To use the **New Info** command, select it from the submenu of the **Recompute** command. FileBrowser opens up an additional subwindow, the "Info Options" window, below the display window. This subwindow contains a menu of attributes, with the current defaults shaded. Selecting a shaded item unshades it; selecting an unshaded item shades it. When you have selected all the attributes you wish to see displayed, issue either the **Recompute** or the **New Pattern** command. The files will be listed with the new information you requested. The "Info Options" window stays open—you can close it at any time.

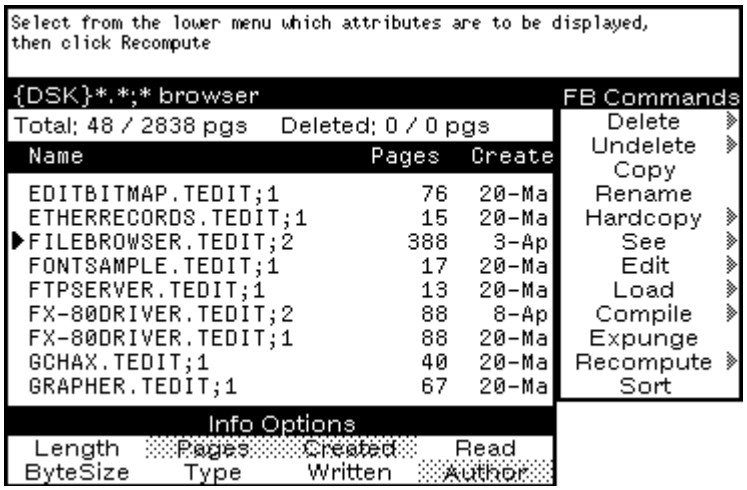


Figure 14-16. A FileBrowser showing the "Info Options" window.

The Info Options items have the following meanings:

Created The date and time that the content of the file was created. This date changes whenever the file is modified, but does not change when a file is copied or renamed.

Written The date and time the file was last written to the file system. This date is never older than the Created date, but it can be newer if the file is copied, unmodified, from one file system to another.

Read The date and time the file was last read. This attribute may be blank if the file has never been read.

Author The login name of the person who wrote the file, or last modified it.

Length The length of the file in (usually eight-bit) bytes.

Pages The number of 512-byte pages in the file. On some servers, this attribute is blank if the file is empty.

ByteSize The size (in bits) of the bytes in the file.

Type A value indicating what kind of data the file contains. The usual values of this attribute are TEXT, meaning the file contains just characters, or BINARY, meaning the file contains arbitrary data. Some servers have additional types, such as INTERPRESS for files in Interpress format.

Set Depth

XNS file servers support a feature that allows enumerating a directory to a user-specifiable depth. The "depth" of a file reflects the number of subdirectories between it and the root of the enumeration, i.e., the directory or subdirectory you gave in the pattern to FileBrowser, not counting any containing wildcards (asterisks). The immediate descendants of the root are at depth 1, files in subdirectories of depth 1 are at depth 2, and so on.

Ordinarily, FileBrowser enumerates a directory to the default depth, which is usually unlimited. To enumerate a directory to a different default, use the FB command with argument :DEPTH *n*, for some positive integer *n*, or T for unlimited depth. To change the depth in an existing FileBrowser, use the **Set Depth** command, a subcommand under the **Recompute** command. The command offers you a menu of choices:

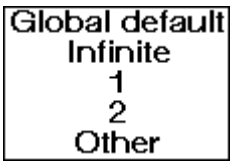


Figure 14-17. The **Set Depth** menu.

Global default means use the default depth, overriding the depth at which this browser was last enumerated. **Infinite** means use no depth limit (same as depth T). **1** and **2** are common depth choices;

to choose some other numeric value, select **Other** and enter the value via the displayed keypad.

The **Set Depth** command does not affect the current display. It takes effect the next time you use the **Recompute** or **FileBrowse** commands from the same browser.

During a **Recompute**, if a subdirectory appears at the specified maximum depth, its descendants are not enumerated; rather, the subdirectory itself appears as an entry in the browser display. This entry can be selected, just like a file, but only a small number of commands can be used on it: you can **Rename** it, you can **Delete** it if it has no descendants, and you can **FileBrowse** it. It has attributes, just as ordinary files do. Its page size is the size of the entire subtree rooted at the subdirectory.

Note: Depth currently affects only XNS servers; all other devices ignore it and enumerate to their own default depths. In addition, due to a bug in XNS Services 10, depth is ignored for nontrivial patterns, i.e., anything but "*.*".

Sort

The **Sort** command allows you to sort the files in the browser by any attribute of the files displayed in the browser. Selecting **Sort** brings up a menu of attributes by which to sort. This menu includes all the attributes currently displayed in the browser (such as Creation Date, Author), plus the choice Name. For some attributes you can sort forward or backwards; the choice is on a submenu, and the default is generally in the order of numerically greatest (e.g., size) or most recent (e.g., creation date) first.

If the attribute you select is not Name, then the file names displayed in the browser will be reformatted to include their directory portion (if there are any subdirectories below the browser's main pattern), as the subdirectory information is no longer implicit in a file's position in the browser.

The sort order Name, Decreasing Version is the default order in which browsers initially are created.

Troubleshooting Problems with FileBrowser

When FileBrowser returns the message: `No files in group` *`FILENAMEPATTERN`* when you know those files exist, the file server is probably down or rejecting connections. If this is so, your only option is to wait until the server is functioning again, and then give the **Recompute** command. In the case of an Xerox file server, the enumeration of files can also fail if you do not have sufficient access privileges; this condition is usually noted by a message in the system prompt window.

When you try to expunge a file and FileBrowser displays the message: `Can't expunge` *`FILENAME`*, it may be because you

don't have write access to the file, or someone else is reading the file. However, the most common reason is that the file is still open. Be sure to close any TEdit windows in which you may still be viewing the file. If you have recently issued a **Hardcopy** command for the file, a background process may still be working on the file.

[This page intentionally left blank]

Digital Clock

Introduction

The digital clock allows you to keep track of the time in multiple time zones. The clock updates itself once a minute. Clicking the left mouse button inside any of the clock windows will cause the digital clock to update itself.

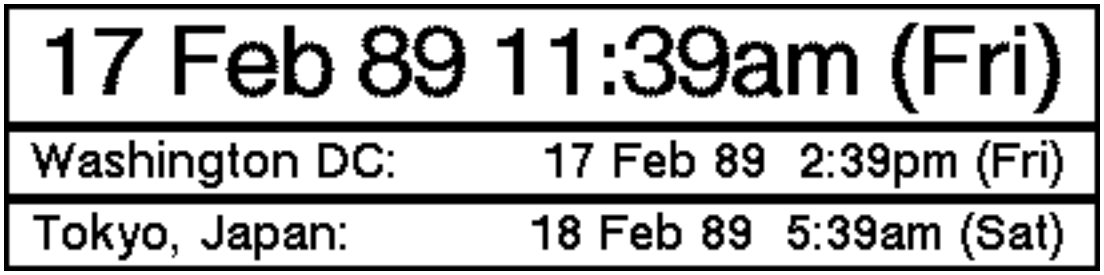


Figure 15-1. The Digital Clock with two auxilliary regional time-zone windows showing the time in Washington and Tokyo.

Starting the Digital Clock

To put a digital clock on the screen, bring up the background menu by holding down the right mouse button, position the mouse over the **Clocks** option and slide off to the right to bring up the Clocks submenu. Slide the mouse cursor over the **Digital Clock** choice and release the right mouse button.

Figure 15-2. The **Digital Clock** option from the **Clocks** submenu from the background menu.

Selecting the digital clock option will delete any existing digital clock and start a new one.

When the clock comes up, it will prompt you for an auxilliary time zone. If you click outside the "Enter Time Zone" menu, the clock will come up with just the local time zone.

Figure 15-3. The "Enter Time Zone" menu.

Stopping the Digital Clock

To remove the clock from your screen and kill the clock process, simply close the clock window using the clock window's right mouse-button **Close** option.

Changing the Digital Clock

You can specalize the digital clock to suit your needs. You modify the digital clock by holding the middle mouse button down in the clock window. Holding this mouse button down in the main clock window brings up the menu shown in Figure 15-4. Holding the middle mouse button down in one of the regional time-zone windows brings up the menu shown in Figure 15-5.

Figure 15-4. The main-window middle-button menu.

Figure 15-5. The auxilliary-window middle-button menu.

Set Font

Brings up a series of menus which allows you to change the font in the main clock window.

Figure 15-6. The font menu.

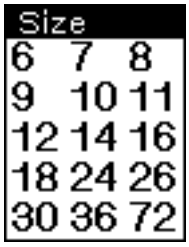


Figure 15-7. The font size menu.

Figure 15-8. The font face menu.

Select each option you want in sequence. If you want to keep any particular aspect of a font, simply click outside that particular menu. Clicking outside all the menus, leaves the font exactly the way it was to begin with.

While the system is searching for a font in the font files, it displays the message, "Fetching Font."

Not all possible combinations of every font exist. When the clock cannot find a particular font, it displays the message "Font Not Found."

Set Time

Brings up a menu which allows you to set the time.

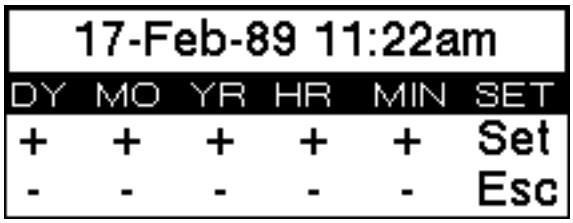


Figure 15-9. The set-time menu.

Each click on a **+** advances the counter for that category; each click on a **-** decreases the counter for that category. Selecting the **Set** option sets the time and closes the menu. Selecting **Esc** (Escape) closes the menu without setting the time.

Set Alarm

Brings up the set-time menu shown in Figure 15-9. The menu functions the same way it does for the **Set Time** option. After you have set the time, the clock will ask you for a message to associate with the alarm time. Type in your message followed by a carriage return.

Figure 15-10. The Digital Clock with the auxilliary window prompting for a message to associate with the alarm time.

Quiet Alarm/Loud Alarm

Toggles the clock back and forth between, a loud/auditory alarm and a quiet/visual alarm.

Figure 15-11. The main window middle-button menu with the **Loud Alarm** option showing instead of the **Quite Alarm** option.

The loud alarm causes the monitor to beep once a minute when the alarm is ringing. The quiet alarm causes the screen to flash once a minute when the alarm is ringing.

Delete Alarm Setting

Turns the alarm off when it is ringing. This option, shown in Figure 15-11, only appears when the alarm is set.

Shape to Fit

Resizes the window to the minimum size necessary for the font you are using.

The window should resize automatically whenever you change the font. If ever the window and font size get out of synch, you can force the window to resize itself by selecting this option.

12-Hour Clock/24-Hour Clock

Toggles the digital clock back and forth between 12- and 24-hour modes. Compare Figures 15-4 and 15-11 to see how these options appear in the menu.

Set Local Time Zone

Allows you to change the time zone shown in the main digital clock window. Selecting this option brings up the "Enter Time Zone" menu, shown in Figure 15-3, from which you can choose the appropriate time zone. Clicking outside this menu leaves the time zone unchanged.

Add New Regional Time Zone

Allows you to add an additional regional time zone to the digital clock display. Selecting this option opens a new auxilliary window on the bottom of the clock and then pops up the "Enter Time Zone"

menu, shown in Figure 15-3, for you to select a time zone for the auxilliary window.

Delete This Window

Deletes the auxilliary window the mouse cursor was in when you pressed the middle mouse button to bring up the auxilliary-window meu.

Figure 15-12. The auxilliary-window menu.

Set Font for Aux Clocks

Changes the font for all the auxilliary clocks indepent of the main clock. Selecting the option brings up the series of menus shown in Figures 15-6, 15-7, and 15-8. Clicking outside of any menu leaves that aspect of the font the way it was originally.

Set Aux Clock Font In Just This Window

Changes the font for the auxilliary window the mouse cursor was in when you pressed the middle mouse button to bring up the auxilliary-window meu. Selecting the option brings up the series of menus shown in Figures 15-6, 15-7, and 15-8. Clicking outside of any menu leaves that aspect of the font the way it was originally.

Figure 15-13. The submenu for setting the font in just one auxilliary window.

Set Time-Zone Heading

Prompts you for a new heading for the auxilliary window you were in when you pressed the middle mouse button to bring up the auxilliary-window meu. Type in the name followed by a carriage return.

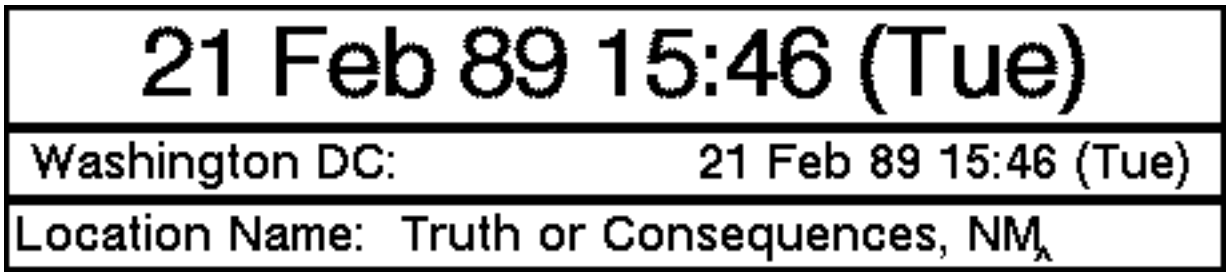


Figure 15-14. An auxilliary window prompting for a new heading.

Occasionally, the window isn't wide enough to display the whole name. In this case choose the option **Shape to Fit** from the main window middle-button menu.

Set Regional Time Zone

Brings up the "Enter Time Zone" menu shown in Figure 15-3, which allows you to change the time zone region. **Set Regional Time Zone** does not allow other than hour increments.

Analog Clock

Introduction

The **Analog Clock** suboption off the **Clock** option on the background menu sets up an analog clock on your screen. The clock is updated once a minute.

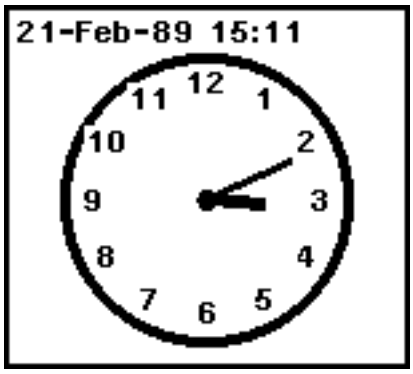


Figure 15-15 The analog clock.

Starting the Analog Clock

To put an analog clock on the screen, bring up the background menu by holding down the right mouse button, position the mouse over the **Clocks** option and slide off to the right to bring up the Clock submenu. Slode the mouse cursor over the **Analog Clock** choice and release the right mouse button.

Figure 15-16. The Analog Clock option from the Clock submenu from the background menu.

The first time you bring the analog clock up in a sysout, it will prompt you to sweep out a window. Once you have done this this clock will always appear in its last size. If you want to resize the analog clock, choose the **Shape** option from the right button window menu, and sweep out the new shape for the clock.

Stopping the Analog Clock

To remove the clock from your screen and kill the clock process, simply close the clock window using the clock window's right mouse-button **Close** option.

Changing the Analog Clock

Holding the middle mouse button down inside the analog clock pops up the following menu.

Figure 15-17. The analog clock menu.

There are four independent properties which the user may control: the hands of the clock, time digits printed where the hands end, rings on the clock face, and 12 numbers around the outside of the clock face.

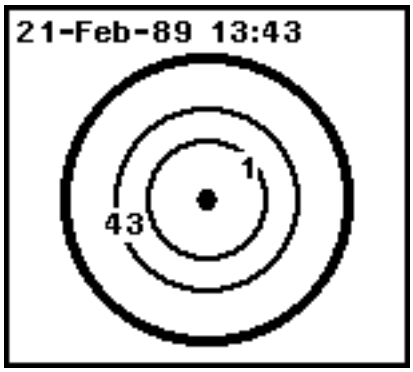


Figure 15-18. An analog clock with no points or numbers, rings, no hands, and times.

NUMBERS

Removes points and adds numbers to the clock face.

POINTS

Removes numbers and adds points to the clock face.

NO.NUMBERS

Removes both numbers and points from the clock face.

RINGS

Adds rings to the clock face.

NO.RINGS

Removes rings from the clock face.

HANDS

Adds hands to the clock face.

NO.HANDS

Removes hands from the clock face.

TIMES

Adds time digits to the clock face.

NO.TIMES

Removes time digits from the clock face.

SHOW.STYLE

Prints the current style to the system prompt window.

SET.TO.DEFAULT

Resets the clock face to the default settings.

CHANGE.DEFAULT

Sets the default to the current style.

SETTIME

Forces the clock to go to the machine to ask it the time.

Known Problems

The clock will occasionally not respond to style changes. To force it to change, close the clock window and restart the clock from the background menu.

Directory Connector

The directory connector allows you to keep track of your currently connected directory and allows you to change the directory. It updates itself about once every ten seconds.

Figure 15-19. The directory connector window.

Starting the Directory Connector

To put a directory connector on the screen, bring up the background menu by holding down the right mouse button, position the mouse over the **Directory Connector** option and release the right mouse button.

Stopping the Directory Connector

To remove the directory connector from your screen and kill its process, simply close the directory connector window using the window's right mouse-button **Close** option.

Changing the Directory Connector Fonts

To change the directory connector font, bring up the background menu by holding down the right mouse button, position the mouse over the **Directory Connector** option, slide off the menu to the right, position the mouse over the **Change Font** option, and release the right mouse button.

Figure 15-20. The **Directory Connector** option on the background menu showing the **Change Font** submenu option.

When you select the Change Font option you will be presented with a series of two menus asking you for a font and a size.

Figure 15-21. The font menu.

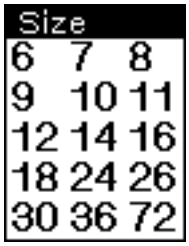


Figure 15-22. The font size menu.

While the directory connector is looking for a font, it prints the message"Fetching Font..." to the system prompt window.

Not all possible combinations of fonts are possible. If a font is not found the directory connector prints the message "Font Not Found." in the system prompt window.

Using the Directory Connector

The directory connector can be used in two ways.

Left Mouse Button

Clicking the left mouse button in the directory connector window causes the window to update itself.

You can shift-select the connected directory name out of the directory connector window with the left mouse button. You do this by pointing to the position in the window where you want the name of the directory to appear and clicking the left mouse button to position the caret cursor there, a FileBrowser prompt window for instance. Next you hold a SHIFT key down and click the left mouse button in the directory connector window. The name of the directory will appear where you positioned the caret cursor.

Middle Mouse Button

Clicking the middle mouse button in the directory connector window brings up a menu of directories to connect to.

Figure 15-23. The "New Directory?" menu.

If the directory you want to connect to is on this list, click the left mouse button on it to connect to it.

If you change your mind and want to abort the operation, click outside the "New Directory?" menu.

If the directory you want to connect to is not on this list, click the left mouse button on * * Connect to Other Directory * * and type the directory name into the prompt window which will appear.

Figure 15-24. The "New Directory?" prompt window.

The prompt window will come up with your currently connected directory. The moment you start typing the system will erase the window and show what you are typing.

If you want to modify the name of your currently connected directory, backspace over as much of the name as you need to and type the name of the new subdirectory. Be sure to end it with a slash.

If you type in an invalid directory name, directory connector will beep and print the message "Not a valid directory name." to the system prompt window.

[This page intentionally left blank]

16. PRINTING

NoteCards includes facilities for printing in the PostScript language by Adobe, and the Interpress, language by Xerox.

This chapter explains:

How to setup your printers.

How to specify where your font directories are.

How to deal with some common printer problems.

How to customize your postscript printer.

Setting Up Printers

You can set up printers from either the background menu or the initialization file. For how to set up your printers with the initialization file, see Appendix C.

Adding Default Printers

To set up a new default printer from the background menu, press and hold the right mouse button in the background, place the mouse cursor on **Set Default Printer** then slide off to the right and select **Add Default Printer**.

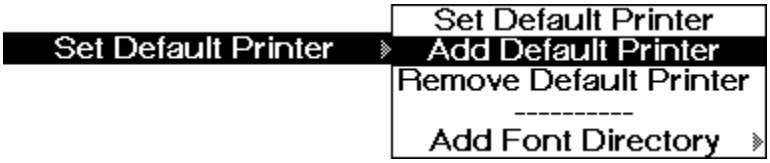


Figure 16-1. The **Set Default Printer** background menu option displaying its submenu.

A prompt window will appear showing you the name of the current default printer and asking for the name of a new default printer.

Figure 16-2. A PostScript printer name being entered as a new default printer.

Figure 16-3. An Interpress printer name being entered as a new default printer.

Enter the name of the new default printer. The moment you type the first character of the new printer name, the old name will be deleted and replaced with what you type. If you don't want to make any changes, strike the carriage return key. If you want to modify the name that is there, you can backspace over the name to erase

part of it. If you type a carriage return without a entering printer name, this operation is aborted.

When you type a carriage return to end the name, a menu will pop up requesting the printer type. Select one of the options with the left mouse button.

Figure 16-4. The system prompting you for the printer type.

If you do not select any option (by clicking outside the menu) and the printer's type was previously defined, it will remain the same. If the printer's type was not previously defined, and you do not select any option, the printer's type will be left undefined which will cause problems when you try to print. If you leave the printer type undefined the system will warn you.

Figure 16-5. The system prompt window giving the new printer status and warning you that the printer type for Oahu is undefined.

Figure 16-6. The system prompt window giving the new printer status.

At this point, if you have already specified the font directories, you should be able to print to any default printer.

Setting the Default Printer

The system maintains a list of printers, one of which is the default printer--the printer that files are sent to when no printer is explicitly specified.

To set the default printer, select the **Set Default Printer** option from the background menu.

Figure 16-7. The **Set Default Printer** option on the background menu.

This will cause the system to open a menu listing all the known printers.

Figure 16-8. The "Choose Default Printer." menu

Selecting one of these options will make that printer the default printer. The current default printer is the topmost option.

Once you have selected a new default printer, the system will confirm your choice by printing the printer name and type to the system prompt window.

Figure 16-9. The system prompt window giving the new printer status.

If the printer type is undefined the system beeps and issues a warning.

Figure 16-10. The system prompt window giving the new printer status and warning you that the printer type for Oahu is undefined.

Removing a Default Printer

If you make a mistake specifying a printer or want to remove a printer for some reason, you can use the **Remove Default Printer** submenu option on the background menu.

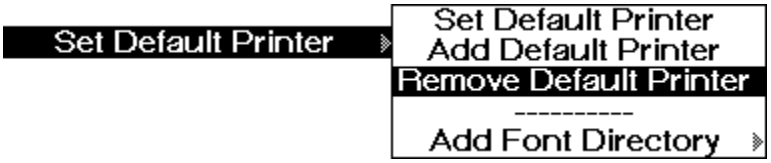


Figure 16-11. The **Remove Default Printer** submenu option off the background menu.

Selecting this option brings up a menu of known printers. Selecting one of these deletes it from the known-printers list and makes the topmost remaining printer the default printer.

Figure 16-12. The "Choose Printer to Remove." menu.

Setting Font Directories

As with printers, it is possible to set your font directories from either the background menu or your initialization file. We recommend that you set your directories from the initialization file. There are two reasons for this. Setting the font directories involves a lot of typing, and these directories are set correctly for you by the standard initialization file unless you have moved the font files. See Appendix C for how to set the font directories from the initialization file.

Adding a New Font Directory

To add a font directory from the background menu, press and hold the right mouse button in the background to bring up the background menu. Position the mouse cursor on the **Set Default Printer** option and slide off to bring up the submenu containing **Add Font Directory**.

Figure 16-13. The **Add Font Directory** submenu options on the background menu.

Selecting **Add Font Directory** brings up a menu asking you to specify the font group you want to add a search directory to.

Figure 16-14. The "Add to which directory group?" menu.

The NoteCards system supports one display and three hard copy modes. For each of these there is a specialized set of fonts. The fonts for the screen display are stored in the display font directories, the fonts for PostScript printers are stored in the PostScript font directories, those for Interpress printers are stored in the Interpress font directories, etc. When you add a new directory to one of these groups, you are providing the system with another place to search for fonts. For example, the display font directories typically include the following directories:

{DSK}/usr/local/lde/fonts/display/presentation/
{DSK}/usr/local/lde/fonts/display/presentation/
{DSK}/usr/local/lde/fonts/display/publishing/
{DSK}/usr/local/lde/fonts/display/printwheel/
{DSK}/usr/local/lde/fonts/display/miscellaneous/

When the system is looking for a new display font, it will search through each of these directories in sequence for the font. The fonts are broken up into separate directories for convenience and efficiency reasons.

Once you have specified the font directory group you want to work with, the system will prompt you for the name of a new directory to search for fonts of that type.

Figure 16-15. The system prompting for a new font directory to search for Interpress fonts.

Enter the name of the new font directory. The moment you type the first character of the font directory name, the old name will be deleted and replaced with what you type. If you don't want to make any changes, strike the carriage return key. If you want to modify the name that is there, you can backspace over the name to erase part of it. If you type a carriage return without a printer name, this operation is aborted.

Note that for Sun machines, the directory name will always start with {DSK}, that subdirectories in the path name are separated by slashes, and that the directory path name must end in a slash (/).

Showing the Font Directories

In order to view a particular set of font directories, use the **Show Font Directories** submenu option shown in Figure 16-13.

Like **Add Font Directory**, this first brings up menu asking you for the directory group you want to show.

Figure 16-16. The "Show which directory group?" menu.

Once you have selected the directory group, the system pops up a window which displays the font directories in that group. You can close this window using the standard window-menu close option or the window will close of its own accord after about two minutes.

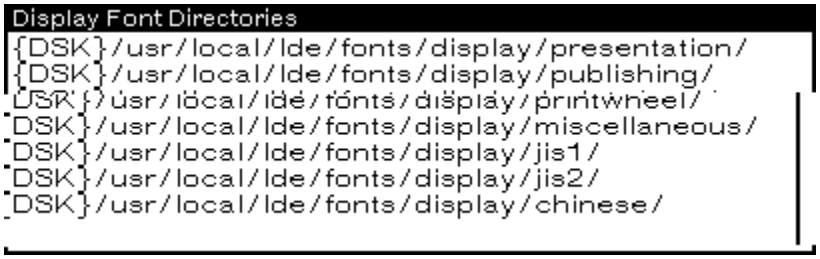


Figure 16-17. A window showing the list of display font directories.

Removing a Font Directory

If you make a mistake entering a font directory name or for some other reason want to remove a font directory, use the **Remove Font Directory** option shown in Figure 16-13.

Like the **Add Font Directory** this first brings up menu asking you for the directory group you want to modify.

Figure 16-18. The "Remove from which directory group?" menu.

Once you have selected the directory group, the system pops up a menu of all the directories currently in that group.

Figure 16-19. The "Remove which directory?" menu.

Selecting one of these options deletes that directory. Clicking outside the menu aborts this operation.

Trouble Shooting Common Printer Problems

This section discusses some of the more common breaks which occur when your printers are not properly set up.

Can’t determine IMAGETYPE for {LPT}

This break occurs when you do not have any printers specified on the default printer list, or if you have not specified the printer type.

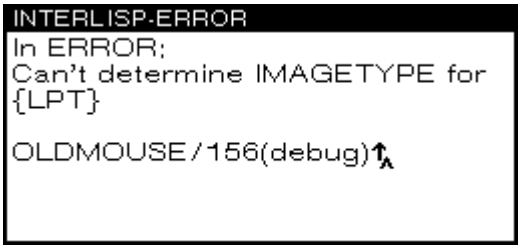


Figure 16-20. A break window indicating that the system cannot find a valid printer.

Type an up-arrow to exit the break and make sure that the printer and its type are properly set.

You can get into this situation by choosing the hardcopy **To a printer** suboption off the standard window menu and then selecting the **Other...** option from the "Which Printer?" menu.

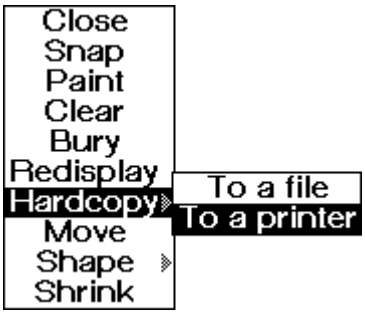


Figure 16-21. The standard right button window menu, showing the hardcopy **To a printer** submenu option selected.

Figure 16-22. The "Which printer?" menu showing the **Other...** option selected.

The **Other...** option does not automatically prompt you for a printer type. Therefore, we recommend that you always add new printers using the background menu **Add Default Printer** submenu option.

Font not Found

This break occurs when you do not have one of your font directories set properly.

Figure 16-21. A break window indicating that the system cannot find the appropriate font.

In this particular case, the system was unable to find an Interpress font. Note that the font description contains the word Interpress. This implies that either there are no Interpress font directories set or that the given Interpress font directories do not contain that font.

Type an up-arrow to exit the break and make sure that the appropriate font directory is properly set. You can do this by selecting the **Show Font Directories** suboption off the **Set Default Printer** option on the background menu. If directories are not correctly set use the **Add Font Directory** and **Remove Font Directory** options to correct the errors.

Can't Convert a POSTSCRIPT for a NIL printer

This break can occur if you have incorrectly typed the case of the printer name.

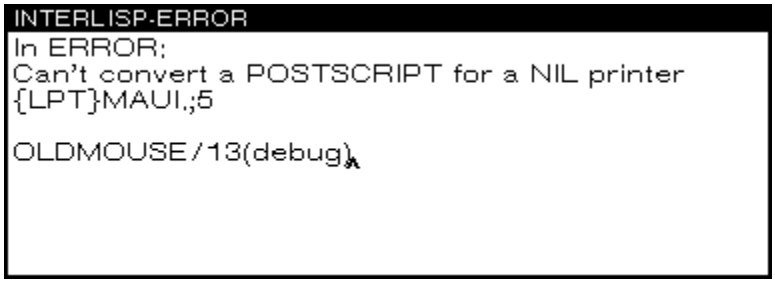


Figure 16-22. A break window indicating that the system cannot find the appropriate printer.

Type an up-arrow to exit the break and if you have entered a printer name in lower case, try entering the same name in upper case by selecting the **Add Default Printer** option off the **Set Default Printer** submenu on the background menu.

Customizing Your PostScript Printer

The PostScript driver allows you to set a number of parameters to modify the output to suit your needs.

Portrait vs. Landscape

There are two terms used to describe how an image is printed on a piece of paper which you need to know.

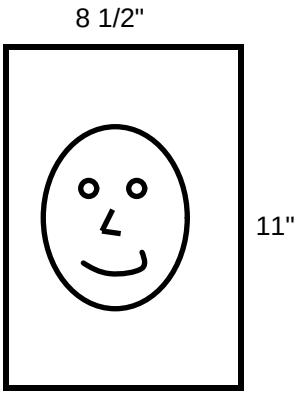


Figure 16-23. Portrait describes the way we normally hold a piece of paper.

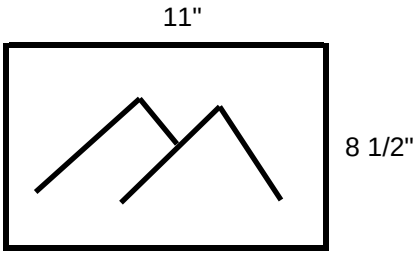


Figure 16-24. Landscape describes a piece of paper held on its side.

Maximum Wild Font Size

Indicates the maximum point size that the system will indicate as available when looking for fonts.

The default value is "72."

Font Association Menu

Maps display font names onto PostScript font names. To change the mapping click the left mouse button on a display font name in the left-hand column then select a PostScript font from the a menu of PostScript fonts names which will appear.

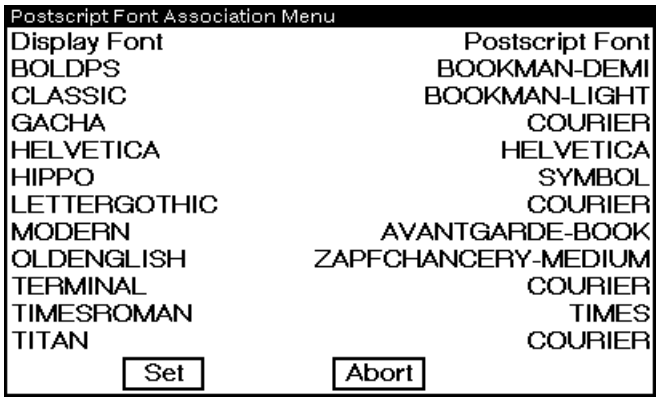


Figure 16-25. The "Postscript Font Association Menu" showing the display fonts on the left and the PostScript printer fonts on the right.

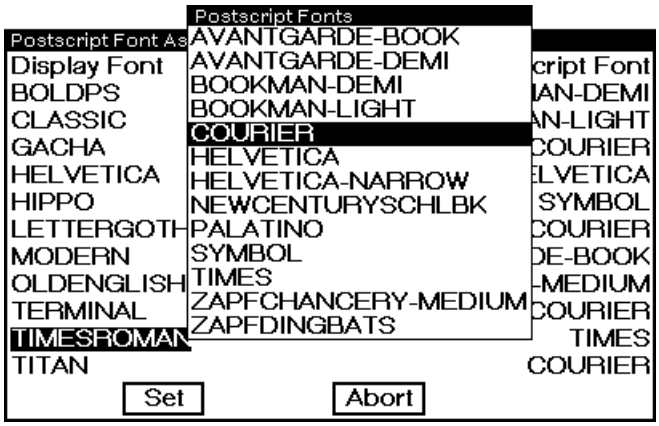


Figure 16-26. The "Postscript Font Association Menu" with the TIMESROMAN display font selected and about to be set to correspond to the COURIER PostScript printer font.

PostScript Short Edge Shift

Is the distance (in points) to shift the image perpendicular to the short edge of the paper. A positive value gives a shift upward in portrait mode, and to the right in landscape mode. A negative value shifts the image down in portrait and to the left in landscaped mode.

The default value is "0."

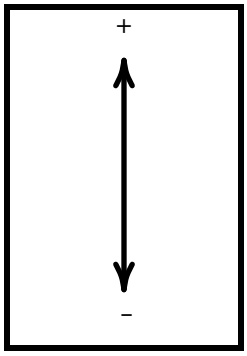


Figure 16-27. The short edge shift in portrait mode.

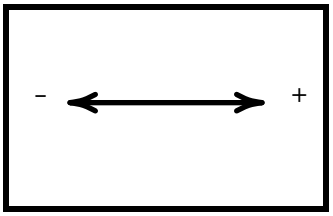


Figure 16-28. The short edge shift in landscape mode.

PostScript Short Edge Points

Indicates the printable region of the page, in points, along the short edge of the paper. It should be adjusted to allow for any short edge shifts of the image.

The default value is "576" points or 8 inches.

PostScript Long Edge Shift

Is the corresponding variable for shifts perpendicular to the long edge of the paper. A positive value here gives a shift to the right in portrait mode and downward in landscape mode.

The default value is "0."

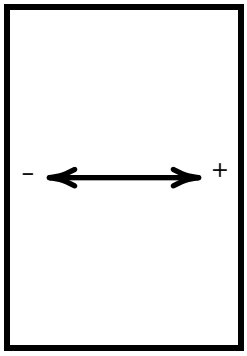


Figure 16-29. The long edge shift in portrait mode.

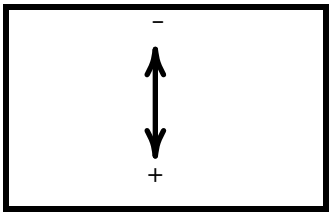


Figure 16-30. The long edge shift in landscape mode.

PostScript Long Edge Points

Indicates the printable region of the page, in points, along the long edge of the paper. It should be adjusted to allow for any long edge shifts of the image.

The default value is "786.24" or "10.92" inches.

Note:

The AST TurboLaser PS has an imageable area on the page which is a different size than that of the Apple LaserWriter. The values of **PostScript Short Edge Points** and **PostScript Long Edge Points** for the AST are "575.76" and "767.76," respectively.

Landscape Mode

Indicates whether the default orientation of output files is landscape.

The default value is "No."

Landscape Text Mode

Indicates if the printing of text files should force the orientation of output files to landscape.

The default value is "No."

Bitmap Scale

Specifies an independent scale factor for display of bit map images, window hardcopies, for example. Values less than 1 reduce the image size, i.e. a value of 0.5 will give a half size bit map image. The position of the scaled bit map will still have the same lower-left

corner, i.e., the scaled bit map is not centered in the region of the full size bit map image.

The default value is "1."

Note:

Setting bit map scale to 0.96, instead of 1, will give cleaner bit map images on a 300 dpi printer. This corrects for the 72 ppi imagestream vs. the 75 dpi printer, using 4x4 device dots per bit map pixel. Also, values of 0.24, 0.48 and 0.72, instead of 0.25, 0.5 and 0.75, will also give cleaner images for reduced size output. In general, use integer multiples of 0.24 for a 300 dpi printer.

Texture Scale

Specifies an independent scale for the display of bit map textures. The value represents the number of printer bits per screen pixel. The default value is 4, which represents each pixel of the texture as a 4x4 block, so that textures are approximately the same resolution as on the screen for 300 dpi output devices, such as the Apple Laserwriter.

The PostScript package extends the allowed representations of a texture, beyond 16-bit integers and 16x16 bit maps, to any square bit map. (If the bit map is not square, its longer edge is truncated from the top or right to make it square.) Use this feature with caution, as large bit map textures, or sizes other than multiples of 16 bits square, require large amounts of storage in the PostScript interpreter, and can cause errors when printing.

Image Size Factor

Specifies an independent factor to change the overall size of the printed image. This resizing affects the entire printed output. Values greater than 1 enlarge the printed image, and values less than 1 reduce it. The **Bitmap Scale** parameter does not consider the **Image Size Factor** when determining the scale factor for a bit map.

[This page intentionally left blank]

17. ERROR RECOVERY AND KNOWN PROBLEMS

NoteCards on the Sun Workstation has an error handling system which includes

the break window error system,

a diagnostic program, URAID, which handles emulator errors.

Occasionally, you may encounter SunOS error messages. Refer to your Sun documentation set for recovery procedures when these errors occur.

Known problems with the NoteCards system are discussed at the end of this chapter.

System Status, Aborting Operations, and Spawning a New Mouse

NoteCards provides a number of keyboard combinations for enquiring about system status and aborting operations.

System Status

Typing CONTROL-T prints status information on the type-in process. The type-in process is the process which has the flashing caret cursor. This is of minimal usefulness as the type-in process is usually just sitting waiting for you to type characters in, and the messages are aimed a people extending the NoteCards system. However, if you need to know if the system is still active, typing CONTROL-T will force the system to give you the status on one process.

Aborting Operations

Hitting the STOP key will frequently abort the process you want it to. However, as the system runs multiple processes the process which receives the stop command may not be the one you intended to receive it .

Typing CONTROL-G gives you a list of all the currently running processes, in menu format, as shown in Figure 17-1.

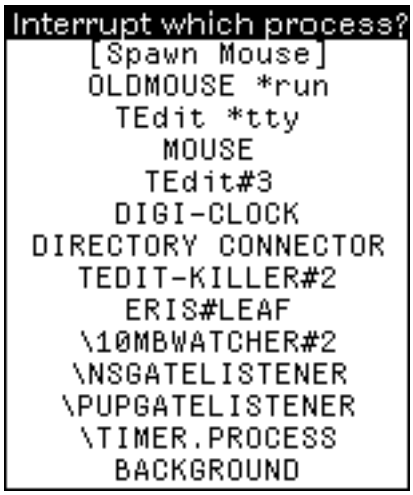


Figure 17-1. The "Interrupt which process?" menu showing all the currently running processes.

Selecting one of these processes will cause a break to occur in that process. You can then uparrow, **^**, out of the break to abort the process. If you decide you have broken the wrong process, you can type **ok** to have the process resume where it left off.

Figure 17-2. A break caused by a CONTROL-G. The OLDMOUSE process is being resumed with the **ok** command.

Because there can be so many processes, TEdit processes in particular (there is one TEdit process for every TEdit window including the expanded-menu command windows), it is important to know which process you want to break to abort an operation. Generally, if you start an operation with the mouse, it either runs under the MOUSE or OLDMOUSE process, which is deleted the moment the operation finishes. If you see an OLDMOUSE process, this is usually the process you want to break.

There are exceptions to the OLDMOUSE rule. Performing operations from the FileBrowser causes a process with an "FB-" prefix to be spawned. All operations on the TEdit expanded menus are run in the main TEdit process associated with that edit window. If you have one TEdit window open on your screen and one TEdit expanded menu open, you will have two TEdit processes running: TEdit, which is the process associated with the main TEdit window, and TEdit#2, which is the process associated with the expanded menu.

Killing the wrong process can cause you to lose information. USE WITH CAUTION!

Spawning a New Mouse

If by some chance you do manage to kill your mouse process off, it is easy to start a new one. Type CONTROL-G and choose the **[Spawn Mouse]** option on the "Interrupt which process?" menu. **[Spawn Mouse]** will always be the very top option. If you create several new mouse processes, the system will remove all but one mouse process.

Break Windows

Occasionally, when performing certain operations like accessing a file which has been moved to another directory, deleted, or renamed, or trying to create a font which does not exist, NoteCards will enter a break. When this happens, NoteCards provides you with information about what caused the break, and asks you what to do.

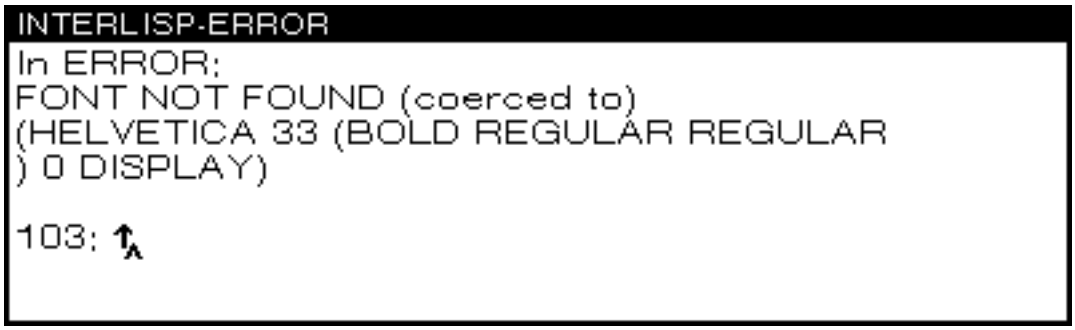


Figure 17-3. A break window caused by a font-not -found error.

In cases like this, you should first determine if the cause of the break is a simple error. Here for instance, there is no Helvetica 33 bold font. There is, however, a Helvetica 32. The appropriate action in this case would be to up arrow out of the break window by typing a **^**, and reexecuting the operation with the correct font.

Another possibility is that the font directories may have been moved and the system may be looking in the wrong place for the fonts. Here again, you would up arrow out of the break window by typing a **^**, edit the initialization file (see Appendix C, Initialization Files) to insert the correct values for DISPLAYFONTDIRECTORIES, load the initialization file, and then retry the operation.

If the break is caused by a missing file, you should up arrow out of the break window by typing a **^**, find the current location of the file, and retry the operation with the new path and file name.

If you are unable to uparrow out of a break, or the system is behaving strangely, you should save all your work by closing your notefiles, and saving your TEdit and Sketch files, perform a **Flush**

image & Exit, and reload your system. **Flush image & Exit** is a submenu off **Logout** on the background menu.

Errors While Running NoteCards

The following errors may occur running NoteCards on the Sun Workstation.

ERROR MESSAGE	REASON/FUNCTION RESPONSIBLE
`File access timed out`	Occurs when you try to access a file on a remotely mounted file system or NFS service that is down.
`File too large`	Self explanatory.
`Too-Many-Files-Open`	Occurs when 1) you exceed the SunOS open file limit (see Chapter 4, System Use Issues) 2) you exceed system file resources while writing a sysout (using the **Logout** command from the background menu.)
`Nonexistent directory`	Occurs when user tries to connect to a nonexistent directory.
`No-Such-Directory`	Occurs when user tries to connect to a nonexistent directory.
`Connection timed out`	Self explanatory.
`Bad Host Name`	Self explanatory.

I/O Errors

These Xerox workstation-specific errors may occur if certain functions are inadvertently used on the Sun Workstation.

ERROR MESSAGE	REASON/FUNCTION RESPONSIBLE
`Floppy: No floppy drive on this machine.`	Self-explanatory.
`Device error: {FLOPPY}`	Occurs when you try to access a floppy device while running on the Sun Workstation.
`Wrong machinetype`	Occurs when functions controlling Xerox disk drive device-specific behavior are entered when running on a Sun.

Virtual Memory Errors

ERROR MESSAGE	REASON/FUNCTION RESPONSIBLE
`File-System-Resources-Exceeded`	Logout, Save image & Exit, Save VM
`Protection-Violation`	Logout, Save image & Exit, Save VM
`File-Wont-Open`	Logout, Save image & Exit, Save VM

URAID

The NoteCards system normally operates as a self-contained environment. In some unusual circumstances NoteCards may encounter a situation from which it cannot recover. In this case, when an unrecoverable emulator error is encountered, the emulator halts and enters into a small debugger called URAID. URAID allows you to inspect memory, or to look inside the sysout file, and attempt to recover from the error.

If you produce the same type of error condition in NoteCards on a Xerox workstation as you did on a Sun Workstation, you get an MP error instead of a URAID error.

Entering URAID

Normally, the emulator automatically enters URAID when an unrecoverable emulator error occurs. If the system freezes and will not let you regain control, you can throw the system into URAID by simultaneously holding down the SHIFT, CONTROL and DELETE (L10) keys or the SHIFT, CONTROL, and NEXT (ALTERNATE) keys.

URAID Commands

URAID has a few simple commands which you can use to attempt error recovery. All URAID commands are case sensitive.

h Hard Reset. Attempts to recover by resetting the Lisp stack. Quits URAID and causes NoteCards to resume execution. This command should not be used unless you are sure that execution can be resumed.

e Exit to SunOS. NoteCards will end. If you are going to call Customer Support, call *before* you give this command. After giving this command, nothing is recoverable.

q Quits URAID and returns to NoteCards.

Note: An error may occur while the NoteCards system is running uninterruptably. The following message signals this error:

```
Error in uninterruptable system code -- ^N
to continue into error handler
```

Disregard the ^N command; it is not supported by URAID. Use the **q** command to continue.

Other Fatal Error Conditions

Occasionally, other emulator, operating system, or system administration errors may occur from which the URAID program cannot recover. Such error conditions include a process dying, the emulator going into an infinite loop, the keyboard being lost, or the system freezing up.

If any of these emulator errors occur, use the UNIX `kill` command to kill the `lde` process.

System Error Conditions

The following are error messages generated by SunOS. For complete information on these error messages, see the *SunOS Reference Manual*, Intro(2).

ERROR MESSAGE		DESCRIPTION
:0	Unused	
:1	EPERM	Not owner
:2	ENOENT	No such file or directory
:3	ESRCH	No such process
:4	EINTR	Interrupted system call
:5	EIO	I/O error
:6	ENXIO	No such device or address
:7	E2BIG	Arg list too long
:8	ENOEXEC	Exec format error
:9	EBADF	Bad file number
:10	ECHILD	No children
:11	EAGAIN	No more processes
:12	ENOMEM	Not enough core
:13	EACCES	Permission denied
:14	EFAULT	Bad address
:15	ENOTBLK	Block device required
:16	EBUSY	Mount device busy
:17	EEXIST	File exists
:18	EXDEV	Cross-device link
:19	ENODEV	No such device
:20	ENODIR	Not a directory
:21	EISDIR	Is a directory
:22	EINVAL	Invalid argument
:23	ENFILE	File table overflow

:24	EMFILE	Too many open files
:25	ENOTTY	Not a typewriter
:26	Unused	
:27	EFBIG	File too large
:28	ENOSPC	No space left on device
:29	ESPIPE	Illegal seek
:30	EROFS	Read-only file system
:31	EMLINK	Too many links
:32	EPIPE	Broken pipe
:33	EDOM	Math argument
:34	ERANGE	Result too large
:35	EWOULDBLOCK	Operation would block
:36	EINPROGRESS	Operation now in progress
:37	EALREADY	Operation already in progress
:38	ENOTSOCK	Socket operation on non-socket
:39	EDESTADDRREQ	Destination address required
:40	EMSGSIZE	Message too long
:41	EPROTOTYPE	Protocol wrong type for socket
:42	ENOPRTOOPT	Bad protocol option
:43	EPROTONOSUPPORT	Protocol not supported
:44	ESOCKTNOSUPPORT	Socket not supported
:45	EOPNOTSUPP	Operation not supported on socket
:46	EPFNOSUPPORT	Protocol family not supported
:47	EAFNOSUPPORT	Address family not supported by protocol family

ERROR MESSAGE		**DESCRIPTION**
:48	EADDRINUSE	Address already in use
:49	EADDRNOTAVAIL	Can’t assign requested address
:50	ENETDOWN	Network is down
:51	ENETUNREACH	Network is unreadable
:52	ENETRESET	Network dropped connection on reset
:53	ECONNABORTED	Software caused connection abort
:54	ECONNRESET	Connection reset by peer
:55	ENOBUFS	No buffer space available
:56	EISCONN	Socket is already connected
:57	ENOTCONN	Socket is not connected
:58	ESHUTDOWN	Can’t send after socket shutdown
:59	Unused	
:60	ETIMEDOUT	Connection timed out
:61	ECONNREFUSED	Connection refused
:62	ELOOP	Too many levels of symbolic link
:63	ENAMETOOLONG	File name is too long
:64	EHOSTDOWN	Host is down
:65	EHOSTUNREACH	No route to host
:66	ENOTEMPTY	Directory not empty
:67	Unused	
:68	Unused	
:69	EDQUOT	Disc quota exceeded
:70	ESTALE	Stale NFS file handle
:71	EREMOTE	Too many levels of remote in path
:72	ENOSTR	Not a stream device
:73	ETIME	Timer expired
:74	ENOSR	Out of stream resources
:75	ENOMSG	No message of desired type
:76	EBADMSG	Not a data message

:77	EIDRM	Identifier removed

Known Problems

Printing Browsers

Printing Browsers which contain dashed lines may cause your printer to hang.

Notefile Indicator Window

Occasionally, after you have reshaped a card, a piece of the text in the notefile indicator window will be garbled or broken. To redisplay the text in this window, place the point of the mouse cursor in the notefile indicator window, hold down the right mouse button, and select the **Redisplay** option from the window menu which pops up.

Fonts

The PostScript Zapfchancery font does not work at this point in time.

TEdit

When a TEdit window is positioned such that part of the display region of the window is off the screen, TEdit will display some lines of text in a garbled fashion. Simply move the window so that the entire extent of the window is on the screen or redisplay the window using the **Redisplay** option on the title-bar right-button mouse menu.

[This page intentionally left blank]

APPENDIX A NOTEFILE CONCEPTS

This document provides a brief explanation of the structure of notefiles. It also describes how checkpointing, aborting, and recovering a notefile after a crash work. Finally, it describes the use of **Inspect & Repair** to repair a notefile with inconsistent links. For additional information see Appendix B, The Notefile Inspector.

Note: The information in this appendix was copied whole from the documentation for a prerelease of NoteCards and has not yet been updated for this release. As such, it may be inaccurate in places.

Background Concepts

Unique identifiers: UIDs

All objects in NoteCards (i.e., cards, links, notefiles) are assigned a unique identifier called a UID. Each UID is a 112-bit number that is guaranteed to be unique across all time and space. UIDs are used in many places in NoteCards as keys for indexing and retrieving cards, links, and notefiles.

Card parts

For storage purposes a note card is decomposed into 4 independent parts: contents, title, property list, and links. Each of these parts is stored separately in the data area of the notefile. This is discussed in the Notefile Structure section below. When a card is saved, only those card parts that have changed are rewritten in the notefile.

The contents of the card are stored on the notefile in a manner appropriate to its type. Thus a Text card's content is a text stream and is written on the notefile exactly the way TEdit writes out text streams (i.e., text followed by "looks" information). In contrast, all titles are stored as strings and all property lists as standard Lisp lists. Storage of Links is described in Section 4 below.

Notefile Structure: index and data areas

A notefile consists of three parts, a header, an index, and a data area. The header and the index are fixed in size for each notefile. The data area follows the index area and grows as cards are added to or modified in the notefile.

The notefile header contains the following information about the notefile: its UID, a number identifying its format, the checkpoint pointer, the size of the index, and a pointer to the next available index entry. If the notefile header is destroyed, it cannot be automatically reconstructed. Careful hand manipulation of the

notefile by a NoteCards wizard is required to recover a notefile with a bad header.

The structure of the index and data area is shown in Figure A-1 .

The index contains a fixed number of index entries. Each index entry that is in use, contains information for one of the cards in the notefile. Specifically, an index entry contains 5 fields: a status character, the card's UID, and 4 pointers. The status character specifies whether the index entry is free (not in use) or contains information for an active or a deleted card. If the index entry is not free, the UID field contains the UID of the card refered to by this index entry and the four pointer fields contain the location in the data area of the 4 parts of its card: contents, title, links and property list.

The number of index entries is fixed at notefile creation time. The default is 1000 entries. The number of index entries is automatically doubled by the notefile compactor if 75% of the entries are used. The compactor also frees (i.e., makes unused) all of the index entries that refer to deleted cards. In normal operation, NoteCards prints a warning whenever more than 90% of the index entries in a notefile are used. At this point, the notefile should be compacted to increase the index size.

The data area contains the actual information about the card. Whenever you change, say, the title of a card, the new title is written at the end of the data area. The title pointer in the card's index entry is then updated to point to this new location in the data area. Thus, in general, a notefile's data area grows every time any part of any card is changed.

The old information, now somewhere in the middle of the data area, is not removed. However, it is no longer directly accessible because there is no index entry that points to it. Thus, for most purposes this old information can be considered "dead space" in the notefile. The notefile compactor rewrites the notefile, eliminating all such dead space.

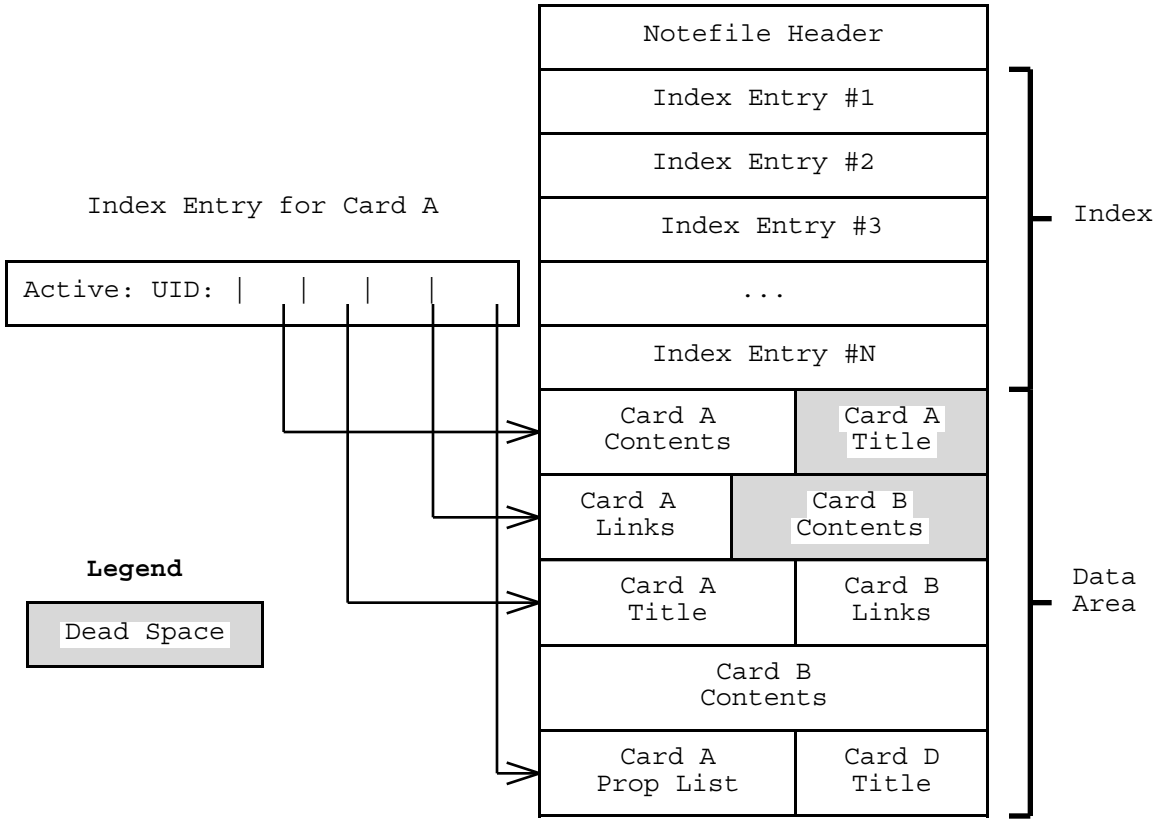


Figure A-1. The structure of a notefile.

As long as the notefile has not been compacted, all the old information can be accessed (and made to be "current") via the notefile Inspect and Repair facility. Inspect and Repair does this by ignoring the index and parsing the entire data area to produce a listing of all the information (both current and old) about a card that is stored on the data area. See the Inspect and Repair manual for more information.

Notefile Checkpointing

As long as a notefile is open, its index area is cached in memory. When a card part is saved, the card part's information is written to the end of the data area but the card's actual index entry on the notefile is not updated with this new location. Instead, the appropriate pointer in the card's in-memory index entry is updated. Thus, the index in the notefile continues to point to the old information in the data area, while the in-memory index points to the new information.

Thus while a notefile is open, its current state is distributed between the actual notefile and information cached in memory. The current index is cached in memory. For cards open on the screen (or cached in memory from the programmer's interface), the "current" card part information is contained in an in-memory cache. For all

other cards, the current information is contained in the data area of the notefile pointed to by the in-memory index entry.

If a machine crashes while a notefile is open, the information cached in memory is lost. A crash not only discards changes made to cards on the screen, but its also leaves the information stored on the notefile in an inconsistent state. For example, the index on the notefile may point to old information in the data area. This occurs because the new information (e.g., a new title) is written to the data area but only the in-memory index pointers (which are lost in the crash) have been updated to point to the new information.

Checkpointing forces all of the in-memory information to be written onto the notefile. Specifically, checkpointing causes all open cards to be saved and the in-memory index to be written to the notefile's index area. Thus, immediately after checkpointing, the notefile itself contains its current (and consistent) state. If the machine were to crash at this point, no information would be lost and the notefile would be consistent.

Checkpointing also writes a **checkpoint pointer** onto the notefile header. The checkpoint pointer contains the location of the end of the data area (i.e., the end of the notefile) at the time the checkpoint is done.

As the notefile is used after the checkpoint, information is written in the data area past the checkpoint pointer but only the in-memory index entries are updated to point to this information. The on-file index entries still point to the information in the data area *before* the location referenced by the checkpoint pointer. Thus, a consistent notefile can be constructed from the index area and all of the information in the data area located before the checkpoint location. This is essentially the notefile as it was at the time of the last checkpoint. (Note: one small exception is that changes to a card's size on the screen are actually written in the middle of the data area rather than at the end. Thus, truncating a notefile to its checkpoint location cannot "undo" the reshaping of a card.)

When opening a notefile after a crash, the system will insure that the notefile is in a consistent state. It does so by truncating the data area to the last checkpoint location, saving the truncated information if requested by the user. This leaves the notefile in the state it was during the last checkpoint before the crash.

Aborting a notefile does the same thing. It truncates the data area to the last checkpoint location, thereby eliminating all changes made to the notefile since the last checkpoint. It also discards the in-memory index. Thus, the notefile is left in the exact state it was after the last checkpoint.

Finally, note that notefile Close forces a checkpoint. Therefore, aborts and recovery after crashes actually restore the NoteFIle to its state as of the last user-initiated checkpoint *or* close.

Storing links and reparing notefiles with inconsistent links

The links card part is divided into three subcomponents: to-links, from-links, and global links. The to-links is a list of all links whose Source is the given card (i.e., that point from the card to some other card). The from-links is a list of links whose Destination is the given card (i.e., that point from some other card to the given card). Finally, the global links is a subset of the to-links that includes only the Global links originating in the given card.

Given this scheme, every link is stored on the notefile in three different places. First, if the link is Local it is stored inside the link icon which in turn is inside the content part of the link's source card. If the link is global, it is stored in the global links subpart of its source card. Second, the link is stored in the to-links list of its source card. Third, the link is stored in the from-links list of its destination card.

These three records of the same link occasionally get out of synch, resulting in an inconsistent notefile. There are a number of symptoms of such inconsistency. For example, the **ShowLinks** display for card may indicate that the card is a destination for a link from some source card X while the ShowLinks display for X does not include a ToLink to that destination card. Occasionally, inconsistent links will also result in link icons that contain the words "DELETED" or "FREE" when displayed on the screen. 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.

One function of the NoteCards **Inspect & Repair** facility is to resynchronize the three records for all links in the notefile. The inspector's third phase rebuilds the links as follows. First it removes all to- and from-links for every card. Then it reads the contents for each card and recreates to-links and from-links by looking at the links found inside the link icons in the card's content and in its global links list. 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.

[This page intentionally left blank]

APPENDIX B NOTEFILE INSPECTOR

The main purpose of the notefile inspector is to repair notefiles by rebuilding their links. However, the inspector is generally useful for checking the health of notefiles, deleting cards and backing up 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.

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 innards of notefile organization. We recommend that you read Appendix A, Notefile Concepts, before continuing.

Note: The information in this appendix was copied whole from the documentation for a prerelease of NoteCards and has not yet been updated for this release. As such, it may be inaccurate in places.

Running the Notefile Inspector: Phase 1: Scouring the Data Area

To start the inspector on a notefile, first be sure that the notefile is not open. Then select the item **Inspect&Repair** from one of the NoteFile Ops menus (i.e. from the notefile Banner, the MenuBox icon or the notefile FileBrowser). 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. We recommend 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 yyyyy."

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 cards and the number of deleted ones. 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 1.)

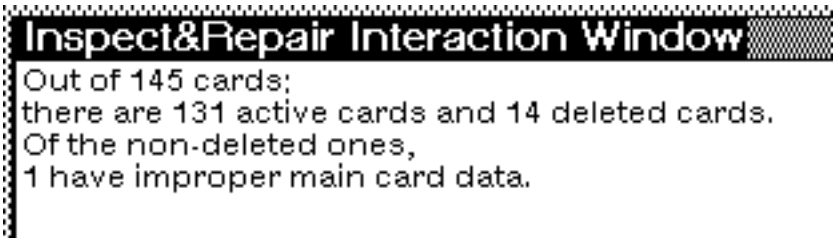


Figure B-1: Snapshot of a sample interaction window

Normally, the messages describing problems are of the form shown in Figure 1, namely a count of the number of cards broken in a given way. Some messages are a bit trickier. If you have fileboxes with bad substances (i.e. main card data), then you're told that if you don't wish to delete or back up such fileboxes to previous versions, then phase 3 will rebuild them. If, however, the filebox is one of the top level boxes (Contents, Orphans, ToBeFiled), then deletion is not an option. (See Section 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 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 B-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 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 of 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 titles of active cards 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 titles for deleted cards as well as active ones. Using this option, one can undelete deleted cards and restoring them to some previous version.

Running the Notefile Inspector: Phase 2: Your Chance to Tinker

After selecting **Inspect Cards** in the interaction window's attached menu, a menu containing numbered titles of notecards pops up and is attached to the interaction window's lower left corner. It contains the first 10 characters of titles 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 100 card titles. If your notefile has more than that, then the menu is composed of several pages each containing around 100 cards. Rapid switching between pages is possible. Figure 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 100 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 the previous section.)

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 the previous section 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 those cards having titles matching a given string. This works much like the Search card in Notecards. That is, you specify a string and the list of cards whose titles contain that string is printed out. This is handy, for example, if you want to undelete a card, but only know its title. Note that the search is case *in*sensitive.

The card inspector menu

In the card inspector menu, those titles 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 entries indicating the problem. For example, in the card inspector menu shown in Figure 3, the shaded menu item "10: BNF for Li|S" indicates that the card with title beginning "BNF for Li" has bad substance. 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 meaning is that the current version of that card part is beyond the last checkpoint pointer. (There may have been a crash, for example, thus preventing the notefile from closing normally.)

Figures 3 and 4 show both pages of a two page card inspector including deleted cards. Sometimes deleted cards (e.g. "110: NIL|LSPT" in Figure 4) show up shaded with LSPT 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 6 and discussion below.)

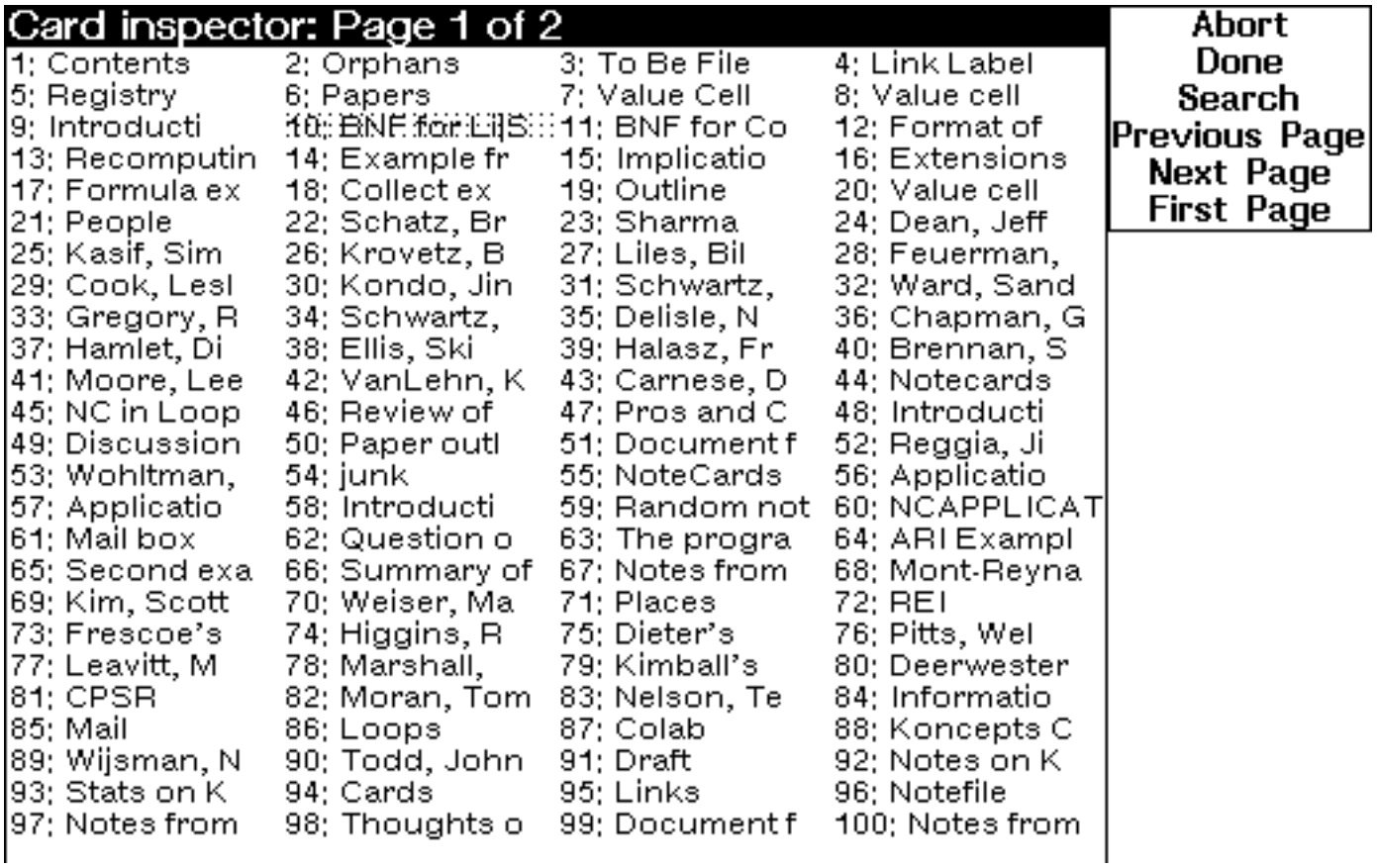


Figure B-3: A card inspector menu: Page 1.

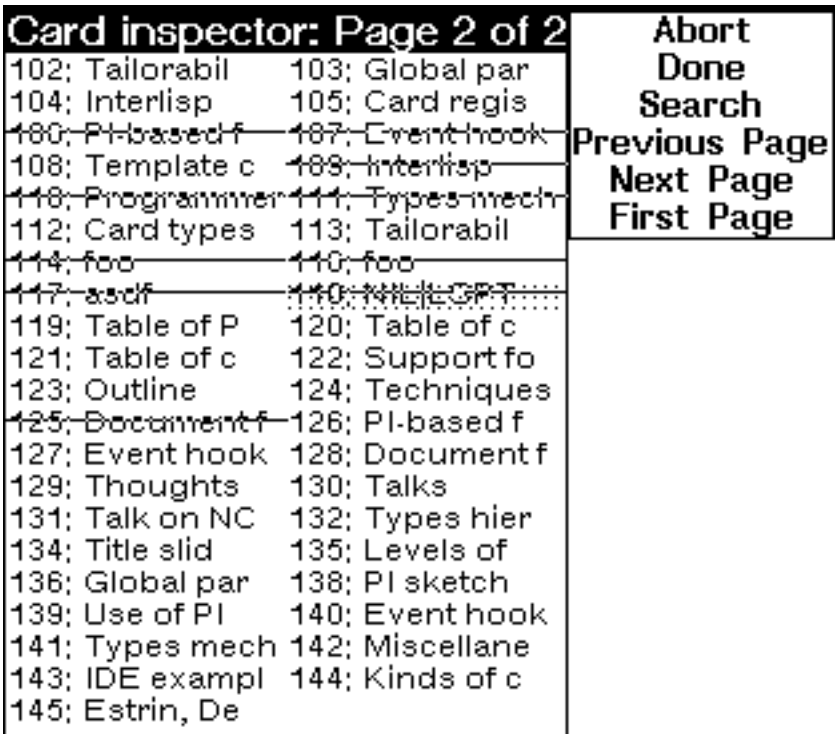


Figure B-4: A card inspector menu: Page 2.

At the top of the first page of any notefile's card inspector, there are entries for the top level boxes and special cards. These are labeled

"1: Contents", "2: Orphans", "3: To Be File", "4: Link Label", and "5: Registry".

Note that if the inspector shows no entry in the card inspector menu for some card, 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 card title entry 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 cards cannot be deleted and thus their popup menus only contain the Inspect option. These are the 5 special cards mentioned above.

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 entry brings up a card parts inspector for that card.

The card parts inspector

Figures 5 and 6 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.

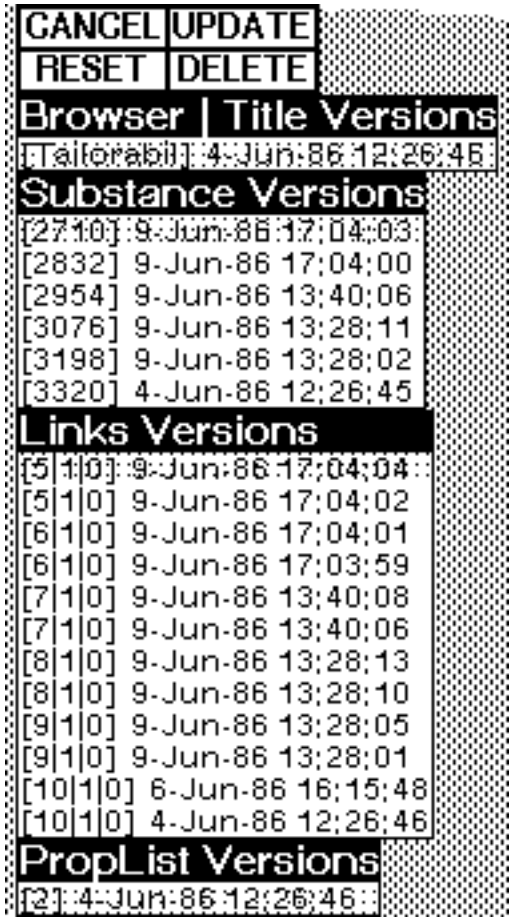


Figure B-5: Card parts inspector for a browser card.

In Figure B-5, the four menus contain entries for every valid version of card parts for the browser card with title "Tailorability". The top menu is for versions 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 6 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. (Very old notefiles may show "NO DATE AVAILABLE" in place of a real date.)

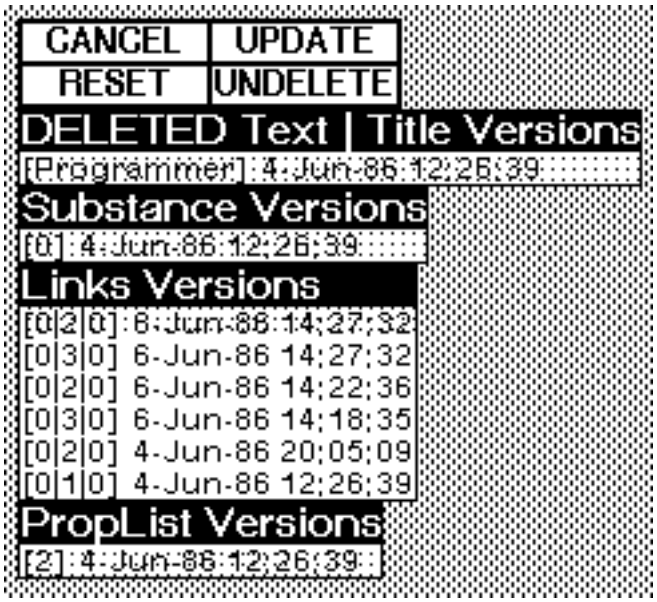


Figure B-6: Card parts inspector for a deleted card.

Figure B-6 shows a card parts inspector for a deleted text card and thus allows the UNDELETE option. By clicking UNDELETE, this card can be backed up to its last version.

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. In addition, each menu item contains some 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.

CANCEL

This cancels 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.

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 CANCEL and then inspecting this card 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 B-7 below.) For example, inspecting a title version brings up the Interlisp inspector on a record containing the title, date and card object. Similarly, inspecting a links or prop list version brings up the Interlisp inspector on a record containing the lists of links (i.e. to links, from links and global links) or the prop list. Note that changing values in these Interlisp inspectors has no affect on the notefile and is ignored.

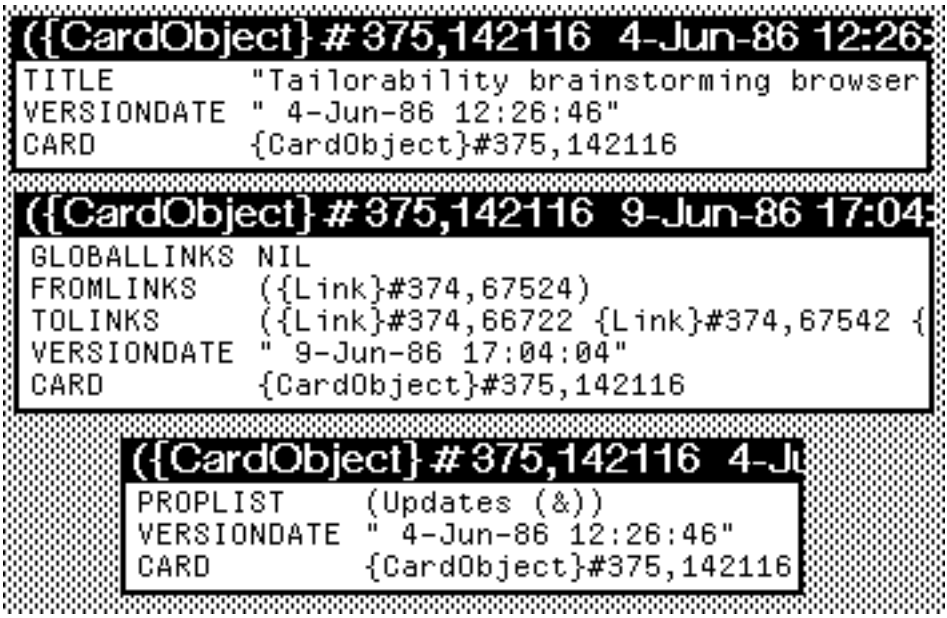


Figure B-7: Card part version inspectors for title, links and prop list.

All substance versions for card types inheriting from Text, Sketch and Graph can currently be inspected. (This includes most cards except those having user-defined substance types, like the NCFile 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 affect 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 canceling the card parts inspector as well as by later aborting the card inspector or interaction window.)

Running the Notefile inspector: Phase 3: Rebuilding your 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.)

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.

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 was inadvertantly deleted. Or back up a card that was accidentally changed to its original 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

It used to be the case that old versions of card parts were in the notefile but inaccessible. Thus, there was little reason not to compact a 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 closes the notefile before starting

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 entry 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

APPENDIX C. INITIALIZATION FILES

Once your system administrator has installed the NoteCards software on your Sun, you can customize your NoteCards environment. The initialization file allows you to specalize your NoteCards environment to the idiosyncracies of your individual site. This file is automatically loaded when you start a fresh sysout.

Site Initialization File

When NoteCards starts, it reads in a site initialization file. This file sets things like the pointer to fonts, printers, and other site-specific parameters.

Locating the Initialization File

When running under SunOS, NoteCards looks for a site initialization file in a number of locations.

LDEINIT

If the UNIX environment variable `LDEINIT` is set to a complete NoteCards file name, NoteCards looks there first for the site initialization file:

```
prompt% setenv LDEINIT /users/smith/site-init.lisp
```

- or -

```
prompt% setenv LDEINIT /users/smith/nc-init
```

/usr/local/lde/site-init.lisp

If `LDEINIT` is not set or there is no file with the name given, NoteCards looks for a site initialization file called `/usr/local/lde/site-init.lisp`. The distribution tape contains a standard site initialization file in the NoteCards lisp directory `/usr/local/lde/lisp/init.NoteCards` which is linked to `/usr/local/lde/site-init.lisp`. You or your system administrator should customize `this file` for your site. The comments below and in the standard `site-init.lisp` describe the parameters it sets and gives some guidelines for customizing it to your local conditions.

{DSK}INIT.LISP

NoteCards looks for a site initialization file on your NoteCards home directory (`{DSK}`). Chapter 4, System Use Issues, describes the `{DSK}` device.

Finally, to load an initialization file after you have been working in a sysout , FileBrowse the directory the initialization file is in, select the file with the left mouse button, and select the **Load** option. The file will load and reset all of the initialization variables.

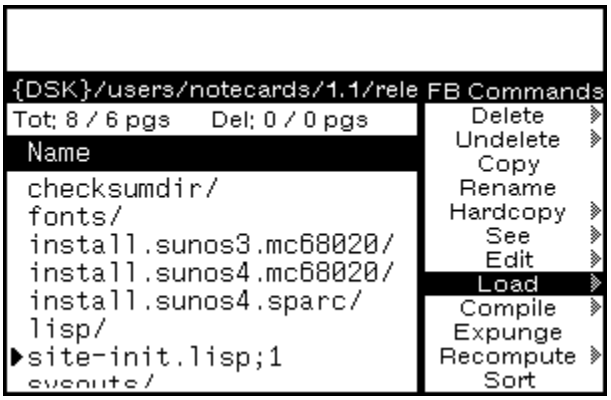


Figure C-1. A FileBrowser loading an initialization file.

Initialization File Structure

The structure of each command in the initialization file is as follows:

(SETQ *variable-name value*)

Each command must appear between parentheses. There must always be an even number of parentheses. Every open parenthesis must have a corresponding close parenthesis. `SETQ` is the assignment command. The variable name is always a single word, and the value is usually either a single word preceded by a single quote mark (') or a list of words enclosed in parentheses preceded by a single quote mark.

The initialization file must always start with two specific lines. The first line must be a comment line, indicated by starting the line with a semicolon (;). The second line must contain the command `(CL:IN-PACKAGE "INTERLISP")`.

Site Variables

The following NoteCards variables should be set in your site initialization file.

LISPUSERSDIRECTORIES The list of paths to search for library files. Every path in this list should also be in `DIRECTORIES`.

```
(SETQ LISPUSERSDIRECTORIES
  '("{DSK}/usr/local/lde/lispusers/"
    "{DSK}/usr/local/lde/lisplibrary/"))
```

Note that each directory name is a string enclosed in double quotes ("). Unless you are a developer who is modifying NoteCards, there will probably never be any reason to change these values. You only need to change the value for this variable if you move the files in these directories to some other location, such as a central fileserver.

DIRECTORIES The list of paths to search for files that are not found in the current NoteCards connected directory. The current NoteCards connected

directory is distinct from the current UNIX connected directory, and NoteCards will not necessarily search for files in the UNIX connected directory.

```
(SETQ DIRECTORIES (COPYALL LISPUSERSDIRECTORIES))
```

Note that there is *no* quote mark (') before the parenthesis in front of `COPYALL`. If you set `DIRECTORIES` in this fashion, this command must follow the one which sets `LISPUSERSDIRECTORIES`.

You can also simply copy the directories from `LISPUSERSDIRECTORIES`, e.g.

```
(SETQ DIRECTORIES
  '("{DSK}/usr/local/lde/lispusers/"
    "{DSK}/usr/local/lde/lisplibrary/"))
```

Unless you are a developer who is modifying NoteCards, there will probably never be any reason to change these values.

DISPLAYFONTDIRECTORIES A list of directories to search when the system is looking for display fonts. The site initialization file should set `DISPLAYFONTDIRECTORIES` to a list of of path names where each path name is represented as a string in double-quotes ("), e.g.,

```
(SETQ DISPLAYFONTDIRECTORIES
 '("{DSK}/usr/local/lde/fonts/display/presentation/"
   "{DSK}/usr/local/lde/fonts/display/publishing/"
   "{DSK}/usr/local/lde/fonts/display/printwheel/"
   "{DSK}/usr/local/lde/fonts/display/miscellaneous/"
   "{DSK}/usr/local/lde/fonts/display/jis1/"
   "{DSK}/usr/local/lde/fonts/display/jis2/"
   "{DSK}/usr/local/lde/fonts/display/chinese/"))
```

If you remove the Chinese and Japanese fonts from the font directories it is not necessary to edit this variable to reflect the change.

You only need to change the value for this variable if you move the fonts to some other location, such as a central fileserver. If you do change the value for this variable, note that each location is represented as a double quoted string and that each path name must end with a slash (*/*).

INTERPRESSFONTDIRECTORIES A list of directories for the system to search for Interpress font widths files. The site initialization file should set `INTERPRESSFONTDIRECTORIES` to a list of of path names where each path name is represented as a string in double-quotes ("), e.g.,

```
(SETQ INTERPRESSFONTDIRECTORIES
'("{DSK}/usr/local/lde/fonts/interpress/presentation/"
  "{DSK}/usr/local/lde/fonts/interpress/publishing/"
  "{DSK}/usr/local/lde/fonts/interpress/printwheel/"
  "{DSK}/usr/local/lde/fonts/interpress/miscellaneous/"
  "{DSK}/usr/local/lde/fonts/interpress/jis1/"
  "{DSK}/usr/local/lde/fonts/interpress/jis2/"
```

```
"{DSK}/usr/local/lde/fonts/interpress/chinese/"))
```

If you remove the Chinese and Japanese fonts from the interpress font directories it is not necessary to edit this variable to reflect the change.

You only need to change the value for this variable if you move the interpress fonts to some other location, such as a central fileserver. If you do change the value for this variable, note that each location is represented as a double quoted string and that the lowest subdirectory name in each path must end with a slash (/).

POSTSCRIPTFONTDIRECTORIES The list containing the name of the PostScript font width files, for PostScript printers.

```
(SETQ POSTSCRIPTFONTDIRECTORIES
   '("{DSK}/usr/local/lde/fonts/postscript/"))
```

You only need to change the value for this variable if you move the PostScript fonts to some other location, such as a central fileserver. If you do change the value for this variable, note that the location is represented as a double quoted string and that the lowest subdirectory name in the path must end with a slash (/).

USERGREETFILES The list of places to search for personal initialization files. If not set in the site initialization file, no personal initialization file is used. The list should be similar to the following

```
(SETQ USERGREETFILES
   '(("{DSK}~/lde/INIT.LISP") ("{DSK}~/INIT.LISP")))
```

This will search for the file `INIT.LISP` in your home directory and in the subdirectory lde immediately under your home directory. The file `INIT.LISP` can be renamed to what ever you want it to be, but it has to be a name which is used by all users of the NoteCards system.

Unless you are a developer who is modifying NoteCards, there will probably never be any reason to change these values or use user greet files.

DEFAULTOSTYPE Specifies the default operating system type for file servers.

```
(SETQ DEFAULTOSTYPE 'UNIX)
```

Unless you are using other than Sun NFS (Network File System) servers there is no reason to change the value of this variable.

DEFAULTPRINTINGHOST A list of names of default printers.

```
(SETQ DEFAULTPRINTINGHOST NIL)
```

`NIL` is the default setting and signifies that there are no default printers specified. This variable can also be set from the **Set Default Printer** background menu option, (see Chapter 16, Printing).

The following example shows how to set `DEFAULTPRINTINGHOST` to a mixed list of PostScript and Interpress printers.

```
(SETQ DEFAULTPRINTINGHOST
```

```
   '(MAUI |Tremor:mv:envos|))
```

- or -

```
(SETQ DEFAULTPRINTINGHOST
   '(MAUI "Tremor:mv:envos"))
```

MAUI is the name of a PostScript printer, |Tremor:mv:envos| the name of an interpress printer. Each interpress printer name must be surrounded by vertical bars (|) or double quotes (").

DEFAULTPRINTERTYPE The default printer type for all the available printers.

```
(SETQ DEFAULTPRINTERTYPE NIL)
```

Possible values are 'POSTSCRIPT, 'INTERPRESS, and NIL meaning unspecified. Note that 'POSTSCRIPT and 'INTERPRESS are all preceded by single quote marks ('), but that NIL is not.

If all your printers are PostScript printers you would make the following setting.

```
(SETQ DEFAULTPRINTERTYPE 'POSTSCRIPT)
```

If all your printers are not of the same type set DEFAULTPRINTERTYPE to NIL, and add the following command for each printer.

(PUTPROP *'Printer-Name* **'PRINTERTYPE** *'Type***)**

Where *'Printer-Name* is the name of the printer and *'Type* is one of 'POSTSCRIPT, 'INTERPRESS, or 'PRESS, for example.

```
(PUTPROP 'MAUI 'PRINTERTYPE 'POSTSCRIPT)

(PUTPROP '|Tremor:mv:envos| 'PRINTERTYPE 'INTERPRESS)
```

IDLE.PROFILE The lines below set the IDLE.PROFILE such that any user with a valid UNIX login is allowed to exit idle mode.

```
(LISTPUT IDLE.PROFILE 'AUTHENTICATE 'UNIX)
```

The value of AUTHENTICATE determines what mechanism is used to check passwords. If 'UNIX, it checks your password with SunOS. If T, it uses the NS authentication protocol (this requires the presence of an NS Authenticatioin server accessible via the network). If NIL, the password is not checked at all--any password is accepted. The latter is only useful if ALLOWED.LOGINS contains *.

```
(LISTPUT IDLE.PROFILE 'ALLOWED.LOGINS '(*))
```

The value of ALLOWED.LOGINS determines who is allowed to exit idle mode. If the value is NIL no login is required at all to exit idle mode. If the value is '(*), the system requires a login but lets anyone exit idle mode.

```
(LISTPUT IDLE.PROFILE 'SAVEVM NIL)
```

This line sets the IDLE.PROFILE such that NoteCards does not save the virtual memory file to disk when it enters idle mode. To

have NoteCards automatically save the virtual memory file to disk when it enters idle mode, replace `NIL` in the line above with `T`.

\\BeginDST The day of the year on or before which Daylight Savings Time takes effect (i.e., the Sunday on or immediately preceding this day). Must be set to 98 in the USA if NoteCards is to perform time computations correctly. (Note: This number is subject to future Congressional legislation.) If you are in a region where Daylight Savings Time is not observed, use the value 367.

```
(SETQ |\\BeginDST| 98)
```

\\EndDST The day of the year on or before which Daylight Savings Time ends. Must be set to 305 in the USA.

```
(SETQ |\\EndDST| 305)
```

XCL:*LONG-SITE-NAME* A long version of your company's name, for example, "Envos Corporation."

```
(SETQ XCL:*LONG-SITE-NAME* "Envos Corporation")
```

XCL:*SHORT-SITE-NAME* A short version of your Company's name, for example, "Envos."

```
(SETQ XCL:*SHORT-SITE-NAME* "Envos")
```

[This page intentionally left blank]

APPENDIX D. CHECKSUM CONTROL

If you encounter inexplicable problems shortly after you install NoteCards, they may be due to files being corrupted — the release tape may have been damaged, errors may have occurred while the tape was being read, etc. If you have unexplained problems, we recommend that you verify the checksums of your installed files.

Description

The script generates checksum files named *filename*.check and compares them to the released *filename*.sum residing in the /checksumdir subdirectory.

The checksum script reports inconsistent files, the correct checksum values for the files, and an error message. The checksum of individual files can be generated with the UNIX command: **sum** *filename*.

Commands

ldechecksum [**-cg**] *[ncdir [dir | dirgroup]]* **[Command]**

-c Generates checksums for your installed files and compares them with correct values. This is the default action.

-g Generates checksums for the files specified.

ncdir Name of the NoteCards installation directory. Usually it is **/usr/local/lde**.

dir Any specific directory residing under *ncdir*. Only relative pathnames with respect to *ncdir* are accepted.

dirgroup The directory group. Valid directory groups are **system**, **fonts**, and **all**.

- **system** includes lisp, sysouts, install.sunos3.mc68020, install.sunos4.mc68020, install.sunos4.sparc.
- **fonts** includes the display, interpress, press and postscript directories.
- **all** includes all of the above.

Output

As it begins checking each directory, the script prints a message in the form:

```
Checking directory: /usr/local/lde/subdir
```

Error and warning messages may be in one of two forms:

```
< E > 36610 6973 NoteCards.sysout
```

Indicates that file NoteCards.sysout is erroneous or does not reside in the directory. The correct checksum of 36610, together with the size (6973 kbytes) of the file, are shown.

```
< W > /usr/local/lde/fonts/display/chinese:
Directory not installed
```

Indicates that Chinese fonts were not installed or were removed after NoteCards was installed.

Examples

prompt% **ldechecksum /usr/local/lde**

All files in the installed NoteCards directories in **/usr/local/lde** are checked.

prompt% **ldechecksum /usr/local/***somedir***/lde system**

This example checks all files in:

```
/usr/local/somedir/lde/install.sunos3.mc68020
/usr/local/somedir/lde/install.sunos4.mc68020
/usr/local/somedir/lde/install.sunos4.sparc
/usr/local/somedir/lde/sysouts
/usr/local/somedir/lde/lisp
```

prompt% **cd/usr/local/lde**

prompt% **ldechecksum -c . fonts/display**

This example checks only the display font directories. The period (**.**) is used because you are positioned under the current NoteCards installation directory.

[This page intentionally left blank]

GLOSSARY

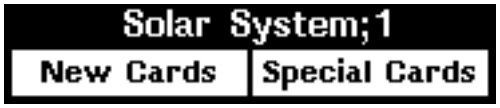
abort To terminate any action before it is finished. When selected from the "Notefile Ops" menu, **Abort** means to close a notefile without saving your changes to the notefile.

back links System generated to-links. These links point from a Document or LinkIndex card back to the source cards from which the Document or LinkIndex was built. Hence the name "back link."

background menu The menu brought up by holding the right mouse button down in the gray region not associated with any windows.

backward links The same as from-links. This term is used when setting Browser and LinkIndex card parameters.

Banner The location from which you access old cards and create new ones. Also called "notefile Banner." The "Solar System" Banner is shown below.



bit map A two-dimensional grid of zeros and ones where a zero represents white space and a one represents black space. The screen is a bit-mapped display.

break The state entered by NoteCards during error processing that allows you to recover from the error by typing Lisp commands in the break window. If you don't know what to do with a break, type ^ after the prompt to abort the operation, then start over.

When a break has occurred, we recommend that you perform an **Inspect & Repair** on the notefile after you have closed it.

break window A window that opens when a break occurs. The title is usually the name of the broken function.

button A button on the two- or three-button mouse.

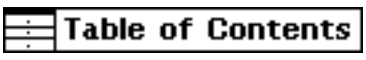
card Refers to any card type in the NoteCards system. Contrast with the entry for **note card**.

card contents The text or graphics in the body of a card.

card menu The menu brought up when you hold the left mouse button down in a card's title bar.

card type Specifies a card's functional capabilities. NoteCards comes with eight card types. Each card type is specalized to perform a specific of operation, e.g., Search cards perform searches on card titles. Document cards create documents.

card-type bit map The image on the left of the link icon which indicates the card type. In the image below, the card-type bit map indicates a FileBox card. See the section "Change Display Mode" in Chapter 8 for more a complete list of card-type bit maps.



card UID The number given to each card which uniquely identifies it. See Appendix A for more information.

caret A blinking arrowhead in a window, indicating where the keyboard characters will appear when typed. You change the position of the caret by moving the **mouse cursor** and pressing the left mouse button.

cascading menu Same as submenu.

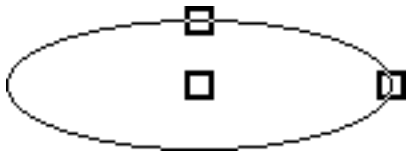
cross-file link A link which points to a destination card stored in a notefile other than the source card's notefile.

click To press and release a mouse button, the left button unless otherwise indicated.

connected directory The directory the system uses by default when you do not specify a complete path when saving, retrieving, or loading a file. Also referred to as the default directory.

connection A line drawn between two nodes in a Graph or Browser card. A connection may or may not represent a link.

control point A position that helps to determine the location and shape of a Sketch element. Each element has one or more control points. In the following sketch, the ellipse has three control points indicated by the boxes.



current selection The text in a Text card that is marked as selected in some way (underlining, highlighting, etc.) and has the caret at one end of the selection, e.g.

dashing The property of a line that causes it to be dashed., e.g.

Links in browser cards may be dashed.

default An action taken (or value specified) unless another is specified by the user.

deleted icon An icon which appears in a source card when the destination card a link points to is deleted. Shown below.

destination card The card which is opened (or flashed if already open) when you click on a link icon.

directory The name of a group of related files. In UNIX a directory is a file containing a list of other directories and files.

extension A string appended to a file name, indicating the type of file. The extension is separated from the file name with a period; the version number is separated from the extension by a semicolon. Sketch files have the extension ".sketch" followed by a version number. See **file name**.

file name A character string used to refer to a file stored on disk. A full file name is composed of the file name followed by the file extension, followed by the version number, e.g. "Solar System. NOTEFILE;2." Here the file name is "Solar System," the extension is "NOTEFILE," and the version number is "2." The file name and extension are separated by a period and the extension and version number are separated by a colon. See **extension** and **version number**.

file server A computer that provides file storage and retrieval service for users on the network.

font A collection of characters in one size and style of type, e.g., 10-point Modern Italic.

font family A complete assortment of letters, numbers, punctuation marks, etc., of a given design, such as Modern or Classic.

font size The distance from the top of the highest character in a font to the bottom of the lowest.

forward links To-links and global to-links. This term is used when setting Browser and LinkIndex card parameters.

from-link A link corresponding to every to-link and global to-link. From-links point from the destination card to the source card. All from-links are global.

global links Links whose start point is the card itself and not some point in the card's contents.

global to-links Links whose start point is a card itself and whose end point is another card. These links are not represented by link icons in a card's body. To see and access these link icons, you must choose the Show Links option from a card's left button title bar menu.

graph-based cards Cards based on the graph editor. Graph and Browser cards are graph-based cards.

hard copy The physical copy (on paper) of an on-screen document.

host Any machine on a network. Often used to refer specifically to a machine that provides a network service, such as filing.

icon An on-screen pictorial image which can represent some other object, such as card, window, or link.

image object A graphic image, such as a Sketch drawing, graph, bit map, link icon, horizontal rule, etc.

initialization file A file that is loaded when NoteCards is first started, to customize your environment according to your tastes and the idiosyncracies of your site.

justification The uniform spacing of words in a line, so that the line ends flush with both margins.

label A character string which does not represent a card in a Graph or Browser card. In the display below, "Planet" is a label.

line bar The left part of a text card where the cursor shifts from being left-pointing to right-pointing allowing you to select larger units (paragraphs and lines) of text.

link A connection between a source card and a destination card.

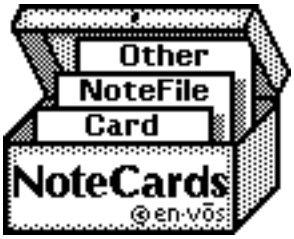
link icon The graphical representation of the link. Clicking on a link icon traverses the link it represents and opens the destination card, or flashes it if it is already open.

link type A word or phrase assigned by the user which classifies the relationship between two cards. Examples of link types include explanation, question, and answer.

local links Links whose start point is somewhere in the contents of a card. Local links contrast with global links.

menu A collection of text strings, buttons, or icons generally used to present a set of possible actions on the screen for user selection with the mouse.

MenuBox Icon The icon from which you access the NoteCards system menus. Also refered to as the MenuBox.



mouse A pointing device equipped with buttons.

mouse cursor A small image (usually an arrow) on the display screen that tracks the position of the mouse and lets you do things like reposition the caret. The cursor changes shape under certain conditions. An hourglass shape () indicates that a process is going on which may take some time to complete. A small image representing the mouse () which indicates that the system is waiting for a confirmation response before a selected process is performed.

node Refers to a card or label in a Browser graph. The graph below shows four nodes, one of which (Moons) is a label.

note cards Refers to non-FileBox cards, i.e., Text, Sketch, Graph, Browser, Search, LinkIndex, and Document cards. The generic term for cards of all types is just "card."

notefile The file produced by the NoteCards system that contains the card index and cards.

notefile index A fixed-size index of all the cards in the notefile. Its size must be explicitly changed.

network An interconnection of several computers and other devices, such as printers, that lets them communicate and share resources. Sometimes also used to refer to the medium itself, such as a coaxial cable in the case of the Ethernet. Also, "net."

number pad menu A calculator-style menu in which numbers can be entered to specify such values as line thickness.

path name The complete name of a file you want to access. Includes the name of the host file server, device, directory, subdirectories, file name, extension, and version number. For example, in UNIX a notefile path name might be `{dsk}/usr/users/kmount/nc/notefiles/demo.notefile`, where `dsk` is the name of a device, `usr` is the directory, `users`, `kmount`, `nc`, and `notefiles` are the subdirectories, and `demo.notefile` is the file name of the notefile you want to access.

persistent menu A menu that stays open until you close the menu or a window it is attached to. Also refered to as a "permanent menu" and "fixed menu."

pixel A pixel is the smallest element of a display surface (a dot on a screen) which can be independently assigned a color or intensity. A blend of the words picture and element.

pop-up menu A menu that appears when you press a mouse button and disappears when you release the button. Usually appears at the location of the cursor. The background menu is an example of a pop-up menu. See also **menu** and **background menu**.

prompt window A small window attached to the top of a card, icon, or editor window where process information is printed and where you are asked questions. See also **system prompt window**.

scroll bar The narrow window that opens when you move the cursor just outside the left or bottom edge of a window. You use the scroll bar to move through a body of information. The scroll bar frequently does not appear unless there is more information than can be displayed in the window.

sketch-based cards Cards based on the Sketch graphical editor. There is only one sketch-based card and it is the Sketch card.

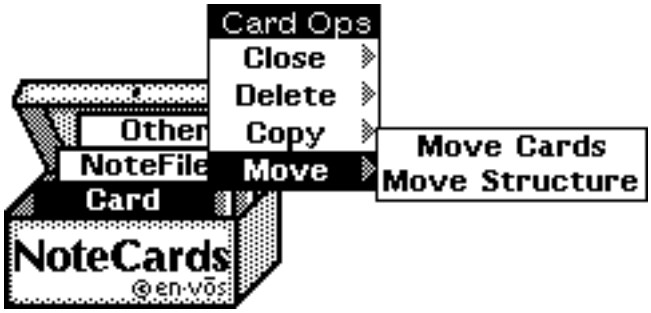
screen point The size of a single pixel on the display. A screen point is 1/72 of an inch, normally.

select This term has several meanings depending on the context of its use. When refering to menus, **select** means:

Move the mouse cursor over the menu command, press the mouse button until the command is highlighted, then release the mouse button. You also select text (see **current selection**).

source card A card containing a link icon.

submenu A menu obtained by sliding to the right off an option on another menu. The existence of a submenus is indicated by a gray arrow head in the extreme right margin of a menu option. In the example below **Close**, **Delete, Copy**, and **Move** have submenus. In the case of **Move**, the user has slid the mouse off the right of the **Move** option and the **Move Cards/Move Structure** submenu has appeared.



sysout A frozen version of the Notecards environment. It contains all the information needed to initialize virtual memory when Notecards is started.

system prompt window The black window, usually at the top of the screen, used to display system information.

text-based cards Cards based on the TEdit text editor which can also contain the output from sketch-based and graph-based cards. The text-based cards are Text, FileBox, Search, LinkIndex, and Document cards.

title bar The black bar containing the window's title that appears at the top of a card or window.

to-links Links whose start point is in a card's contents and whose end point is another card. These links are represented as link icons in a card's body.

type size The distance from the top to the bottom of the characters that represent the highest and lowest points in a character set. Measured in points.

type-in process The editor process associated with a card or window which is the destination for keyboard input.

version number The number at the end of the file name that indicates when the file was created with respect to other files of the same name. The system assigns each successive file created with the same name a higher version number. The higher the version number, the newer the file.

virtual memory Working space on the local disk that can be used, with the aid of swapping programs, to emulate random access memory.

window A defined area within a display screen that can be used as a working space. Multiple windows can appear simultaneously on one display screen, and can overlay one another.

window menu The menu that appears whenever you press the right button with the cursor in the black title bar of a card or window, or at the top of a window with no title bar.

[This page intentionally left blank]

NOTECARDS PROGRAMMER'S INTERFACE

Introduction

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

The functions described below are divided into 8 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. Customizing the NoteCards Interface
8. Handy Miscellaneous Functions

Information for Prerelease Users

The two main changes to NoteCards from 1.2 are multiple open notefiles and cards stored as datatypes rather than atoms. The former change probably has a greater effect on users of the programmer's interface. Many functions now take a NoteFile argument that didn't before. In addition, many functions have been renamed or dropped altogether. Though we attempted to preserve some backward compatibility by keeping around old function names, we strongly recommend that you go over all your code and bring it up to date with this documentation. (For example, the whole notion of "Substance types" has been removed along with the Substance fns, etc.)

Other changes to watch out for include: traversal of NoteCards structures ala browsers; copying of groups of cards and their "internal" links across notefiles; new <quietFlg> args to many of the notefile access functions; undisplaying cards without uncaching; case insensitivity in NCP.TitleSearch; and registering of cards by key in the new notefile hash table. There are lots of handy new functions like NCP.ChangeCardTypeFields, NCP.CreateCardTypeStub, NCP.CollectCards, NCP.CopyCards, NCP.ListOfOpenNoteFiles, NCP.NumCardSlotsRemaining, NCP.ExpandNoteFileIndex, NCP.NoteFileMenu, NCP.NoteFileProp, NCP.WhichNoteFile, NCP.CreateLink, NCP.CardUserDataProp, NCP.DisplayedCards, NCP.RegisterCardByName, NCP.LookupCardByName, NCP.ListRegisteredCards, NCP.OpenCard, NCP.CloseCards, NCP.CacheCards, NCP.UncacheCards, NCP.DisplayCard,

NCP.UndisplayCards, NCP.AskYesOrNo, NCP.CardNeighbors, NCP.MapCardsOfType, NCP.MapLinksOfType.

1. NoteFile Creation and Access

Note that some of the following accept filename arguments, some take notefile object arguments and some can take either. In any case, if the function accepts a filename, the .notefile suffix will be attached if not already present.

(NCP.CreateNoteFile <Filename><quietFlg>)

If <Filename> is not already a notefile, then creates a notefile with name "<Filename>.NoteFile", and returns this filename which can later be passed to NCP.OpenNoteFile. If <quietFlg> is non-nil, then communicative messages are held to a minimum.

(NCP.OpenNoteFile <NoteFileOrFilename> <don'tCreateFlg> <convertw/oConfirmFlg> <quietFlg> <menuPosition> <readOnlyFlg> <Don'tCreateInterfaceFlg>)

<NoteFileOrFileName> can be either a notefile object or a file name. Opens notefile, returning resultant notefile object if successful, else nil. If <don'tCreateFlg> is non-nil, then a new file is not created if the given one doesn't exist. If <convertw/oConfirmFlg> is non-nil, then if needed, the file is converted to the most recent format without user confirmation. If <quietFlg> is non-nil, then communicative messages are held to a minimum. If <menuPosition> is non-nil, then it should be a position at which to bring up the notefile main menu icon. If <readOnlyFlg> is non-nil, then the notefile is opened for read only. If <Don'tCreateInterfaceFlg> is non-nil, then don't bring up a control panel menu for the notefile.

(NCP.CloseNoteFiles <NoteFilesOrT> <quietFlg>)

If <NoteFilesOrT> is a list (or a single notefile), then close all open notefiles on that list (or the single one). If T, then close all open notefiles. If <quietFlg> is non-nil, then communicative messages are held to a minimum.

(NCP.CheckpointNoteFiles <NoteFilesOrT> <quietFlg>)

If <NoteFilesOrT> is a list (or a single notefile), then checkpoint all open notefiles on that list (or the single one). If T, then checkpoint all open notefiles. 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. <quietFlg> non-nil will keep messages to a minimum.

(NCP.AbortNoteFiles <NoteFilesOrT> <Don'tConfirmFlg> <quietFlg>)

If <NoteFilesOrT> is a list (or a single notefile), then abort all open notefiles on that list (or the single one). If T, then abort all open notefiles.Aborting a notefile closes it and scraps all work since last checkpoint or successful close. <quietFlg> non-nil will keep messages to a minimum. <Don'tConfirmFlg> non-nil will prevent the asking of user for confirmation before throwing away work since last checkpoint.

(NCP.CompactNoteFile <fromNoteFileOrFilename> <toFilename> <inPlaceFlg>)

<fromNoteFileOrFilename> can be either a notefile or a file name. It is compacted, usually recovering space. If <inPlaceFlg> is non-nil, then <toFilename> is ignored and the compaction is in place. Will close <fromNoteFileOrFilename> if it's currently open.

(NCP.CompactNoteFileInPlace <NoteFileOrFilename>)

Compacts <NoteFile> in place, replacing the old version. Equivalent to (NCP.CompactNoteFile <NoteFile> NIL T).

(NCP.NumCardSlotsRemaining <NoteFile>)

Returns the total number of cards that can be created in <NoteFile> before its index must be expanded. <NoteFile> should be an open notefile.

(NCP.ExpandNoteFileIndex <NoteFile> <numNewCardSlots> <QuietFlg>)

Causes <NoteFile> (which should be a valid notefile) to be checkpointed and then have its index expanded in place to make room for <num NewCardSlots> new cards. If <NoteFile> is currently closed, then it is opened, expanded and then closed immediately.

(NCP.RepairNoteFile <NoteFileOrFilename> <readSubstancesFlg>)

Runs the Inspect&Repair facility on <NoteFileOrFilename>. It must *not* be currently open. If <readSubstancesFlg> is non-nil, then the inspector will check the contents of card substances rather than the simpler check of substance length. This means the first phase of the inspector takes MUCH longer so use with care. See the documentation on the Inspect&Repair facility for more information.

(NCP.DeleteNoteFile <NoteFileOrFilename> <Don'tConfirmFlg> <quietFlg>)

Deletes the <NoteFileOrFilename> notefile. Must not be open. <quietFlg> non-nil will keep messages to a minimum. <Don'tConfirmFlg> non-nil will prevent the asking of user for confirmation before deleting.

(NCP.NoteFileFromFileName <Filename>)

Returns the notefile corresponding to Filename if any, else nil.

(NCP.FileNameFromNoteFile <NoteFile>)

Returns the full name of the given notefile.

(NCP.NoteFileMenu <NoteFile>)

Returns the main menu for <NoteFile>.

(NCP.OpenNoteFileP <NoteFile>)

Returns non-NIL if <NoteFile> is a currently open notefile.

(NCP.ValidNoteFileP <NoteFile>)

Returns non-NIL if the notefile is a valid notefile.

(NCP.NoteFileClosingP <DontCheckForAbortFlg >)

Returns non-NIL if it has been called from under a close notefile operation. If DontCheckForAbortFlg is NIL, aborting the notefile counts as closing it. If DontCheckForAbortFlg is non-NIL, aborting the notefile is considered a different operation, and the function will return NIL.

(NCP.ListOfOpenNoteFiles)

Returns a list of all currently open notefiles.

(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.

2. Creating and Accessing NoteCard Types

These functions give the user access to the NoteCard user-defined types facility. (In the functions below, card type arguments should be atoms.) For example, one of the simplest card types commonly created is what we call a "form" card. At card create time, it adds certain predefined text to the card's contents. Otherwise, form cards behave just like Text cards. Figure 1 shows the code needed to implement a sample form card type. For a full explanation of this facility, see the NoteCards Types Mechanism documentation.

FooForm.AddFooFormType

```
(LAMBDA NIL                                          (* rht; "10-Jul-86 11:41")

        (* * This creates the FooForm card type.)

  (NCP.CreateCardType (QUOTE FooForm)
        (QUOTE Text)
        (QUOTE ((MakeFn FooForm.MakeFn)))
        (QUOTE ((DisplayedInMenuFlg T)))))
```

FooForm.MakeFn

```
(LAMBDA (Card Title NoDisplayFlg)                    (* rht; "10-Jul-86 11:55")

        (* * The MakeFn for the FooForm card type,
        Note the call to the Text card's makefn.)

  (PROG1 (NCP.ApplySuperTypeFn MakeFn Card Title NoDisplayFlg)
         (NCP.CardAddText Card (CONCAT
                        "Some initial text for the FooForm card."
                                  (CHARACTER 13)
                                  ">>field1<<"
                                  (CHARACTER 13)
                                  "etc."))))
```

Figure 1: Sample card type definition

(NCP.CardTypes)

Returns list of all currently defined NoteCard types.

(NCP.CardTypeP <type>)

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

(NCP.CardTypeFns)

Returns a list of the valid Fns fields for NoteCard types. Currently, these include: MakeFn, EditFn, QuitFn, GetFn, PutFn, CopyFn, MarkDirtyFn, DirtyPFn, CollectLinksFn, DeleteLinksFn, UpdateLinkIconsFn, InsertLinkFn and TranslateWindowPositionFn.

[The following hooks should some day become legitimate Fns or Vars fields for NoteCard types. Currently, they are properties placed on the card type's atom.

WhenSavedFn If you need to have certain functionality happen just before a card's contents is saved to the notefile, then make a function be the value of the WhenSavedFn prop of the card type atom. It will be called whenever a card's contents is saved.

WhenDeletedFn If you need to have certain functionality happen just before a card is deleted, then make a function be the value of the WhenDeletedFn prop of the card type atom. It will be called whenever a card is deleted. If the function returns 'ABORT, then the deletion of the card will be aborted.

LinkIconLeftButtonFn One may now specify an operation other than TraverseLink to be used when left buttoning in link icons. Just put the property "LinkIconLeftButtonFn" with the value of a function on the (destination card's) card type atom. The function will get called with two args, the destination card and the window containing the link icon. (This is similar to the ExtraLinkIconMenuItems card type property consulted by the link icon middle button code.)

AttachedBitMapFn You may now supply a function to calculate the attached bitmap for a card type. This function should be the value of the AttachedBitMapFn prop of the card type atom, and it will be passed Card, ScaledHeightToMatch, and Scale. If it exists, it will be applied first in calculating the bitmap to be displayed. If it returns a bitmap, a list of heights and bitmaps will be computed and placed

on the LinkIconAttachedBitMap field of the card type. This hook may also be used to calculate the bitmap to be displayed on the right side of a link icon if it is a cross-file link and its attached bitmap is being displayed. In this case, the function should be the value of the AttachedBitMapFn prop of the card type atom 'CrossFileLink. (For computing the list of heights and bitmaps in an AttachedBitMapFn, see NCP.MakeTypeIconBitMapSet at the end of this section.)

Don'tForceFilingFlg It is possible to turn off forced filing on either the card type level or the card level. To define all cards of a type as not needing filing, you should put the property "Don'tForceFilingFlg" with the value of T on the card type atom. To define an individual card as not needing filing, use the function (NCP.MarkAsNotNeedingFiling <card>) (see Section 8: Handy Miscellaneous Functions).

NewCardPos If present, the value of this property determines where new cards of the type are brought up on the screen, assuming the RegionOrPosition argument now accepted by all card type makefns (passed to NC.DetermineDisplayRegion) is NIL. **]**

(NCP.CardTypeVars)

Returns a list of the valid Vars fields for NoteCard types. Currently, these include: SuperType, StubFlg, FullDefinitionFile, LinkDisplayMode, DefaultWidth, DefaultHeight, LinkAnchorModesSupported, DisplayedInMenuFlg, LinkIconAttachedBitMap, LeftButtonMenuItems and MiddleButtonMenuItems.

(NCP.CardTypeFnP <fn>)

(NCP.CardTypeVarP <var>)

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

(NCP.CardTypeFn <type> <fn>)

Returns the <fn> field for <type>. Note that this may be a value inherited at card type creation from <type>'s super type.

(NCP.CardTypeVar <type> <var>)

Returns the <var> field for <type>. Note that this may be a value inherited at card type creation from <type>'s super type.

(NCP.CardTypeSuper <type>)

Returns the super type of <type>. Equivalent to (NCP.CardTypeVar <type> 'SuperType).

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

Makes a new NoteCard type with name <TypeName> and super type <SuperType>. Any functions not appearing in <FnsAssocList> or vars not appearing in <VarsAssocList> will be inherited from <SuperType>.

Note that, for now, specializing the FileBox card type is very dangerous and has unknown repercussions.

(NCP.DeleteCardType <TypeName> <DeleteSubTypesFlg)

Deletes the NoteCard type with name <TypeName>. If DeleteSubTypesFlg is non-NIL, then it recursively deletes all sub-types of <TypeName>. If DeleteSubTypesFlg is NIL, then attempting to delete a type with sub-types is an error.

(NCP.CreateCardTypeStub <TypeName> <SuperType> <FullDefinitionFileName> <FnsAssocList> <VarsAssocList>)

Makes a stub for a new NoteCard type with name <TypeName> and super type <SuperType>. Some subset of the fns and vars can appear in <FnsAssocList> and <VarsAssocList>, however, the full definition will be loaded from <FullDefinitionFileName> the first time an attempt is made to access an undefined field name.

(NCP.ChangeCardTypeFields <TypeName> <FnsAssocList> <VarsAssocList>)

<TypeName> should be an existing NoteCard type. Some subset of the fns and vars should appear in <FnsAssocList> and/or <VarsAssocList>. These will be changed in the card type and then the inheritance mechanism will propagate these changes to any inheriting card types. For example, to make some existing card type FOO appears in the card type menu, do (NCP.ChangeCardTypeFields 'FOO NIL '((DisplayedInMenuFlg T))).

(NCP.ApplyCardTypeFn <CardTypeFn> <Card> <arg1> ...)

This macro applies the card type fn <CardTypeFn> (unevaluated) of the card type of <Card> to <Card> and the other args.

(NCP.ApplySuperTypeFn <CardTypeFn> <Card> <arg1> ...)

This macro applies the card type fn <CardTypeFn> (unevaluated) of the super type of <Card>'s type to <Card> and the other args.

(NCP.IsSubTypeOfP <type1> <type2>)

Returns non-nil if type1 inherits directly or indirectly from type2, that is, type2 can be found somewhere up the SuperType chain from type1.

(NCP.TextBasedP <cardOrType>)

(NCP.SketchBasedP <cardOrType>)

(NCP.GraphBasedP <cardOrType>)

If <cardOrType> is a card then we use its type. Returns non-nil if that type inherits directly or indirectly from Text, Sketch or Graph respectively.

(NCP.AutoLoadCardType <TypeName>)

<TypeName> is currently undefined. This asks NoteCards to look around for the file containing it and load it. It searches NOTECARDSDIRECTORIES for a file named NC<TypeName>TYPE.

(NCP.LinkIconAttachedBitMap <TypeName> <Size>)

Returns the link icon attached bitmap of size Size for card type TypeName. Default size is determined by the global variable NCP.DefaultLinkIconAttachedBitMapSize, whose initial value is 17.

(NCP.MakeTypeIconBitMapSet <Bitmap> <Heights>)

Returns a list of heights and bitmaps to be used in determining the appropriately sized bitmap for a given link icon. (Typically used when supplying an AttachedBitmapFn - see above.) <Bitmap> will be scaled to all of the heights in <Heights>, which defaults to NC.DefaultLinkIconAttachedBitMapHeights if NIL.

3. Creating NoteCards and FileBoxes

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

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

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>. Note that actually a top level copy of <props> is used (i.e. (APPEND <props>)). <InterestedWindow> is a window used to attach a prompt window for messages. <RegionOrPosition> is used to position and/or shape the new card.

(NCP.CreateTextCard <NoteFile> <title> <nodisplayflg> <props> <parentfileboxes> <InterestedWindow> <RegionOrPosition>)

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. <InterestedWindow> is a window used to attach a prompt window for messages. <RegionOrPosition> is used to position and/or shape the new card.

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

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.CreateNoteCard. <InterestedWindow> is a window used to attach a prompt window for messages. <RegionOrPosition> is used to position and/or shape the new card.

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

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 (<backarrow>ALL) in which case browsing will be done on all links in either the forward or reverse direction. If both ALL and _ALL (<backarrow>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 (layed out as a forest, sacrificing screen space for speed), COMPACT (layed out as a forest, using minimal screen space), LATTICE (layed out as a directed acyclic graph, the default), *GRAPH* (layed 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 layed on its side or up and down. The third element, if present, should be the atom REVERSE. This indicates that horizontal graphs should be layed out from right to left instead of left to right and that vertical graphs should be layed out from bottom to top rather than vice versa.

If all of LINKTYPES, DEPTH, or FORMAT are omitted, the user is asked to choose them from a stylesheet. If one or more is specified, even as being NIL, the user is not prompted for them. <InterestedWindow> is a window used to attach a prompt window for messages. <RegionOrPosition> is used to position and/or shape the new card.

(NCP.CreateSketchCard <NoteFile> <title> <nodisplayflg> <props> <parentfileboxes> <InterestedWindow><RegionOrPosition>)

Creates and returns an initially empty sketch/map card having given title, props, and parents. <InterestedWindow> is a window used to attach a prompt window for messages. <RegionOrPosition> is used to position and/or shape the new card.

(NCP.CreateGraphCard <NoteFile> <title> <nodisplayflg> <props> <parentfileboxes> <InterestedWindow><RegionOrPosition>)

Creates and returns an initially empty graph card having given title, props, and parents. <InterestedWindow> is a window used to attach a prompt window for messages. <RegionOrPosition> is used to position and/or shape the new card.

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

Creates and returns a Document card starting from <rootcard>. The user may specify new values for the set of parameters for making a document with <parametersProplist>. For example, a value of '(TitlesFromNoteCards Bold ExpandEmbeddedLinks ALL)

for <parametersProplist> would cause these values for the parameters TitlesFromNoteCards and ExpandEmbeddedLinks to be used, and the current defaults for the other parameters will be used. If no <parametersProplist> is provided, the user will be prompted for the parameters with a stylesheet. As usual, the resulting card will have the given props and parents. <InterestedWindow> is a window used to attach a prompt window for messages. <RegionOrPosition> is used to position and/or shape the new card.

(NCP.MakeLinkIndex <NoteFile> <linktypes> <backpointersP> <nodisplayflg> <props> <parentfileboxes> <InterestedWIndow> <RegionOrPosition>)

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 litatoms ALL and/or _ALL (<backarrow>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. <InterestedWindow> is a window used to attach a prompt window for messages. <RegionOrPosition> is used to position and/or shape the new card.

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 impact on most of 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.

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.

Cards can be displayed, cached or closed. A cached card has its information cached in memory thus saving time (to access) at the expense of space. All cards displayed on the screen are also cached. In almost all cases, users will need only use NCP.OpenCard and NCP.CloseCards. NCP.OpenCard does both caching and displaying while NCP.CloseCards does both undisplaying and uncaching, if necessary.

(NCP.OpenCard <card> <region/position> <typeSpecificArgs>)

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. <typeSpecificArgs>, if any, will be passed to the card type's EditFn.

(NCP.CloseCards <cardOrListOfCards> <quietFlg>)

Undisplay and uncache all the cards in <cardOrListOfCards>. <quietFlg> non-nil cuts down on messages.

(NCP.DisplayCard <card> <region/position> <typeSpecificArgs>)

If <card> was cached but not displayed, then bring it up on the screen. The <region/position> argument is as in NCP.OpenCard. <typeSpecificArgs>, if any, will be passed to the card type's EditFn.

(NCP.UndisplayCards <cardOrListOfCards> <quietFlg> <writeChangesFlg>)

If any of <cardOrListOfCards> were displayed, then undisplay them. <quietFlg> non-nil cuts down on messages. Normally, changes are not written to the notefile when a card is undisplayed, but only when it is uncached. <writeChangesFlg> non-nil makes changes be written through to the notefile.

(NCP.CacheCards <cardOrListOfCards>)

If any of <cardOrListOfCards> are not currently cached, then cache them.

(NCP.UncacheCards <cardOrListOfCards> <quietFlg>)

If any of <cardOrListOfCards> are cached but not displayed, then uncache them. <quietFlg> non-nil cuts down on messages.

(NCP.CardDisplayedP <card>)

Returns non-nil if given card or box is currently displayed in a window.

(NCP.CardCachedP <card>)

Returns non-nil if given card's information is currently cached.

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.CacheCards (and then uncaching with NCP.CloseCards).

(NCP.CardType <card>)

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

(NCP.ValidCardP <card>)

Returns <card> if <card> exists (hasn't been deleted), otherwise returns NIL. (This is currently a synonym for NCP.CardType.)

(NCP.SameCardP <card1> <card2>)

Returns non-nil if <card1> is the same card as <card2>. Error if either arg is not a valid card.

(NCP.NewCardP <card>)

Returns non-nil if <card> is new; ie. not yet saved in the notefile. Error if the arg is not a valid card.

(NCP.CardBeingDeletedP <card>)

Returns non-nil if <card> is in the process of being deleted; for use with user-defined card types. Error if the arg is not a valid card.

(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 a card object 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 a card object or a list.

(NCP.CardParents <card> <FollowCrossFileLinksFlg>)

Returns list of fileboxes in which <card> is filed. <FollowCrossFileLinksFlg>, if non-nil, causes cross-file links to be followed to recover fileboxes in remote notefiles in which <card> is filed.

(NCP.FileBoxChildren <filebox> <FollowCrossFileLinksFlg>)

Returns list of children of <filebox> in the order in which they appear in the box's textstream. <FollowCrossFileLinksFlg> is as in NCP.CardParents.

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

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 <NoteFile> should not be nil and this returns all links in <NoteFile> having a label in <labels>. If all three args are nil, then this is a slow synomym for (NCP.AllLinks <NoteFile>).

(NCP.GetCrossFileLinkDestCard <CrossFileLinkCard> <InterestedWindow> <Don'tOpenDestNoteFileFlg>)

For a given <CrossFileLinkCard>, tries to follow the link to a remote card in the destination notefile and returns that card if found. If <Don'tOpenDestNoteFileFlg> is non-nil, then destination notefile must be open. <InterestedWindow> is an optional window argument whose prompt window is used for messages.

Cross-file link cards are "hidden" cards that serve as placeholders for true cross-file links. That is, a (two-way) cross-file link from card A to card B (in different notefiles) actually consists of 2 links and 2 cross-file link cards. In the source notefile, card A is linked to a cross-file link card AA. AA "knows" about the notefile that contains B and B's UID. Similarly, the destination notefile contains a cross-file link card BB which is linked to card B. Again, the substance of BB contains a filename "hint" for the source notefile as well as the

UID for A. One-way cross-file links only contain cross-file link cards in the source notefile. The destination card contains no record that it's been linked to.

(NCP.CardNeighbors <cards> <linkTypes> <FollowCrossFileLinksFlg>)

Return a list of cards each of which is one link away from some card in <cards>. Only links having types in <linkTypes> are considered. Both <cards> or <linkTypes> can be either nil, card objects (or link type atoms) or a list. Null <cards> means that all cards linked to by anyone are returned. <linkTypes> can include link types prefixed with the character '_' (<backarrow>). This means to follow links of that type in the reverse direction. <linkTypes> can also include either or both of the atoms ANY and _ANY (<backarrow>ANY). The former means to include any cards one link away in the forward direction, while the latter causes inclusion of any cards one link away in the backwards direction. <linkTypes>=NIL is equivalent to <linkTypes>='ANY. If <FollowCrossFileLinksFlg> is non-nil, then any links that cross notefile boundaries are followed to their remote destination cards. Otherwise, cross-file links are ignored.

(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.CardRegion <card> [<newRegion>])

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. If <newRegion> is provided, then the saved region of the card is changed. If the card is currently displayed, then it is reshaped to the new region.

(NCP.CardSubstance <card> [<newSubstance>])

Returns the substance of <card>. For example, returns a textstream in the case that the type of <card> is built on the TEdit text editor. In general, this is what the PutFn of a card type writes down to the notefile and what the GetFn reads in. If a <newSubstance> argument is present, then the substance of the card is replaced and NCP.MarkCardDirty is called.

(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 <cardOrListOfCards>)

Deletes the given cards and fileboxes from their notefile, or deletes just the one if <cards> is a single card object.

(NCP.CardNoteFile <card>)

Returns <card>'s NoteFile.

(NCP.CardWindow <card>)

Returns <card>'s window if <card> is currently displayed somewhere on the screen. (The function NCP.WindowFromCard is an alias for NCP.CardWindow.)

(NCP.CardFromWindow <window>)

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

(NCP.CardFromTextStream <TextStream>)

Returns the card associated with <TextStream>, or NIL if <TextStream> doesn't belong to some text card.

(NCP.FileBoxP <card>)

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

(NCP.AllCards <NoteFile>)

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

(NCP.CardsOfTypes <CardsOrNoteFile> <Types>)

Returns a list of all cards of the given type (or types) for the given notefile (or list of cards).

(NCP.AllBoxes <NoteFile>)

Returns a list of all fileboxes in the given notefile.

(NCP.MapCards <NoteFile><fn> <collectResultsPredicate>)

Maps down the set of all cards in the current notefile, applying <fn> to each. If <collectResultsPredicate> is non-nil, then for those cards satisfying the predicate, the values of <fn> applied to them are collected.

(NCP.MapCardsOfType <types> <NoteFile> <fn> <collectResultsPredicate>)

This is similar to NCP.MapCards, but only looks at cards whose type appears on <types>. <types> can be a single type or a list of types.

(NCP.ContentsFileBox <NoteFile>)

(NCP.OrphansFileBox <NoteFile>)

(NCP.ToBeFiledFileBox <NoteFile>)

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.

5. Creating and Accessing Links

Links consist of source card, destination card, link type, display mode and anchoring mode. The new function NCP.CreateLink can be used to create any sort of link. We still provide the four functions NCP.GlobalGlobalLink, NCP.LocalGlobalLink, etc. for those who have grown used to that style.

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

See documentation in the previous section.

(NCP.CreateLink <source> <destination> <linkType> <displayMode)

<source> can be either a card or a list of two elements (<sourceCard> <sourceLoc>). <sourceCard> should be a card to use as the source of the link while <sourceLoc> should be either the atom GLOBAL (in which case a global-to-global link is created) or a Loc directive as described in NCP.CardAddText above, that is, an integer or one of the atoms START, END or NIL. This creates and returns a new link with type <linkType>, connecting <sourceCard> to <destinationCard>. For text cards, Loc, if present, designates where to insert the link. If the link is local-to-global, then <displayMode> should be a valid displaymode or NIL. (See description of NCP.LinkDisplayMode for the valid values for <displayMode>.) (In the future, Locs for non-text cards will be specifiable. In the far future, we hope to allow local anchoring at the destination end of the link as well as the source.)

(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. See the description of NCP.LinkDisplayMode for the valid entries for the <displaymode> arg.

(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> <followCrossFileLinkFlg)

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. If the link is a cross-file link and if <followCrossFileLinkFlg> is non-nil, then the link will be traversed, opening the remote notefile if necessary to determine information about the source or destination card of the link.

(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 an instance of the LINKDISPLAYMODE record. Or it can be one of the litatoms Icon, Title, Label, or Both. Finally, it can be a list of three elements (<ShowTitleFlg> <ShowLinkTypeFlg> <AttachBitmapFlg>). Each element 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.CoerceToLinkDisplayMode <thing>)

Returns a LINKDISPLAYMODE record. Thing can be a LINKDISPLAYMODE record, cardtype, card, link, atom, or list. If thing is a LINKDISPLAYMODE record, that record is returned. If thing is a cardtype, the default LinkDisplayMode for that cardtype is returned. If thing is a card, the LinkDisplayMode of that card is returned. If thing is a link, the LinkDisplayMode of that link is returned. If thing is an atom or a list, then the corresponding LinkDisplayMode, as specified under **NCP.LinkDisplayMode** (see above), is returned.

(NCP.LinkType<link> [<newLinkType>])

Returns old linktype of <link>. If <newLinkType> is present, set <link>'s type to <newLinkType>.

(NCP.LinkSource <link>)

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

(NCP.LinkDestination <link>)

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

(NCP.DeleteLinks <links>)

Removes all links in <links> (or just one if <links> is a single link object.

(**NCP.ValidLinkP <link>)**

Error in IMAGEOBJ GETFN: HRULE.GETFN1 Returns non-nil if <link> is a link in the current notefile.

(**NCP.SameLinkP <link1> <link2>)**

Error in IMAGEOBJ GETFN: HRULE.GETFN1 Returns non-nil if <link1> is the same link as <link2>. Error if either arg is not a valid link.

(**NCP.AllLinks <NoteFile>)**

Error in IMAGEOBJ GETFN: HRULE.GETFN1 Returns a list of all existing links in <NoteFile>. (This is equivalent to but faster than (NCP.GetLinks NIL NIL NIL <NoteFile>).)

(**NCP.MapLinks <NoteFile> <fn> <collectResultsPredicate>)**

Error in IMAGEOBJ GETFN: HRULE.GETFN1 Maps down the set of all links in the given notefile, applying <fn> to each. If <collectResultsPredicate> is non-nil, then for those links satisfying the predicate, the values of <fn> applied to them are collected.

(**NCP.MapLinksOfType <types> <NoteFile> <fn> <collectResultsPredicate>)**

Error in IMAGEOBJ GETFN: HRULE.GETFN1 This is similar to NCP.MapLinks, but only looks at links whose type appears on <types>. <types> can be a single type or a list of types.

6. Creating and Accessing Link Labels

Error in IMAGEOBJ GETFN: HRULE.GETFN1 The following functions allow the user to manipulate link labels.

(**NCP.CreateLinkType <linkType> <NoteFile> <QuietFlg>)**

Error in IMAGEOBJ GETFN: HRULE.GETFN1 Creates a new link type with name <LinkType> in <NoteFile> unless there is already one defined by that name. A non-NIL <QuietFlg > suppresses the error message that is normally printed if the link type has already been defined for this notefile.

(**NCP.DeleteLinkType <linkType> <NoteFile>)**

Error in IMAGEOBJ GETFN: HRULE.GETFN1 Deletes the link type <linkType> from <NoteFile>. The link type must exist and must not be the type of any existing link, and it must not be a system-defined link type (e.g. SubBox or BrowserContents).

(**NCP.RenameLinkType <linkType> <newLinkType> <NoteFile>)**

Error in IMAGEOBJ GETFN: HRULE.GETFN1 Changes any links in <NoteFile> having link type <linkType> to have type <newLinkType>. <linkType> must exist and neither <linkType> nor <newLinkType> should be a system-defined type.

(**NCP.LinkTypes <NoteFile>)**

Error in IMAGEOBJ GETFN: HRULE.GETFN1 Returns a list of all existing link types in <NoteNFile> including system-defined ones.

(NCP.ReverseLinkTypes <NoteFile>)

Returns a list of the reverse link types for every link type in <NoteFile>. Thus, whereas SubBox would appear in the list returned by NCP.LinkTypes, _SubBox (<backarrow>SubBox) would appear in the list returned by NCP.ReverseLinkTypes.

(NCP.UserLinkTypes <NoteFile>)

Returns a list of all existing user-defined link labels in <NoteFile>.

(NCP.SystemLinkTypeP <LinkType>)

Returns non-nil if <LinkType> is a system link type..

(NCP.ValidLinkTypeP <LinkType> <NoteFile>)

Returns non-nil if <LinkType> is a defined link type for <NoteFile>.

7. Customizing the NoteCards Interface

There are currently several areas where the user may tailor the NoteCards user interface: the NoteCards Session Icon menus, the left button menu on the title bar of a notefile's menu icon, the middle button menu of a notefile's menu icon, the ShowCards middle button menu on a notefile's menu icon, and the title bar menus of displayed cards and card types.

The Session Icon: The menus associated with this icon may be modified using the following functions:

(NCP.AddSessionIconMenuItem <MenuName> <Item>)

Adds the menu item <Item> to the session icon menu specified by <MenuName> (one of 'Card, 'NoteFile, or 'Other). <Item> should be a complete menu item and may contain subitems in the standard format. For the Card and Other menus, the item can be a normal menu item. For the NoteFile menu, it must be slightly different. Instead of providing an expression to be evaluated as the second part of the menu item (and of any subitems), you should provide the name of a function of two arguments: NoteFile and Window. This function will be applied to these arguments when the menu item is selected.

Returns the item added if successful, NIL otherwise.

Example:
(NCP.AddSessionIconMenuItem ' NoteFile '(Foo% Function NC.DoFoo "Performs the function Foo on this notefile"))

(NCP.RemoveSessionIconMenuItem <MenuName> <ItemName>)

Removes the menu item named <ItemName> from the session icon menu specified by <MenuName> (one of 'Card, 'NoteFile, or 'Other). <ItemName> should be only the name of the menu item.

Returns the full item removed if successful, NIL otherwise

Example:
(NCP.RemoveSessionIconMenuItem ' NoteFile 'Foo% Function)

(NCP.RestoreSessionIconMenu <MenuName>)

Restores the menu specified by <MenuName> (one of 'Card, 'NoteFile, or 'Other) to its initial state. If <MenuName> is NIL, all three menus will be restored.

(NCP.SessionIconWindow)

Returns the session icon window.

(NCP.BringUpSessionIcon <IconPosition>)

Brings up the NoteCards icon at IconPosition. If no IconPosition is given and the icon is already on the screen, it will be flashed. If it is not on the screen, the user will be prompted to place it.

The left button menu on the title bar of a notefile's menu icons: The user may modify this menu using the following functions.

(NCP.AddNoteFileIconMenuItem <Item> <OpenOrClosedOrBoth>)

Adds the menu item <Item> to the NoteFile icon menu. <Item> should be a complete menu item and may contain subitems.. The second part of the menu item (and of any subitems) should be a function of two arguments: NoteFile and Window. Window will be the icon window for the notefile. This function will be applied to these arguments when this menu item is selected. <OpenOrClosedOrBoth> specifies for which type of notefile the new operation will be valid. If <OpenOrClosedOrBoth> is 'Open, the item will appear as an operation for open notefiles. If <OpenOrClosedOrBoth> is 'Closed, the item will appear as an operation for closed notefiles. If <OpenOrClosedOrBoth> is 'Both, the item will be available for both open and closed notefiles.

Returns the full item added if successful, NIL otherwise.

Example:
(NCP.AddNoteFileIconMenuItem '(Foo% Function NC.DoFoo "Performs the function Foo on this notefile") 'Open)

(NCP.RemoveNoteFileIconMenuItem <ItemName>)

Removes the menu item named <ItemName> from the NoteFile icon menu.

Returns the full item removed if successful, NIL otherwise.

Example:
(NCP.RemoveNoteFileIconMenuItem 'Foo% Function)

(NCP.RestoreNoteFileIconMenu)

Restores the NoteFile Icon menu to its initial state.

The middle button menu of a notefile's menu icon: It is now possible for users to add menu items to this menu using the following functions.

(NCP.AddNoteFileIconMiddleButtonItems <Notefile><MenuItems>)

Adds list of menu items <MenuItems> to the middle button menu of the main menu icon corresponding to <Notefile>. These menu items will remain on the menu until the next time the notefile is closed. The second item of each menu item should be an atom that is the name of a function to be called when selected.

(NCP.AddDefaulltNoteFileIconMiddleButtonItems <MenuItems>)

Adds list of menu items <MenuItems> to the middle button menu of the main menu icons for all notefiles. These menu items will remain on the menus for the life of the session. The second item of each menu item should be an atom that is the name of a function to be called when selected.

The ShowCards middle button menu on the notefile icon: Users can indicate that certain cards should be added to or deleted from this menu using the following functions.

(NCP.AddSpecialCard <Card>)

Adds <Card> to the list of special cards appearing in the middle button menu on the ShowCards option of the notefile icon (for the notefile containing <Card>).

(NCP.RemoveSpecialCard <Card>)

Removes <Card> from the list of special cards appearing in the middle button menu on the ShowCards option of the notefile icon (for the notefile containing <Card>).

The title bar menus of cards and card types: Users can add menu items to either a specific window containing a card, or to the default menu of a card type.

(NCP.AddTitleBarMenuItemsToWindow <Win> <Button> <NewMenuItems><TopOrBottom>)

Adds the menu items <NewMenuItems> to the title bar menu of <Win>. <Button> should be one of 'Left or 'Middle, corresponding to either the LeftButtonTitleBarMenu or the MiddleButtonTitleBarMenu. <Win> should be the window of a visible notecard. <TopOrBottom> should be one of 'Top or 'Bottom, indicating where in the menu the new items should appear.

(NCP.AddTitleBarMenuItemsToType <Type> <Button> <NewMenuItems> <TopOrBottom>)

Adds the menu items <NewMenuItems> to the title bar menu of all cards of type <Type>. <Button> should be one of 'Left or 'Middle, corresponding to either the LeftButtonTitleBarMenu or the MiddleButtonTitleBarMenu. <TopOrBottom> should be one of 'Top or 'Bottom, indicating where in the menu the new items should appear.

Menu of notefiles: the user may use the list and menu of noticed notefiles maintained by NoteCards.

NCP.NoticedNoteFileNames

Global var containing a list of the currently available notefile names noticed by NoteCards.

(NCP.NoticedNoteFileNamesMenu <IncludeNewNoteFileFlg> <AllowedOperations> <InterestedWindow> <Operation>))

Provides user with a menu of noticed notefile names. <IncludeNewNoteFileFlg> should be non-NIL if new notefiles are allowed. <AllowedOperations> should be one of the atoms: OPEN, CLOSED or NIL for both. <Operation> should be a string or atom containing the name of the operation to be performed on the result and the word NoteFile; e.g. (QUOTE Open% NoteFile). This is used in the prompt for a new notefile name. <InterestedWindow> is the window to receive a prompt window for any messages printed.

(NCP.ForgetNoteFileName <NoteFileOrFileName>)

The notefile is removed from the menu of noticed notefiles. It will not be added to the menu again until explicitly remembered using NCP.RememberNoteFileName regardless of other operations performed on the notefile.

(NCP.RememberNoteFileName <NoteFileOrFileName>)

The notefile is added to the menu of noticed notefiles. This is only necessary if the NoteFile name has been forgotten.

NCP.GrayShade

Global var containing the shade used for shading various menu items in the interface. The default value of NCP.GrayShade is GRAYSHADE. This variable should be changed with the function NCP.SetGrayShade (see below).

(NCP.SetGrayShade <Shade>)

This function should be used to change the value of NCP.GrayShade to <Shade>. In addition to changing NCP.GrayShade, it also resets all cached menus that use NCP.GrayShade. If <Shade> is NIL, NCP.GrayShade will not be changed, but the menus will still be reset. Returns the new value of NCP.GrayShade.

8. Handy Miscellaneous Functions

(NCP.BringUpNoteCardsIcon <IconPosition>)

Brings up the NoteCards session icon at the position <IconPosition>. If no position is given, will prompt the user th place the icon. If the icon is already on the screen and no position is given, it will be flashed. (equivalent to (NoteCards <IconPosition>).

(NCP.TitleSearch <NoteFile> <keys> <caseSensitiveFlg>)

Returns a list of all cards in <NoteFile> having all of the <keys> (can be atom, string or list) within their titles. Normally case insensitive unless <caseSensitiveFlg> is non-nil.

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

Returns a list of all cards in <NoteFile> such that for every <propOrPair> arg, if it is atomic, then the card contains 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. (The function NCP.WC is an alias for NCP.WhichCard.)

(NCP.WhichNoteFile <x> <y>)

Works just like NCP.WhichCard, returning the notefile corresponding to the indicated window. If the window is for a card, then the card's notefile is returned, if it's for a notefile menu, then

that notefile is returned. (The function NCP.WNF is an alias for NCP.WhichNoteFile.)

(NCP.NoteFileIconWindow <NoteFile>)

Returns the main menu icon window, if any, for given <NoteFile>.

(NCP.DisplayedCards <NoteFiles > <CardTypes >)

Returns a list of all cards currently displayed on the screen that are in one of **<NoteFiles >** and are of one of the types in **<CardTypes >**. (Shrunken ones *are* included.) If **<CardTypes >** is NIL, then looks at all card types. If **<NoteFiles >** is NIL, then looks at all notefiles.

(NCP.SelectCards <instigatingCardOrWindow> <singleCardFlg> <selectionPredicate> <message> <checkForCancelFlg> <NewCardFlg>)

Returns a list of those cards selected from the screen. A menu appears in or near <instigatingCardOrWindow> displaying <message> and having buttons for DONE, UNDO, CANCEL (unless <singleCardFlg> is non-nil, in which case there is only a CANCEL button) and NEW CARD (if <NewCardFlg> is non-NIL). Card selections must satisfy <selectionPredicate> and are made by left buttoning in the title bars of the desired cards. If user hits CANCEL button, then NIL is returned unless <checkForCancelFlg> is non-nil, in which case the atom DON'T is returned.

(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.
BuildBackLinks	ToCardsBoxes, ToCards, ToBoxes, NONE.
CopyEmbeddedLinks	ALL, NONE, <listOfLinkLabels>.
ExpandEmbeddedLinks	ALL, NONE, <listOfLinkLabels>.

[See the Notecards user's manual 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>
MenuLingersAfterNoteFileClose	T or NIL
ShowNoteFileOnCards	T or NIL

NewNoteFileInitialSize	<positive number (default is 1000)>
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
UseDeletedLinkIconIndicators	T or NIL
DelTEditProcessWhenShrinking	T or NIL
ExtraTEditProps	a prop list
EnableBravoToTEditConversion	T or NIL
IncludeCardObjectInShowInfo	T or NIL
LinkDashingInBrowsers	T or NIL
ArrowHeadsInBrowsers	one of the litatoms {AtEndpoint, AtMidpoint, None}
SpecialBrowserSpecs	T or NIL
DefaultFont	a font
LinkIconFont	a font
MenuFont	a font
NoteFileIndicatorFont	a font

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

(NCP.CoerceToInterestedWindow <WinOrCardOrNoteFile>)

Coerces a window, card or notefile into a window which can have a prompt window.

(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> <WaitMsecs>)

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. The prompt window will not be cleared before the specified wait has expired.

(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.AskYesOrNo <Msg> <prompt> <FirstTry> <ClearFirstFlg> <MainWindow> <DontCloseAtEndFlg> <DontClearAtEndFlg>)

This function can be used to ask yes/no questions of the user in a window's prompt window. The fields are as in NCP.AskUser except that PROMPTFORWORD is always used so there is no Flg for that.

(NCP.CardDates <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.

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

Returns the region that <card> would occupy if displayed. If <region/position> is NIL, then asks the user to position a ghost region to get the position. Also checks previous size of card or default heights and widths for the card type to find the size. If BringUpCardAtPreviousPos is on, then won't require positioning of ghost region even if <region/position> is NIL.

(NCP.SetUpTitleBar <CardWindow> <CardType>)

This function is usually called from the MakeFn for those card types that inherit from a card type that doesn't display in a window, like the List or NoteCard type. It creates and installs left and middle menus using the menu items found in <CardType>'s definition. It also installs a default button event fn on the window.

(NCP.LinkFromLinkIcon <linkIcon>)

If <linkIcon> is an image object for a link icon, then returns the associated link. Note that link icons can be obtained by calling the card type's CollectLinksFn (see the card types mechanism documentation).

(NCP.MakeLinkIcon <link>)

If <link> is a valid link, then creates and returns a new link icon image object containing it.

(NCP.MarkCardDirty <card> <resetFlg>)

Mark <card>'s substance as dirty thus forcing it to be written down at the next save. If <resetFlg> is non-nil, then *unmark* <card> as dirty.

(NCP.MarkAsNotNeedingFiling <card>)

Mark <card> as not needing filing. This is done by setting the card's Don'tRequireFilingFlg prop to T.

(NCP.CoerceToCard <cardIdentifier>)

Return the card object (if any) associated with <cardIdentifier> which can currently be any of: a card object, window or text stream.

(NCP.CollectCards <RootCards> <LinkTypes> <MaxDepth> <FollowCrossFileLinksFlg>)

Starting from <RootCards> and following links having types in <LinkTypes> to a maximum depth of <MaxDepth>, collect and return a list of all cards encountered. A value of NIL for <MaxDepth> causes search to be "infinitely" deep. <LinkTypes> and <FollowCrossFileLinksFlg> are as described in NCP.CardNeighbors.

(NCP.CopyCards <Cards> <DestNoteFileOrFileBox> <RootCards> <QuietFlg> <CopyExternalToLinksMode> <InterestedWindow>)

This copies all cards in <Cards> along with all links among them. If <CopyExternalToLinksMode> is 'COPY, external to-links will also be copied; if it is 'DON'TCOPY, external to-links will not be copied; and if it is NIL, the user will be asked if he/she wishes to copy these links. <DestNoteFileOrFileBox> designates a filebox in which to file the card copies. (If <DestNoteFileOrFileBox> is a notefile, then its Contents box is used.) <RootCards> should be a subset of Cards or NIL. If <RootCards> is NIL, then all the card copies are filed in the destination box, otherwise just those appearing in <RootCards>. If <RootCards> is the atom NONE, then none of the new cards will be filed in the destination filebox. <QuietFlg> cuts out the messages. Note that currently, <Cards> must all live in the same notefile, but this *can* be a different notefile from the destination notefile. <InterestedWindow> is a window used to attach a prompt window for messages.

(NCP.MoveCards <Cards> <DestNoteFileOrFileBox> <RootCards> <QuietFlg> <CopyExternalToLinksMode><InterestedWindow>)

This moves all cards in <Cards> along with all links among them. If <CopyExternalToLinksMode> is 'COPY, external to-links will also be copied; if it is 'DON'TCOPY, external to-links will not be copied; and if it is NIL, the user will be asked if he/she wishes to copy these links. <DestNoteFileOrFileBox> designates a filebox in which to file the card copies. (If <DestNoteFileOrFileBox> is a notefile, then its Contents box is used.) <RootCards> should be a subset of Cards or NIL. If <RootCards> is NIL, then all the card copies are filed in the destination box, otherwise just those appearing in <RootCards>. If <RootCards> is the atom NONE, then none of the new cards will be filed in the destination filebox. <QuietFlg> cuts out the messages. Note that currently, <Cards> must all live in the same notefile, but this *can* be a different notefile from the

destination notefile. <InterestedWindow> is a window used to attach a prompt window for messages.

The following functions allow users to register cards by name for fast access. This is normally done using the card's notefile's system registry. Thus this registering is preserved over notefile closing.

(NCP.RegisterCardByName <Name> <Card> <RegistryCard>)

Stores <Card> in <RegistryCard> hashed under the name <Name>. If <RegistryCard> is nil, then use <Card>'s notefile's system registry.

(NCP.LookupCardByName <Name><NoteFIleOrRegistryCard>)

If <NoteFileOrRegistryCard> is a registry card, then recovers the card that was hashed under <Name>. If <NoteFileOrRegistryCard> is a notefile, then use its system registry.

(NCP.UnregisterName <Name><NoteFIleOrRegistryCard>)

If <NoteFileOrRegistryCard> is a registry card, then smashes whatever was stored under <Name>. If <NoteFileOrRegistryCard> is a notefile, then use its system registry.

(NCP.ListRegisteredCards <NoteFIleOrRegistryCard> <IncludeKeysFlg>)

If <NoteFileOrRegistryCard> is a registry card, then returns the list of cards hashed in it. If <NoteFileOrRegistryCard> is a notefile, then use its system registry. If <IncludeKeysFlg> is non-nil, then return list of cons pairs with car=Name and cdr=Card.

One can associate functionality with the opening and closing of notefiles using two special List cards registered under the names OpenEventsCard and CloseEventsCard. Their substances each consist of a list of lisp s-expressions to be evaluated at notefile open and close time respectively. Thus, to cause a new expression to be evaluated at notefile open time, one cons's (or appends) the expression to the substance of OpenEventsCard (either via DEdit or NCP.CardSubstance). Note that during the evaluation of the expressions, the atom NoteFile is bound to the notefile being opened or closed. When a notefile is closed, the expressions in the CloseEventsCard are evaluated both just before the notefile is closed, and after it is finished being closed. In this case, the atom When is bound to either 'Before or 'After, indicating when the expressions are evaluated with respect to the close operation.

(NCP.GetOpenEventsCard <NoteFIle>)

Returns the open events card for <NoteFile> creating a new one if necessary. Thus, this is basically just (NCP.LookupCardByName 'OpenEventsCard <NoteFile>), except that a new List card is created and registered under the name OpenEventsCard if none exists.

(NCP.GetCloseEventsCard <NoteFIle>)

Returns the close events card for <NoteFile> creating a new one if necessary. Thus, this is basically just (NCP.LookupCardByName 'CloseEventsCard <NoteFile>), except that a new List card is

created and registered under the name CloseEventsCard if none exists.

Sometimes, it's nice to have a place to temporarily hang your hat, so to speak. For this purpose, we provide what are called *UserProps* for both card and notefile objects. These are temporary in that they vanish when the notefile closes.

(NCP.CardUserDataProp <Card><Prop> [<NewValue>])

Returns the value under the <Prop> user data prop for <Card>. If <NewValue> is present, then value under <Card>'s <Prop> is reset. (That is, it works like WINDOWPROP or NCP.CardProp.)

(NCP.NoteFileProp <NoteFile><Prop> [<NewValue>])

Returns the value under the <Prop> user data prop for <NoteFile>. If <NewValue> is present, then value under <NoteFile>'s <Prop> is reset. (That is, it works like WINDOWPROP or NCP.CardProp.)

(NCP.NoteFileAddProp <NoteFile><Prop> [<NewValue>])

Returns the value under the <Prop> user data prop for <NoteFile>. If <NewValue> is present, then it is added to the list under <NoteFile>'s <Prop>. If <NoteFile> has no value under <Prop>, then <NewValue> is placed on that property as a list. (That is, it works like WINDOWADDPROP.)

Error breaks/messages under the programmer's interface: When the programmer's interface detects an error, it calls NCP.ReportError or NCP.ReportWarning depending on the severity of the error. Normally the former causes a break while the latter merely prints a message. However, this behavior can be changed using the global var NCP.ErrorBrkWhenFlg, a litatom whose value should be one of NIL, ALWAYS or NEVER (default is NIL).

(NCP.ReportError <Function><Message>)

Forces a break using BREAK1 unless the value of NCP.ErrorBrkWhenFlg is the litatom NEVER. If so, then simply prints a message and continues.

(NCP.ReportWarning <Function><Message>)

Prints the given warning message <Message> unless the value of NCP.ErrorBrkWhenFlg is the litatom ALWAYS. If so, then forces a break with BREAK1.

There are a few old functions preserved for backward compatibility, but we encourage users to rewrite all their code to use only functions described above. The old functions still available are: NCP.BringUpCard, NCP.ActivateCards, NCP.ActiveCardP, NCP.DeactivateCards, NCP.ValidCard, NCP.GetContentsFileBox, NCP.GetOrphansFileBox, NCP.GetToBeFiledFileBox, NCP.GetLinkSource, NCP.GetLinkDestination, NCP.CreateLinkLabel, NCP.DeleteLinkLabel, NCP.RenameLinkLabel, NCP.GetLinkLabels, NCP.GetUserLinkLabels, NCP.GetReverseLinkLabels, NCP.ValidLinkLabel, NCP.AddLeftButtonTitleBarMenuItems, NCP.AddMiddleButtonTitleBarMenuItems.

[This page intentionally left blank]

Bitmap Editor

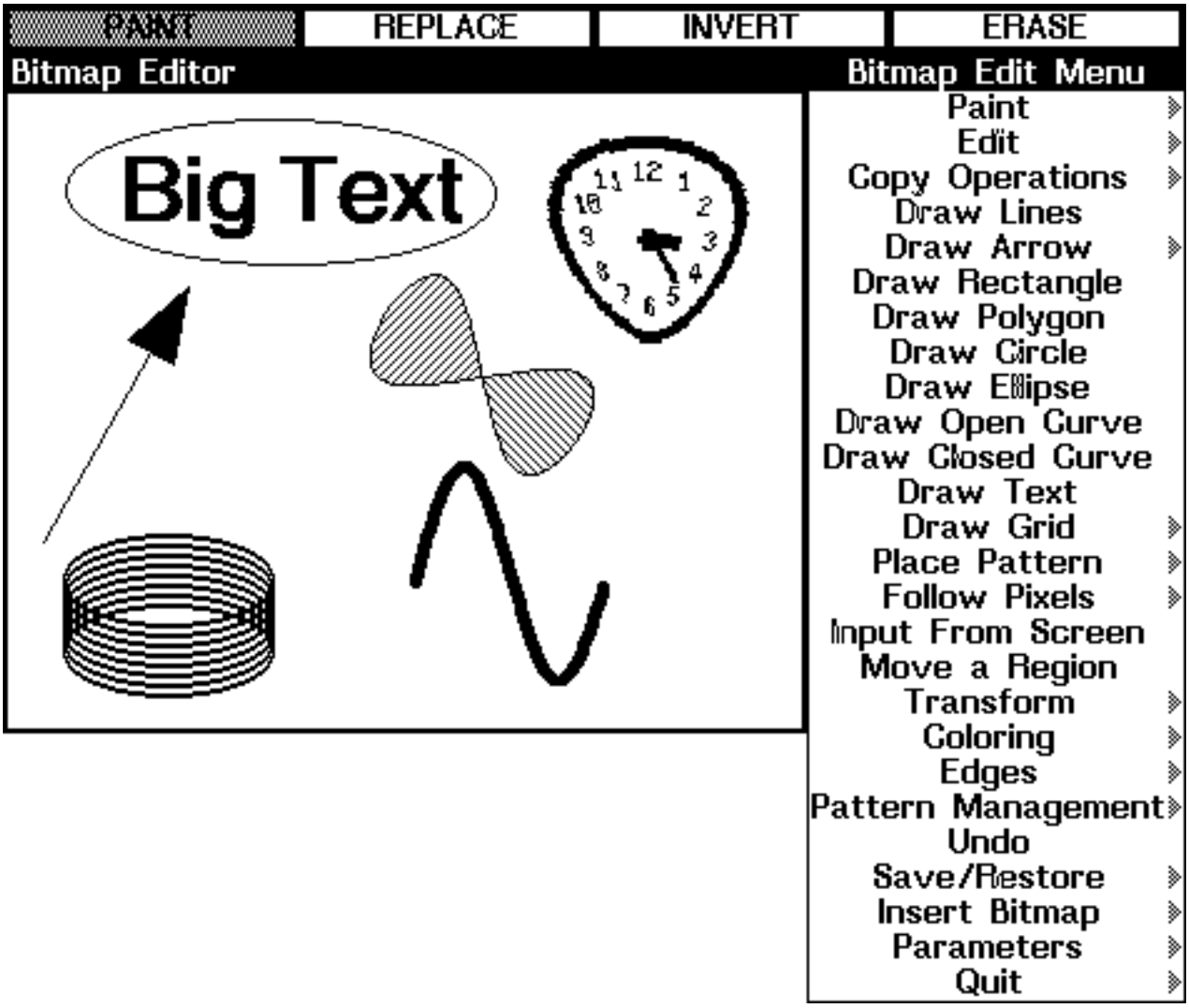
Vista Laboratory
Xerox Special Information Systems

November, 1988

The bitmap editor is invoked as follows.

(EditBitmap BitmapOrFile Window ...)

If BitmapOrFile is an existing bitmap, the user will be prompted to place a window displaying the existing bitmap. If BitmapOrFile represents a file containing a bitmap in binary form, it will be read from the file and subsequently edited via the window. If BitmapOrFile is non-NIL and Window is given, the bitmap will be edited within the existing window although it may be clipped if the bitmap is larger than the window. Finally, if both BitmapOrFile and Window are NIL, the user will be prompted to sweep out a region on the screen that will define the bitmap. The editing window is then placed as desired.



The four menu items at the top of the editor window set the primary drawing operation. Paint logically ORs the drawing bits with the bitmap bits to give the result. In Replace mode, all drawing bits replace the bitmap bits whether they are on or off. Invert does a logical exclusive OR between the drawing bits and the bitmap bits. Erase preforms a

logical AND between the inverse of the drawing bits and the destination bits. The drawing operation may also be changed via Parameter menu subitems.

A brief description of all the menu items follows.

Abort - Error out of the bitmap editor. Equivalent to a control-E.

Add Border - The bitmap is expanded by adding a border with a user specified width and texture.

Add Texture - Adds a user selectable texture to the entire bitmap.

Add Pattern - Allows the user to define a pattern (an independent bitmap) that may be placed in the primary bitmap, used for painting, etc.

Air Brush Size - Sets the size of the air brush.

Air Brush Speed - Sets the speed of the air brush. The affects the density of "spray."

Apply Operation to Copy and Original - Applies the primary bitmap operation (Paint, Replace, Invert, or Erase) to the original bitmap and an independent copy that is created and maintained by the primary editor. The copy may be separately edited.

Arrowhead - Sets the width and height of the arrowhead created by the Draw Arrow command.

Auto-Save File Name - Sets the name of the file to be used for auto-saving of the bitmap.

Auto-Save Interval - Sets the interval between automatic saving of the bitmap.

Auto-Save Status - Displays the current auto-save status, file name, and interval.

Averaging - Applies an averaging matrix to the entire bitmap. The weighted sum of each pixel and its eight neighbors is compared with a threshold. If the sum is greater than the threshold, the pixel is turned on (or left on), else it is turned off. The user may use a default averaging scheme or define her own in a Lisp executive.

Bitmap Editor Paint with Airbrush - Allows painting of the bitmap with an "air brush." The size, shape, and speed of the air brush may be changed via subitems of the Parameters menu item. The operation (Paint, Replace, Invert, or Erase) and the speed may be changed while in this mode by clicking the right mouse button and selecting the appropriate menu item.

Bitmap Editor Paint with Brush - Similar to the standard Interlsp-D paint facility. The size and shape of the brush are modified via subitems of the Parameters menu item. The operation (Paint, Replace, Invert, or Erase) may be changed while in this mode by clicking the right mouse button and selecting the appropriate menu item.

Bitmap Editor Paint with Pattern - Similar to Bitmap Editor Paint with Brush, but paints with a pattern (arbitrary bitmap) instead of a standard brush.

Brush Shape - Sets the brush and air brush shape. The available choices are round, square, horizontal, vertical, and diagonal.

Checkpoint Restore - Resets the bitmap to the state it was in just prior to the last Checkpoint Save.

Checkpoint Save - Saves the current bitmap locally. The saved bitmap is not written to a file, but merely saved as a window property. Each Checkpoint Save destroys the previous saved bitmap.

Clear - Clears the entire bitmap.

Clear All But Region - Clears the entire bitmap except for a reqion swept out by the user.

Clear Region - Clears a region swept out by the user.

Coloring - Menu subitems change coloring of the bitmap and regions.

Copy Operations - Menu subitems create and edit a copy of the bitmap.

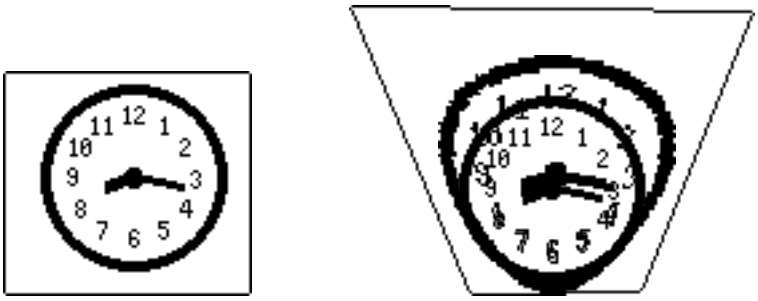
Copy/Edit Pattern - Creates a copy of an existing pattern and opens an edit window on the pattern.

Create New Copy of Bitmap - Creates a copy of the existing bitmap and maintains it separately. The copy may be edited separately.

Dashing - Turns dashing on or off and sets the style. When dashing is on, all line drawing operations will be done with the existing dashing pattern.

Delete Pattern - Allows the deletion of one of the patterns currently defined for the bitmap editor.

Distort Region - Distorts a rectangular region of the bitmap. This operation is time consuming on an 1186 which does not have hardware floating point. The operation operates as follows. A rectangular region is first swept out by the user as in the figure to the left. The vertices are then moved with the left button resulting in a display as in the middle figure. The result is shown in the rightmost figure. Note that if the user wishes to replace the region with the distorted result, the editor must be in the REPLACE mode. Otherwise, the distorted region will be PAINTed over the original region.



Draw Arrow - Draws one or more line segments with an open arrow at the end point. The size of the arrow may be changed by setting the appropriate parameters.

Draw Circle - Draws a circle in the same manner as SKETCH. The circle will be drawn with the current drawing brush size and dashing.

Draw Closed Curve - Draws a closed curve in the same manner as SKETCH. The curve will be drawn with the current drawing brush size and dashing.

Draw Ellipse - Draws an ellipse in the same manner as SKETCH. The ellipse will be drawn with the current drawing brush and dashing.

Draw Filled Arrow - Draws one or more line segments with a filled arrow at the end point. The size of the arrow may be changed by setting the appropriate parameters.

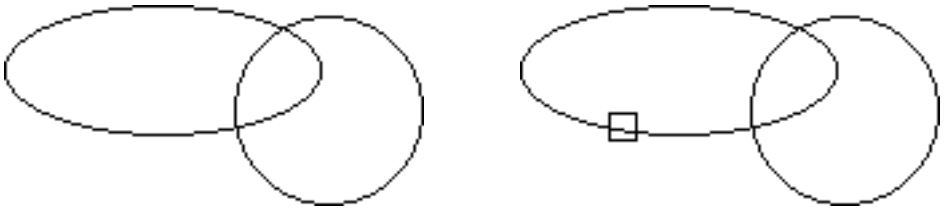
Draw Grid - Draws a grid over the entire bitmap. The width and height of the grid is specified via a number menu. The grid will be drawn with the current drawing brush size and dashing.

Draw Grid in Region - Draws a grid in a specified region. The width and height of the grid is specified via a number menu. The grid will be drawn with the current drawing brush size and dashing.

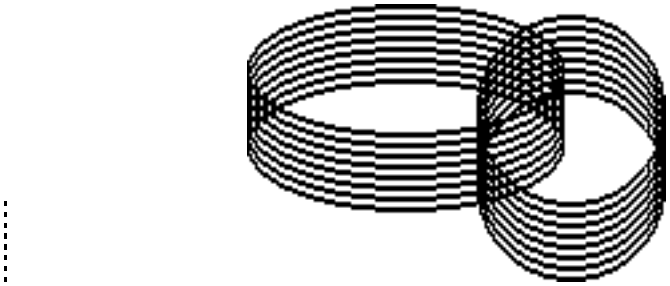
Draw Lines - Draws one or more line segments in the same manner as SKETCH. The lines will be drawn with the current drawing brush size and dashing.

Draw Open Curve - Draws a closed curve in the same manner as SKETCH. The curve will be drawn with the current drawing brush size and dashing.

Draw Over Connected Pixels from Region - Draws over all "connected" pixels starting from one or more pixels in a small region. Drawing may be done with the current drawing brush or a pattern. A pixel is connected to another pixel if it is one of the eight neighbors of the pixel. The operation is as follows. Suppose we have a display as shown to the left. We then select starting pixels by placing the box cursor as shown in the right figure.



Suppose we then select a pattern as shown to the left below. The resulting display will appear as in the figure to the right. Note that the user is requested to select the offset of the drawing brush or pattern.



Draw Over Connected Pixels from Selected Pixel - Similar to Draw Over Connected Pixels from Region. but a single pixel is selected as the starting point. The user places a separate "magnifier" window. When the cursor enters the bitmap editor, it changes to rectangular crosshairs and the surrounding 64 pixels are magnified 8 times and displayed in the magnifier window. When the crosshairs are over the selected pixel, the left button is clicked to specify the starting pixel.

Draw Polygon - Draws a polygon in the same manner as SKETCH. The polygon will be drawn with the current drawing brush size and dashing.

Draw Rectangle - Draws a rectangle in the same manner as SKETCH. The rectangle will be drawn with the current drawing brush size and dashing.

Draw Text - Places text. The font family, size, and style may be changed via Parameters menu subitems. The text is typed into a prompt window and then moved to the desired postion with the mouse.

Drawing Brush Size - Sets the size of the drawing brush.

Edges - Menu subitems trim the bitmap and add borders.

Edit - Menu subitems edit regions of the bitmap.

Edit Copy of Bitmap - Opens an edit window on the currently stored copy of the bitmap. See Create New Copy of Bitmap.

Edit Pattern - Opens an edit window on a selected pattern.

Edit Region - Opens an edit window on a selected rectangular region of the bitmap.

Erase Connected Pixels from Region - Erases all "connected" pixels starting from one or more pixels in a small region. A pixel is connected to another pixel if it is one of the eight neigbors of the other pixel.

Erase Connected Pixels from Selected Pixel - Similar to Erase Connected Pixels from Region, but a single pixel is selected as the starting point. The user places a separate "magnifier" window. When the cursor enters the bitmap editor, it changes to rectangular crosshairs and the surrounding 64 pixels are magnified 8 times and displayed in the magnifier window. When the crosshairs are over the selected pixel, the left button is clicked to specify the starting pixel.

Exact Size - Allows changing the size of the current bitmap by explicitly resetting the origin, width, and height in terms of the bitmap coordinates.

Expand - Expands the width and/or height of the bitmap by an integer factor of one to four.

Fill Box - Fills a specified rectangular region with the default texture.

Fill Box with Specified Color - Fills a specified rectangular region with a chosen texture.

Fill Region(s) - Fill one or more closed regions with the default texture.

Fill Region(s) with Specified Color - Fill one or more closed regions with a chosen texture.

Follow Pixels - Menu subitems follow connected pixels and erase or draw over them**.**

Font - Sets the font family, size, and style for text placement**.**

Get Fro

BOONE-V-COE

Plaintiffs, W. H. Boone and J.T. Coe, brought this action against defendant, J.F. Coe, to recover certain damages alleged to have resulted from defendant's breach of a parol contract of lease for one year to commence at a future date. It appears from the petition that the defendant was the owner of a large farm in Ford Co., Texas. Plaintiffs were farmers living in Monroe Co., Kentucky. In the fall of 1909, defendant made a verbal contract with plaintiffs, whereby he rented to them his farm in Texas for a period of 12 months, to commence from the date of plaintiff's arrival at defendant's farm. Defendant agreed that if plaintiffs would leave Kentucky and move to defendant's farm in Texas and take over the farm for 12 months after their arrival, the defendant would prepare a home on the farm for plaintiffs which they could occupy as residence. Defendant also agreed .....

The plaintiffs did move from Kentucky during the fall of 1909. Upon their arrival at the farm in December, defendant denied them entry to the farm. Plaintiffs then left, returning to Kentucky. In reliance on the contract for the farm, plaintiffs incurred costs of ¤1400.

Defendant's demurred to the petition for damages which was sustained. Plaintiffs appeal. We concur with the previous decision.

.....

CrossFile Links
Destination NoteFile Browser

By: Daniel Jordan (Jordan.pa@Xerox.COM)

Stored: {qv}<notecards>1.3L>Library>DestNoteFileBrowser, .lcom, .ted
and {nb:parc:xerox}<notecards>1.3L>Library>DestNoteFileBrowser, .lcom, .ted
Written: October 10, 1988 by Dan Jordan
Last updated: January 6, 1989 by Dan Jordan
Supporting files: (all NC patches) dsjpatch075, dsjpatch076, rarpatch026

Introduction to the Destination NoteFile Browser

The Crossfile Links Destination Notefile Browser allows the user to modify and manage the destination notefiles associated with crossfile links in (source) notefiles. Using this facility, the user can easily move groups (or subgroups) of cross-linked notefiles to different directories and keep the cross-linked information between them up to date.

This facility features a "destination notefile browser" that displays in a window all destination notefiles for all crossfile links in a particular (source) notefile or group of notefiles. This browser is similar in appearance to a "file browser". Through a destination notefile browser, the directories or entire names of the destination notefiles associated with any crossfile links may be changed, to reflect changes in the location or name, respectively, of the actual destination notefile themselves.

Loading the Destination NoteFile Browser Package

Loading the Crossfile Links Destination Notefile Browser package is done in the same way as loading any other NoteCards library package. For example, you can type the following in an Interlisp exec:

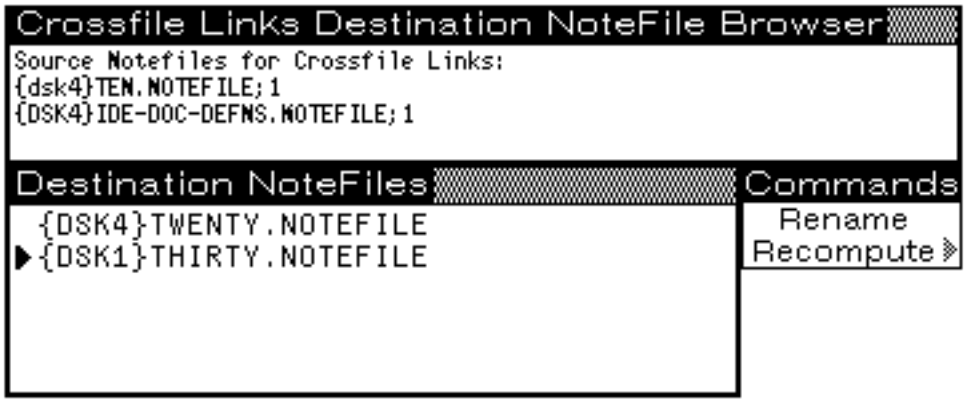
(FILESLOAD (FROM NOTECARDS) DESTNOTEFILEBROWSER)

Alternatively, you can add the atom DESTNOTEFILEBROWSER to the global var NOTECARDSLIBRARYFILES in your init file. This will ensure that DESTNOTEFILEBROWSER is loaded when bringing up a fresh sysout. (Note that the supporting patch files are to be found on the directory specified by the global var NOTECARDSNEXTDIRECTORIES.)

Using the Destination NoteFile Browser

Once the package is loaded, the user can access the Crossfile Links Destination Notefile Browser facility by selecting the "Destination NoteFile Browser" item on the "Other" menu item on the Notecards main menu. The user will be prompted to specify a region for a window and a "destination notefile browser" window will be opened in that region. The user will then be prompted to select the source notefiles that are to be used as the basis for the computed set of crossfile link destination notefiles. That is, for the set of source notefiles selected, ALL crossfile link destination notefiles for ALL crossfile links in the source notefiles will be computed. The source notefiles will be displayed in the top window of the browser, and any resultant destination notefiles for this set of source notefiles will be displayed in the main browser window.

The directories or entire names of any of the destination notefiles may be changed by selecting the destination notefiles of interest as you would items in a "file browser". There are two commands available on a destination notefile browser:



Rename The user will be prompted for a new name. The selected destination notefiles of all crossfile links for this group of source notefiles will have their names changed to this new name. If the new name is just a directory name, e.g., {qv}<ide>notefiles>, all selected destination notefiles will only have their directories changed and will retain their (root) filenames. If the new name is a complete filename (with directory), e.g., {qv}<ide>notefiles>new.notefile, all selected destination notefiles will have their names changed to exactly that new name. The browser will automatically be updated to reflect the destination notefile names.

Recompute This command is used to update, or recompute, the information in the browser (e.g., after new crossfile links have been added to the source notefiles.) There are two subitems on this menu: **Same source notefiles** (the default) and **New source notefiles**. The browser will be recomputed with the same source notefiles, unless

New source notefiles has been selected, in which case the user will be prompted for a new set of source notefiles.

NoteCards Library Packages

Stored as: {nb:parc:xerox}<notecards>1.3m>library>librarydoc.ted

ACE animation package	Denber
Animation-Cards	Bagley
Animation2	Bagley
Areafill	Bagley
BitmapEditor	Gaska
CaseCardFunctions	Cole
DestNoteFileBrowser	Jordan
LexDemo	Todd/Newman
NCBitmapCard	Gaska
NCCaseCluster	Trigg
NCChain	Trigg
NCClusterCard	Trigg
NCCollaboration	Trigg
NCCollaboratorCard	Trigg
NCDemo	Trigg
NCDraftCard	Rao
NCFunctionCard	Halasz
NCFullTextSearch	Todd
NCGestures	Trigg
NCGuidedTourCard	Trigg/Cohen/Ir ish
NCHacks	[Many]
NCHistoryCard	Irish
NCIdeaSketchCard	?
NCInspectorCard	Trigg
NCKey	Rao
NCLogger	Trigg/Irish
NCMailCard	Kelley
NCMaps	Feuerman
NCMergeFiles	Halasz
NCPath (NCPathLanguage, NCPathParse, NCPathUse)	Newman
NCPlotCard	Jordan
NCScreen	Trigg
NCSeditCard	Kelley
NCShadedCrossFileLinks	Irish
NCSentences	Halasz
NCSpreadSheetCard	Trigg
NCStat	Trigg
NCStructEditCard	?
NCStructEditBrowserCard	Trigg

NCTableTopCard	Shrager/Trigg/Cohen/Irish
NCToulminCard	Irish
ShapeOfIt	Kelley
TextCardKeys	

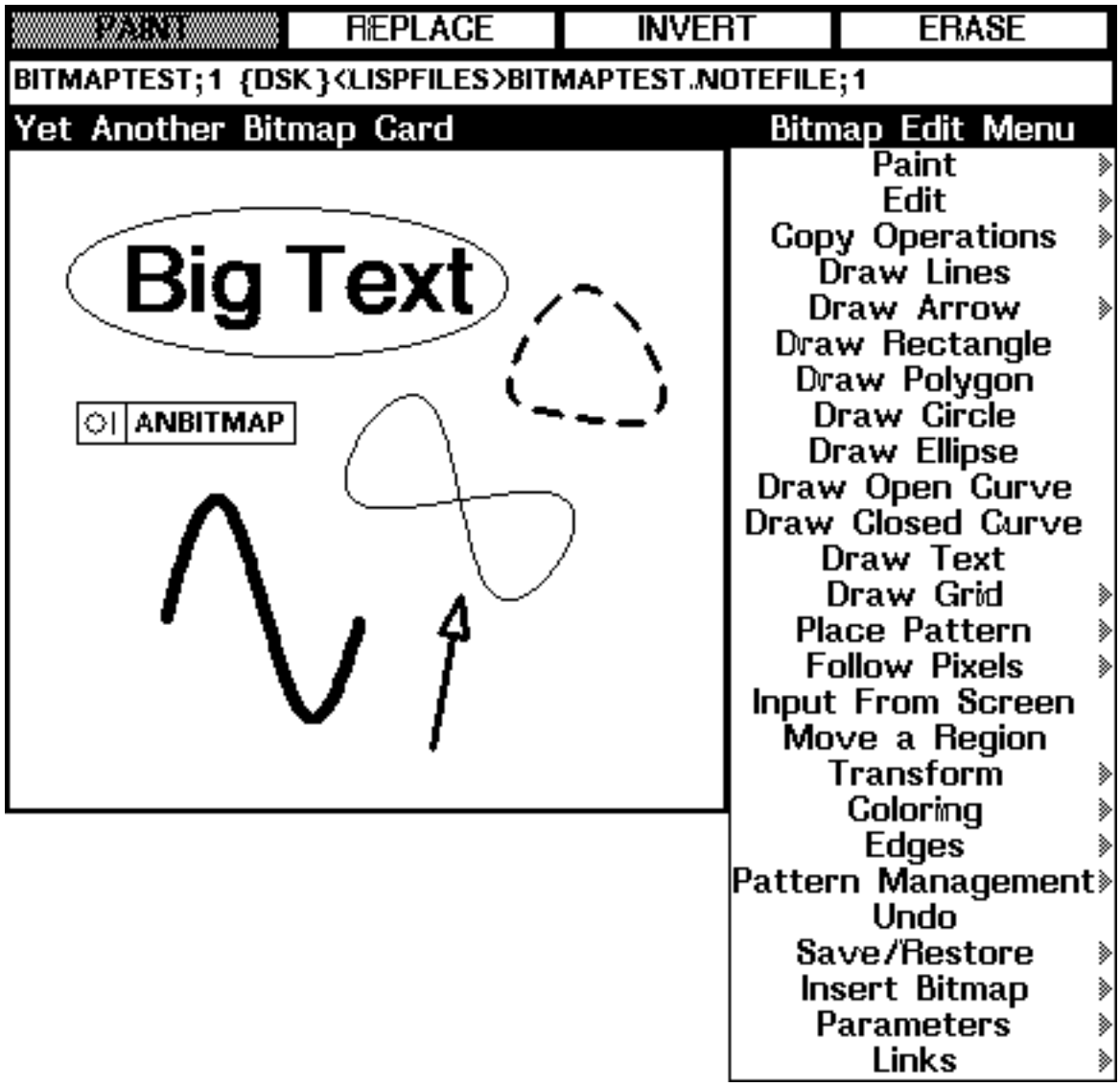
Notecards Bitmap Card

Vista Laboratory
Xerox Special Information Systems

November, 1988

This software implements bitmaps as a new substance type within Notecards. Although one can create and edit bitmaps within TEdit or SKETCH, the Bitmap Card gives more capabilities. The Bitmap Editor is described in a separate document (Reference 3). Editing is terminated via the Notecards left button title bar menu items (Close and Save, Delete Card, etc.).

Links are added, deleted, and moved via the Links menu item and its sub-items. Links are not actually part of the bitmap and so are not treated as items to be manipulated by the editor.



To install the Bitmap Card, load NCBITMAPCARD.LCOM which loads BITMAPEDITOR.LCOM and READNUM.LCOM. The Bitmap Editor also requires the Lispuser's package FILLREGION.

References:

1. Notecards Release 1.2k Reference Manual, Xerox Special Information Systems, April, 1986.

2. Notecards 1.3 Release Notes, Intelligent Systems Laboratory, Xerox Palo Alto Research Center, August, 1986.

3. Bitmap Editor, Vista Laboratory, Xerox Special Information Systems, November 1988.

NCCollaboration

By: Randy Trigg (Trigg.pa@Xerox.COM)

Stored: {qv}<notecards>1.3L>Library>NCCollaboration, .lcom, .ted
Written: June 14, 1988 by Peggy Irish
Last updated: June 15, 1988 by Peggy Irish.
See also NCCollaboratorCard and NCHistoryCard packages.

INTRODUCTION

This package was written to assist people using the draft-passing style of collaboration. This style refers to settings in which a small number of collaborators work closely together, but not at the same time. Rather, they take turns using a single shared hypertext workspace. These collaborators need the ability to discuss the use of the medium as well as the work constituting the content of the project. In particular, they need tools to support the processes by which they monitor their own and each other's work.

To this end, several mechanisms have been created to support collaboration. First, the notion of having a notefile understand the identities and certain properties of its individual collaborators has been developed. (See NCCollaboratorCard documentation.) This has also been extended so that multiple simultaneous users may be recognized, in the case where several people are using a notefile while sitting together at a workstation. Second, a method for recording the changes made to a notefile by a collaborator has been implemented. (See NCHistoryCard documentation.) Finally, a "collaboratization" process for configuring a notefile so that it contains these mechanisms has been provided, and is contained in this package. (The NCCollaboratorCard and NCHistoryCard packages will automatically be loaded when NCCollaboration is loaded**.)**

LOADING NCCOLLABORATION

Loading the NCCollaboration package adds the menu item "Collaboratize" to the session icon notefIle menu and to all notefIle icons' left button menus**Error in IMAGEOBJ GETFN: BMOBJ.GETFN3**UP **Error in IMAGEOBJ GETFN: BMOBJ.GETFN3** "

COLLABORATIZING A NOTEFILE

Selecting "Collaboratize" will cause a number of things to happen to the chosen notefile. A collaborator card will be created for the current collaborator. A new menu item, "Collaborator Info," will be added to the notefIle icon's middle button menu. The NoteFile Use History filebox will be created to hold all history cards in this notefile. Also, as part of the opening of a collaboratized notefile, a history card will be created

and opened for the current notefile session. This card will automatically be filed in the NoteFile Use History filebo**x.**

Creating a Collaborator Card

When a collaboratized notefile is opened, a menu of collaborators' names will appear, from which the current collaborator should select his name. During the collaboratization process, the notefile has no prior knowledge about collaborators; therefore the menu will contain only the "New Name(s)" option: Error in IMAGEOBJ GETFN: BMOBJ.GETFN3"¤

The user should select "New Name(s)," which will cause a new collaborator card to be opene Error in IMAGEOBJ GETFN: BMOBJ.GETFN3"þ

This collaborator card stores information about the current collaborator. The title of the card is initialized with the username of the collaborator, which he may change. The collaborator should fill in his or her initials by selecting the "?" with the left mouse button. A prompt window will appear, in which the collaborator can type his or her initials. These initials will appear in the titles of history cards created later. The collaborator may also change the Font property by selecting the existing Font value and choosing the desired font properties from the resulting men Error in IMAGEOBJ GETFN: BMOBJ.GETFN3""þ and then selecting DONE. The ExtraTEditProps property may also be changed by selecting the existing value of ExtraTEditProps and typing in the appropriate attribute-value pairs. Error in IMAGEOBJ GETFN: BMOBJ.GETFN3UP

The Font and ExtraTEditProps properties define the defaults to be used in cards created by this collaborator. Once the collaborator is satisfied with the collaborator card, he or she should select DON Error in IMAGEOBJ GETFN: BMOBJ.GETFN3U

COLLABORATOR INFO

If the notefIle icon middle button menu item "Collaborator Info" is selected, a menu containing the collaborators in this notefile will appear, with the current collaborator's name highlighted. Three commands also appear at the bottom of the men Error in IMAGEOBJ GETFN: BMOBJ.GETFN3"¤

The information in any of the collaborator cards may be changed at any time by selecting the name from this menu and editing the collaborator card that is brought up. The "New Name(s)" command is used as described in t**he Creating a Collaborator C**ard section, with the exception that a new collaborator card does not have a default title unless it is the first one created in the notefile. The "Remove Name" command allows the current user to remove a collaborator name from this notefile. The "Change current collaborator" command allows a different collaborator name to be chosen for the current session in this notefil**e.**

CREATING A HISTORY CARD

Once collaboratized, (and upon collaboratization), a new history card is automatically created for a notefile each time the notefile is opened. Each history card is automatically filed in the NoteFile Use History filebox. Upon creation, each history

card is titled with the current date and the initials of the current collaborator. A time stamp is entered in the card to indicate when the current session bega Error in IMAGEOBJ GETFN: BMOBJ.GETFN3 ‘‘‘‘
For more detailed information on history cards, please see the history card documentati

NCCollaboratorCard

By: Randy Trigg (Trigg.pa@Xerox.COM)

Stored: {qv}<notecards>1.3L>Library>NCCollaboratorCard, .lcom, .ted
Written: June 14, 1988 by Peggy Irish
Last updated: June 15, 1988 by Peggy Irish.
See also NCCollaboration and NCHistoryCard packages.

INTRODUCTION

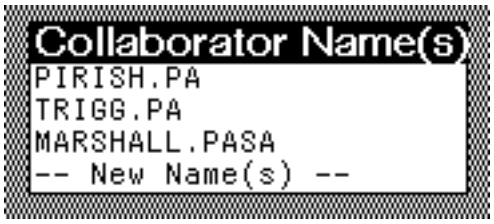
To allow a notefile to distinguish between collaborators, the Collaborator card type was created. A card of this type should be created for each person or set of persons working in a notefile. The collaborator card contains information relevant to collaboration, such as the user's name, initials, and preferred font. The initials are used by History cards, which are described in the History Card documentation. The font is automatically used to display text in cards created by the collaborator. When each collaborator or set of collaborators chooses a different font, it is easy to tell at a glance what work each person has contributed to a notefile's text. A collaborator card may also be created for multiple users, in the case where several people are working on a notefile together and want to claim group ownership of their contributions. When a collaborator (or set of collaborators) opens a "collaboratized" notefile, they are asked to identify themselves. The notefile then looks up their collaborator properties by searching for their collaborator card, and applies these properties to the working session.

LOADING NCCOLLABORATORCARD

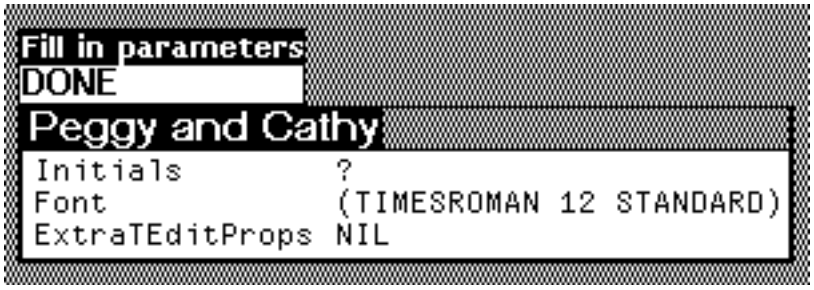
This package should be used in conjunction with the NCCollaboration package. NCCollaboration should be loaded, and it will automatically load the NCCollaboratorCard and NCHistoryCard packages. (See the NCCollaboration documentation for details of its use.)

OPENING A COLLABORATIZED NOTEFILE

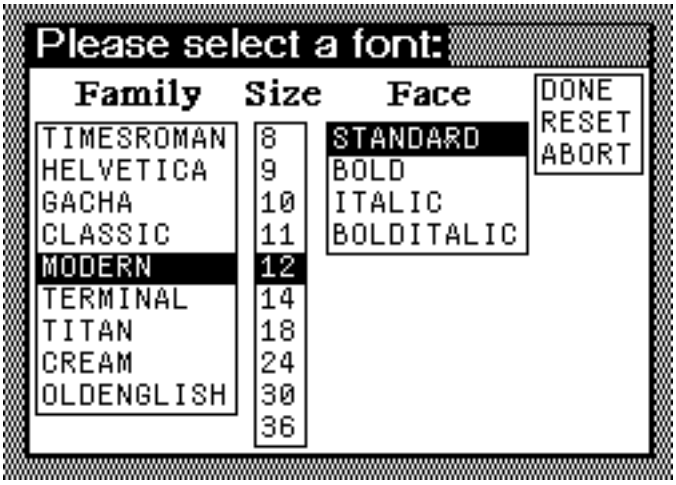
When a collaboratized notefile is opened, a menu of collaborators' names will appear, from which the current collaborator should select either his name or "New Name(s)." (During the collaboratization process, the notefile has no prior knowledge about collaborators; therefore the menu will contain only the "New Name(s)" option):



If one of the names is chosen, that collaborator will be instantiated as the current collaborator, and the corresponding collaborator information will be applied to the session. If "New Name(s)" is selected, the user will be prompted for a name. Then a new collaborator card will be created which has that name as its title:

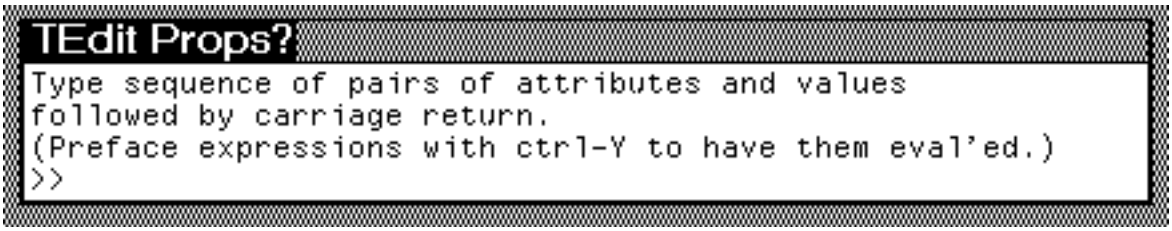


The collaborator should provide a value for the Initials property by selecting the "?" with the left mouse button. A prompt window will appear, in which the collaborator can type the desired initials. These initials will appear in the titles of history cards created later. The collaborator may also change the Font property by selecting the existing Font value and choosing the desired font properties from the resulting menu:

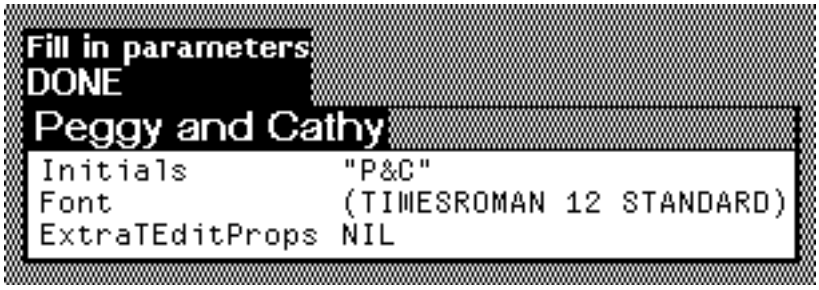


Selecting DONE from the right side of this menu completes the Font property edit operation. The ExtraTEditProps property may also be changed by selecting the

existing value of ExtraTEditProps and typing the appropriate attribute-value pairs into the prompt window:

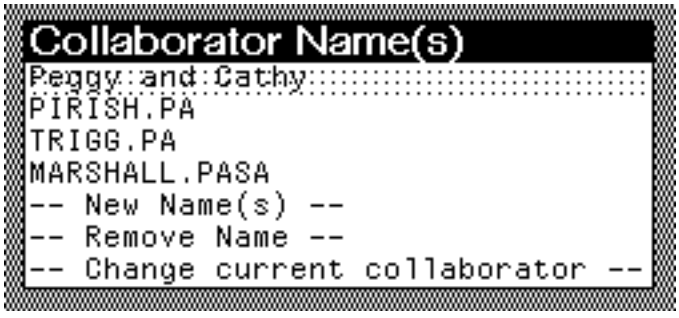


The Font and ExtraTEditProps properties define the defaults to be used in cards created by this collaborator. Once the collaborator is satisfied with the collaborator card, DONE should be selected.



COLLABORATOR INFO

If the notefIle icon middle button menu item "Collaborator Info" is selected, a menu containing the collaborators in this notefile will appear, with the current collaborator's name highlighted. Three commands also appear at the bottom of the menu:



The information in any of the collaborator cards may be changed at any time by selecting the name from this menu and editing the collaborator card that is brought up. The "New Name(s)" command is used as described in the **OPENING A COLLABORATIZED NOTEFILE** section. The "Remove Name" command allows the current user to remove a collaborator name from this notefile. The "Change current collaborator" command allows a different collaborator name to be chosen for the

current session in this notefile.

NCDraftCard

By: Ramana Rao (Rao.pa@Xerox.COM)

Stored: {qv}<notecards>1.3k>library>NCDraftCard, .dcom, .ted
Last updated: Mar 30, 1987.

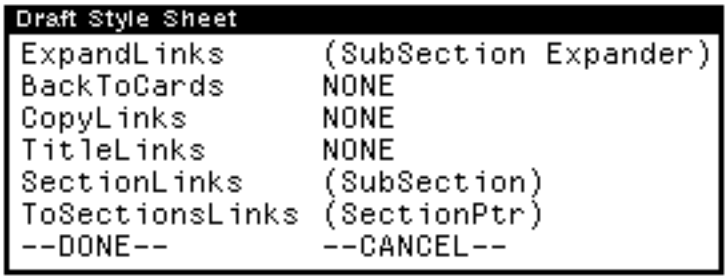
INTRODUCTION

This NoteCards library package defines a new card type called draft cards which supercede document cards. A draft card, primarily, collects the substance from a tree of cards starting from a set of root card, but also does auxiliary processing like sectioning and titling. (The process can be viewed and is implemented as a non-cyclic traversing and interpreting of a notecard network by invoking methods to process card and link types. The draft card defines a set of interpretation methods which implement functionality for document generation).

When a draft card is created, the user is asked to select a set of root card (using the standard NoteCards card selection interface). After this selection, the user is presented with a menu for defining the link and card categorys which are processed by the draft generator. After the draft is generated, a user can use the left button title bar menu to edit the draft style and recompute the draft. A programmer's interface function will be implemented on demand (or rather courteous request) or when I see a break in the clouds. Also true for a long list of desirable features.

CREATING A DRAFT

Draft cards are created by selecting Draft from the NewCards menu (assuming the NCDraftCard library package has been loaded). The user selects a set of root card for generating the document. Then the following draft style editor pops up allowing the user to edit the fields by selecting them. (This is not all there is to a document style as I will say below).



Selecting the items on the bottom line will generate the draft or cancel the creation. Each of the other lines represent a category of links or cards which provides specific functionality during draft generation. The *ExpandLinks* category define the link types which are expanded during draft generation i.e. the substance of the target card (the card pointed to by the link) is pulled into the draft and processed recursively. To prevent runaway loops, a card is only expanded once. The *BackToCards* category define the card types for which back links are put into the draft. Back links allow the user to return to the "underlying" card from the draft. *CopyLinks* are copied directly into the draft. *TitleLinks* produce titles in the draft by inserting the title of the card pointed to by the link. Similarly, *SectionLinks* produce section headings, but in addition they generate section numbers. The section numbers are generated according to "depth in tree using *SectionLinks*" i.e. the section number of a card has as many numbers as *SectionLinks* have been traversed to get to it. *ToSectionLinks* produce references to sections within the draft. Whenever a link of the *ToSectionLinks* category is encountered, the target card is checked for a section number and a reference of the form, "Section xx", is produced. This works for all sections which are part of the draft independent of whether they are before or after the section reference. All categories can be set to ALL, NONE, or a list of link or card types (using the interactive type selector which pops up when the user selects on the field).

The initial settings of the draft style sheet are inherited from a global default style. Any changes made to the style sheet only apply to the associated draft and do not affect the global default settings. To change the global default settings in the current sysout, the daring user can inspect and edit *ncdraft-default-style*. To maintain these defaults across sysouts the user can set *ncdraft-user-props* (a property list using the properties you see when you inspect *ncdraft-default-style*) in her init file. The user can also inspect and edit a draft's style by pulling across on "Edit Draft Style" item in the card menu. Inspecting styles will reveal a host of other parameters which will influence the draft generation process and will also more annoyingly reveal that the

field names which correspond to the link and card types have different names then in the style sheet editor. Use the field name you see in the inspector for adding an entry to *ncdraft-user-assoc* in you init file. (This is all hoaky, but at least functional. You are welcomed to bug me about how you can do something).

Alternatively draft cards can be created programatically as follows:

(NCP.MakeDraft <NoteFile> <rootcard> <style-assoc-list> <nodisplayflg> <props> <parent-fileboxes>)

> Currently Unimplemented. Bug me when you really need this. Creates and returns a draft card starting from <rootcard>. All parameters not set by <style-assoc-list> will be inherited from the default style. The resulting draft card will have the given props and parents.

DRAFT CARD OPERATIONS

The Draft Card operations are available via the left button (card) menu in the Draft Card card's title bar. **Recompute Draft** recomputes the draft using the associated root card and style. **Edit Draft Style** pops up the style sheet described above and allows you to change the link or card type categories. **Edit with Inspector** is a pull-across item of **Edit Draft Style** which allows you to inspect and edit all of the style parameters. Please READ the last section if you are going to do this.

KNOWN BUGS

Ha! All features, non-features, or future features.

FUTURE FEATURES

User Interface Improvements: Allow the user to edit all the style parameters with a style sheet editor that is like a form (like what FREEMENU would be if it were what it is trying to be). Right now my focus is more on what stylistic things a user might want to dictate, and not on how he will set these parameters.

Automatic generation of Table of Contents. You can do this in a separate draft card with the current functionality by being a little clever at defining which link types to expand, copy, section, or title depending on how you want it to come out.

Sectioning and Title Style parameters. Font choice and other looks options. The hidden style parameters are a small step in this direction, allowing bolding/notbolding of these as well as dictating number of carraige returns before and after a section.

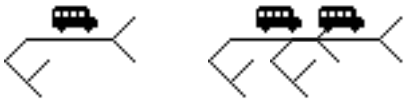
Figure Numbering and Figure references.

Bibliographic References. You can use titling to achieve this with great pain currently, but better ways are coming along. A quick and dirty hack first, and then a full service hack that Miller is working on.

Draft/Network Coupling. Editting the draft would automatically edit the underlying substance. Not an easy feature to provide, but probably a very useful one.

Generate Draft to Tedit Stream.

NCGuidedTourCard



By: Randy Trigg (Trigg.pa@Xerox.COM)

Stored: {qv}<notecards>1.3L>Library>NCGuidedTourCard, .lcom, .ted
Written: August 1, 1988 by Randy Trigg
Last updated: May 5, 1989 by Peggy Irish.
Loads NCTableTopCard package.

INTRODUCTION

The author of a complex NoteCards network is sometimes faced with the problem of conveying its meanig to future online readers using the same hypermedia environment. The NoteCards guided tours library package provides a way to reify and manipulate branching paths through a NoteCards network, thus allowing readers to explore the notefile under the guidance of the notefile's author.

A guided tour is a graph whose nodes are tabletop cards (see the documentation for the NCTableTopCard NoteCards library package) and whose edges are GuidedTour links connecting the cards. Because the tabletop cards may have other links connecting them to other parts of the network, the guided tour is a subgraph of the subnetwork containing those cards. The guided tour card is implemented as a specialization of the NoteCards browser card. Thus, creating and editing a guided tour is almost identical to creating and editing a browser. The biggest change is the ability to "run" the guided tour by means of a row of buttons across the top of the browser window. In this way, readers and authors access the tour through the same graph-based interface.

Readers interested in learning more about the history of and motivation for the guided tour package (as well as the tabletop facility) are directed to the paper "Guided Tours and Tabletops: Tools for Communicating in a Hypertext Environment," by Randall H. Trigg, Conference on Computer-Supported Cooperative Work, September 1988. (Copies can be obtained by writing to SSLPapers.pa@Xerox.com.)

LOADING THE NCGUIDEDTOURCARD PACKAGE

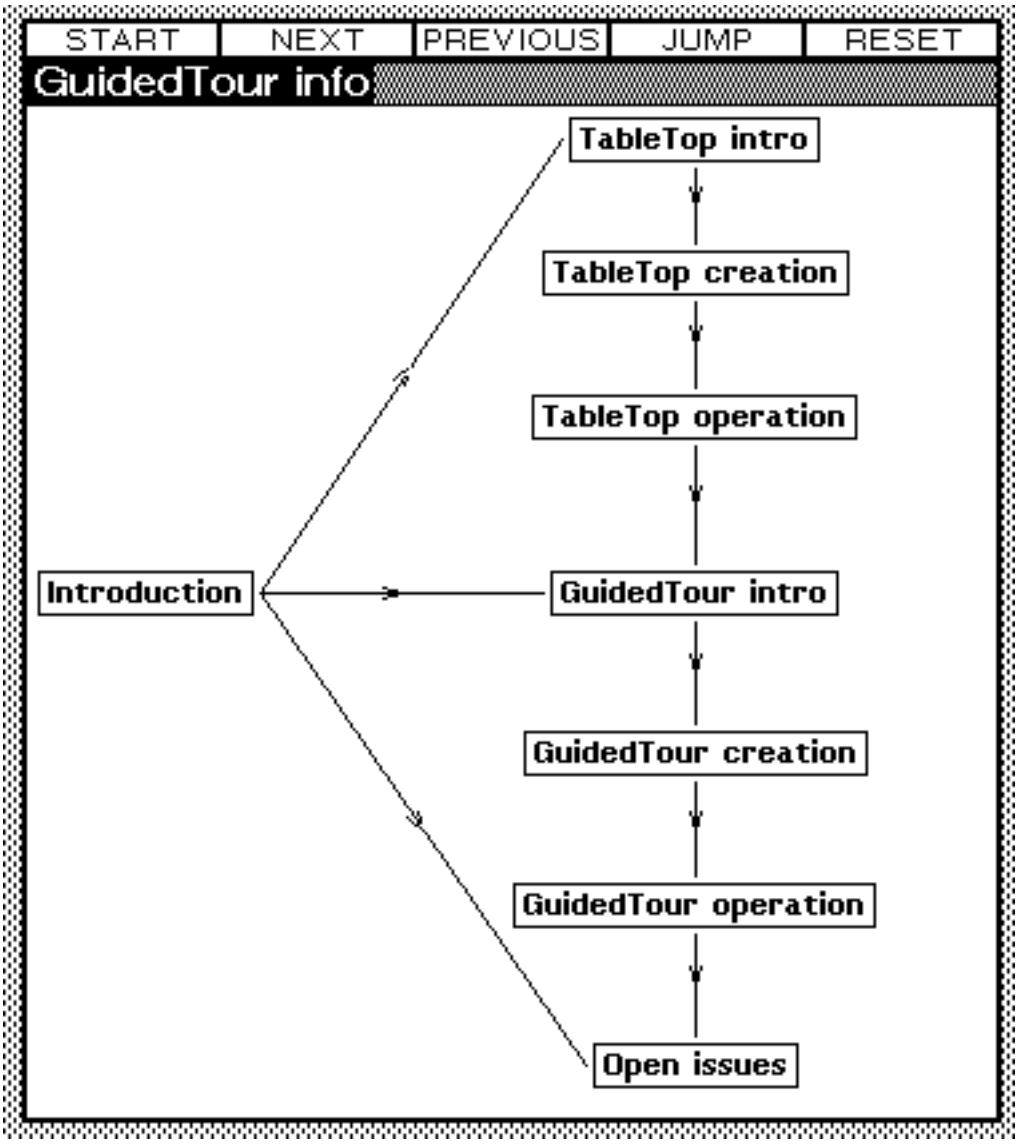
Loading the NCGuidedTourCard package is done in the same way as loading any other NoteCards card type library package. For example, you can type the following at an Interlisp exec:

```
(FILESLOAD (FROM NOTECARDS) NCGUIDEDTOURCARD)
```

Alternatively, you can add the atom NCGUIDEDTOURCARD to the global var NOTECARDSLIBRARYFILES in your init file. This will ensure that NCGUIDEDTOURCARD is loaded when bringing up a fresh sysout. Finally, if a link to a guided tour card is followed and the NCGUIDEDTOURCARD package is not currently loaded, then it will be auto-loaded by the system.

"RUNNING" A GUIDED TOUR

Operating or "running" a guided tour is done via the five buttons arrayed along the top of the guided tour card as shown below.



In each of the following operations, the current displayed tabletop is automatically closed down before bringing up the next. (This is made more efficient by not closing those cards shared by the current and next tabletops.)

START - Clicking on START causes the first tabletop in the tour to be brought up. For simple linear or singly-rooted tree structures there is an obvious "first" tabletop. Some tours, however, might have multiple roots, i.e. multiple nodes without incoming edges. In such cases, the user is asked to choose from a menu of possible starting tabletops.

NEXT - This is the standard way to move along a path. Clicking NEXT simply closes down the current tabletop and brings up the next one on the path. If the current node

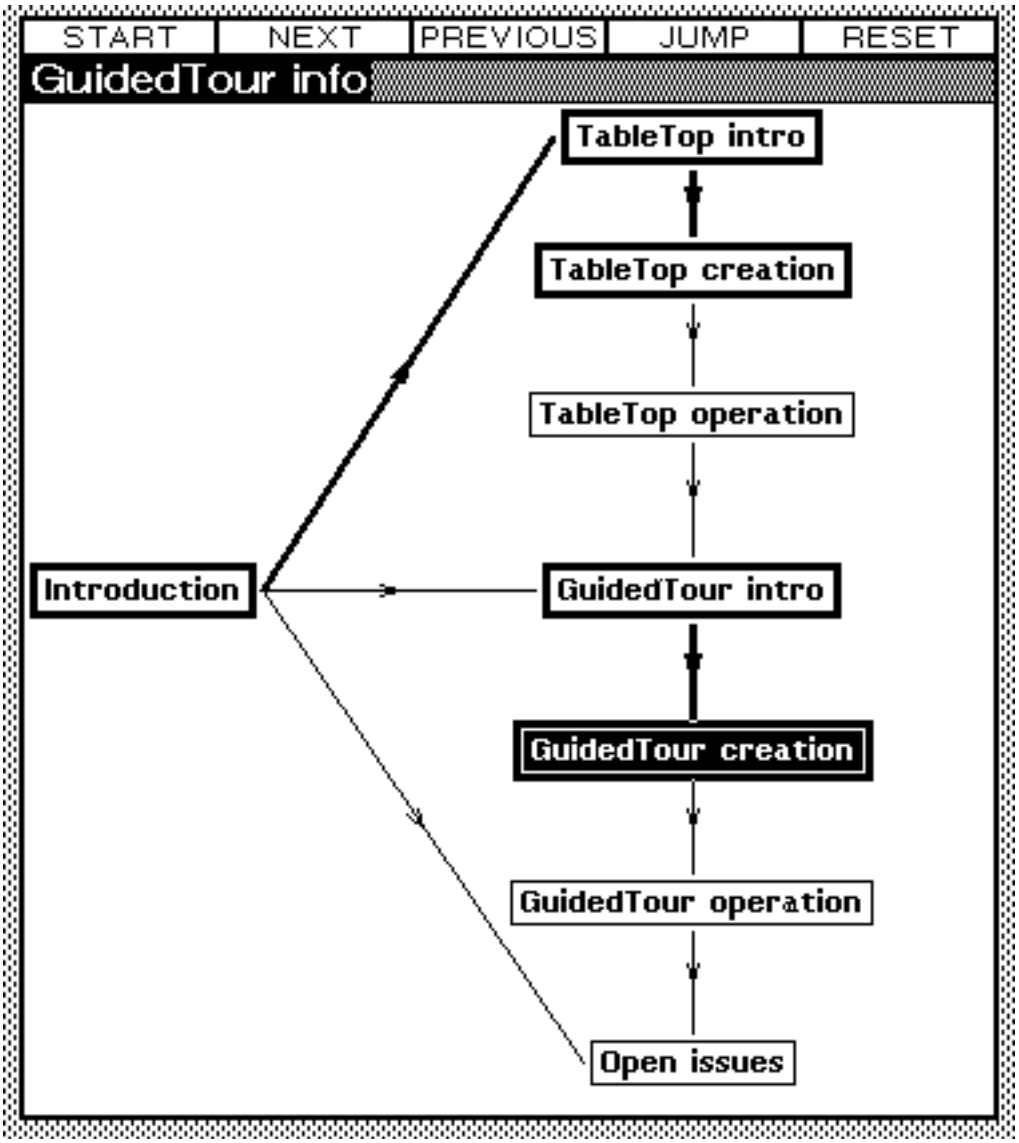
is a branch point having multiple outgoing edges, then the user is asked to choose from a menu of possible next tabletops. In any case, the new current node is highlighted as is the edge that was followed.

PREVIOUS - Clicking here provides the reader with a menu of tabletops already visited. These appear in the order in which they were visited. Clicking on an item in the menu effectively JUMP's to that point in the tour.

JUMP - This operation is used to leave the current path by jumping to an arbitrary tabletop selected by the user from the guided tour graph (again after closing down the current tabletop).

RESET - This operation closes the current tabletop and resets the "state" of the guided tour. In particular, any highlighting of nodes and links in the guided tour graph is turned off and the PREVIOUS list is cleared.

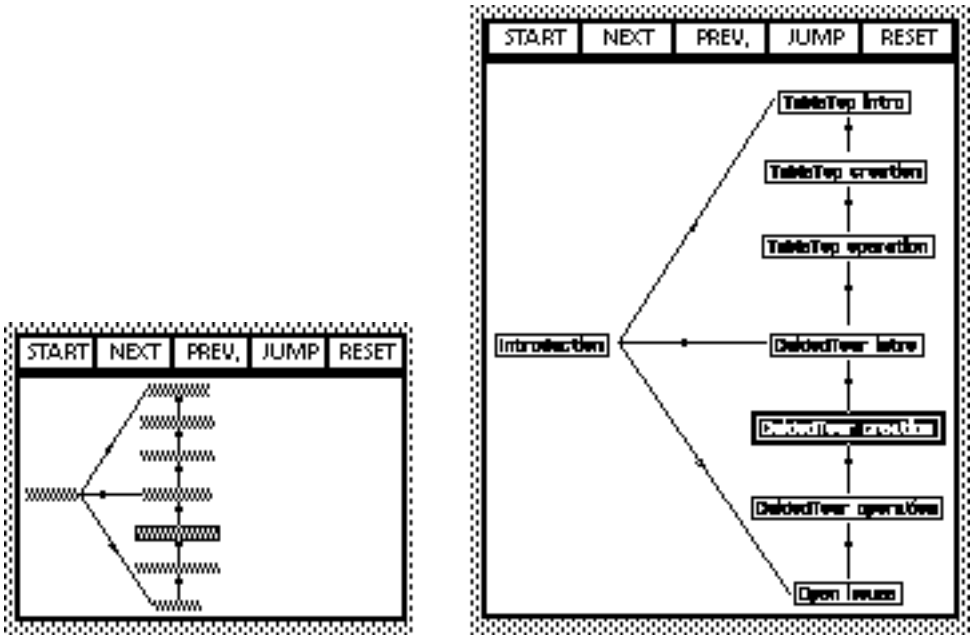
At any point in time, the graph structure displayed in the guided tour card provides various indicators of the reader's place in the tour.



The node in the graph whose tabletop is currently open is highlighted in reverse-video. The nodes for tabletops previously visited are drawn with heavier borders. Finally, the edges corresponding to links followed using the NEXT command are displayed in bold. The guided tour card appearing above shows a point in time when the reader has visited five cards. Starting at "Introduction," the top branch was taken, two tabletops were visited, and then a JUMP was made to "GuidedTour intro" whereupon NEXT was again used to move to the presently displayed tabletop "GuidedTour creation."

This standard mode of operating a guided tour (using the buttons along the top of the card) is extended in several ways. First, because screen space is at a premium and

because the window containing the guided tour graph is often large, the user can run the tour from the guided tour window's shrunken icon, which is a miniture version of the guided tour window. This icon may be shaped to suit the user's needs and to fit the available screen space. The current tour stop is highlighted in reverse-video in the shrunken icon.



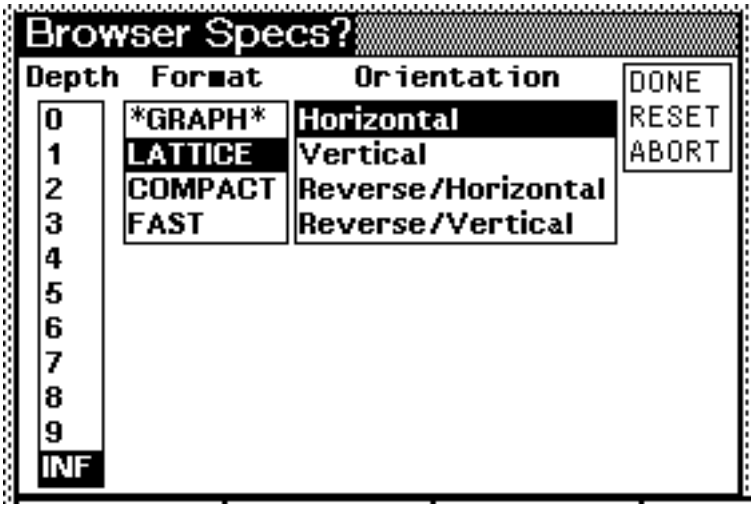
Second, it may be desirable to "circumvent" the tour by peeking at the cards in the next tabletop or by bringing up a tabletop without closing down the current one. This can be done directly by middle-buttoning the link icons appearing in the guided tour graph. (See the NCTableTopCard documentation for a description of tabletop operations.) Finally, it is important to emphasize that you can stray from the tour at any time by following links emanating from cards in the tabletop. Such explorations deeper into the notefile don't affect the current state of the tour.

CREATING AND MODIFYING GUIDED TOURS

There are two common styles of constructing tours. In the first, authors create an empty guided tour card and build and link together tabletops using the guided tour's editing menu. In the second, authors build a guided tour over already constructed tabletop cards. In any case, the elements of guided tour construction include creating guided tour cards, creating or including tabletops (and other tour nodes), linking together nodes in the tour, and modifying the tour.

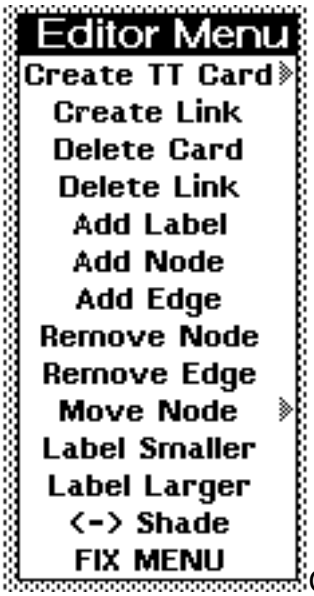
Creating a new guided tour card

Creating a guided tour card is very much like creating a Browser card. Just choose GuidedTour from the NewCards menu on the notefile icon. If you want the guided tour to be empty, then choose DONE when asked to specify the tour's roots. Otherwise, select a set of starting cards (usually tabletops). Unlike browser cards, guided tour cards don't require you to specify a set of link types to follow. There is a default set of link types (initially the single GuidedTour link type) used by guided tour cards during creation and Recompute. Because of this, the BrowserSpecs stylesheet for guided tour cards doesn't contain the usual columns for link types.



Adding nodes to a guided tour

All editing of the guided tour contents is done through the guided tour editing menu.



Ô Ä F Â@ F Â@ Ô Ä Ô 0 ÄÄ

´ÿ‡Ä ÄßÂÄ0Ä´ÈÄÕÂ@´ÿÄÕÂ@ Ô ?>łðÄðÞ < Ä Ô Ä Ä F 3 Â@ F Â@ Ô **Ä Ô**

Â@ Ô

‘Ü°

Ä Ô Û 0Ž>8| Ä F 0|ÛÌ3

˝û Â@ F ł0Â|Û@‡˝<ÔÄÛÔ0Á‡l

‘ŰŒIÄ F Ã 3>| Â@ F Â@ Ô **Ä Ô** Ä F Â@ F Â@ Ô Ä Ô Ì 08

Ÿ08Þ l‘ Â@ F

Ü =Þ⊣‘ÿÂ@ Õ ìû

ØÝl‘ Ä F

Ü 7lÜÞ‘ Â@ F

Ÿ07lÜÞ‘ Â@ Ô

Ÿ02o Ä Ô Ä F Â@ F Â@ — ÓÓÓÓÓÓÓÓÓÓÓÓ DDDDDDDDDDDDD(
*Node*s in the guided tour are usually tabletops, but can be arb**itrary card** types. To add an existing card as a node in the guided tour, use the Add Node menu item just as for standard browsers. To create a new tabletop card from scratch, click on the Create TT Card item. You will be asked for cards to be included in the tabletop just as though you created the tabletop from the NewCards menu on the notefile icon. By pulling *across on the Create TT Card menu item, you ca*n create guided tour nodes which are cards of other types. If you use such non-tabletop cards in your tour, you'll find that they generally act similar to tabletop cards that contain only one card. (But see the Handy Hooks section below to change this default behavior.)

Linking together nodes

Nodes in the guided tour can be linked toget**her using the Create** Link menu item. Normally, you will not be asked to specify a link type when creating a link using the editing menu. (If, however, the default set of link types has been changed to have greater than one element, the**n a menu of link types is popped up for you** to select from.) Note also that new links created from the editing menu are always "Global" links.

Modifying the g**uided tour and specifying roots**

As with standard browser cards, guided tours can be modified by adding new nodes and edges, removing old ones, rearranging nodes spatially within t**he tour, and annotating the tou**r using labels. In addition, it's usually a goo**d idea** t**o specify explicitly what the roots of the tour sh**ould be. (Roots can be (re-)specified by selec**ting t**he Change Browse**r Roots item from th**e middle button title bar menu of the guided tour card.) Other**wise, tour readers clicking on START have to choose fr**om a system-constructed menu of all nodes in the to**ur tha**t don't have inc**oming edges.**

PROGRAMMER'S INTERFACE TO GUIDED TOUR CARDS

To create a new guided to**ur card programmatically, call NCP.CreateCard just as you woul**d to create a browser card:

(NCP.CreateCard 'G**uidedT**our NoteFile T**itle NoDisplayFlg** Props ParentFileBoxes ParamList InterestedWindow)

See the documentati**on on NC**P.CreateCard for details on the args.

(NCGT.StartTourAtRoot GTCard)

Equivalent to hitting ST**ART on guided tour card GTCard.**

(NCGT.NextTourStop GTCard Don'tCloseCurrentFlg)

Equivalent to **hitti**ng NEXT on guide**d tour card GTCard e**xcept that if Don'tCloseCurrentFlg is non-nil, then current tab**letop won't** *be closed down firs*t.

(NCGT.PreviousTourStop GTCard Don'tCloseCurrentFlg)

Equivalent to hitting PREVIOUS on guided tour card GTCard except that if Don'tCloseCurrentFlg is non-nil, then current tabletop won't be closed down first.

(NCGT.JumpToTourStop GTCard Don'tCloseCurrentFlg NextNode)

Equivalent to hitting JUMP on guided tour card GTCard *except that if*

*Don't*CloseCurrentFlg is non-nil, then current tabletop won't be closed down first. Also if NextNode is a valid node in the graph, then user won't get bugged; we'll jump directly to that card.
(NCGT.ResetTour GTCard Don'tCloseCurrentFlg)
Equivalent to *hitting RESET on* guided tour card GTCard except that if Don'tCloseCurrentFlg is non-nil, then current tabletop won't be closed down first.
Handy Hooks
GuidedTourBringUpFn. If this prop is on a card type's atom, then when cards of that type are encountered in guided tours, they are brought up by calling this function. (This is useful when you don't want the default behavior for stops in the tour that aren't tabletop cards.) This function is expected to return the list of cards it brings up.
GuidedTourCloseDownFn. If this prop is on a card type's atom, then when cards of that type are encountered i*n guided tours, they are* closed down by calling this function. (Note: This prop doesn't seem to be called anywhere, as of the version of October 19, 1988.)
GTNextTourStopFn. This function is called by NCGT.NextTourStop, and its job is to "filter" the list of possible next tour stops from the *current tour stop.* The list of next cards (the destination cards of the NCGT.GuidedTourLinkTypes links from the current tour stop) is passed to the *GTNextTourStopFn, w*hich returns some sublist of the next cards. Both the card type atom of the card in the current tour stop and the car*d type atom of the gu*ided tour card are checked for the GTNextTourStopFn. If both card type atoms have a GTNextTourStopFn, only the one from the current tour stop cardtype is called.
GTCreateGuidedTourLinkFn. If this prop is on the card type atom of the guided tour card, then this function is called whenever a new link is created using the graph edit menu of the guided tour card. This is a way of getting control over *the type of link use*d, whether it's global or local, etc.
GTResetGuidedTourFn. If this prop is on the card type atom of the guided tour card, then this function is called whenever the guided tour is reset.
GTBringUpCachingFn. This is an NCP.CardUserDataProp of a guided tour card whi*ch is called each time a card* or tabletop is brought up.
GTCloseDownCachingFn. This is an NCP.CardUserDataProp of a guided tour card which is called each time a card or tabletop is closed down. If this prop is non-nil, all cards in the guided tour stop which are being closed are merely undisplayed -- they are still cached. It is the job of the **GTBringUpCachingF***n and the GTCloseDownCac***h**ingFn to decide which of these cached but undisplayed tour stop cards should be uncached.
GTCleanUpCachingFn. This is an NCP.CardUserDataProp of a guided tour card which is **calle**d when a guided tour card is closed. Its job is to clean up whatever cached but undisplayed guided tour stop cards are still laying around from the GTBringUpCachingFn and the GTCleanUpCachingFn.
ImmediatelyGoToNextGTStopFn. Whenever a new **guided t**our stop card is brought

up, its card type atom is checked for this function. If the function exists and returns non-NIL, NCGT.NextTourStop is immediately called. Thus, for example, it is possible for a us**er to trave**l through a whole guided tour, by merely pressing a guided tour's Start button.

Global Variables

NCGT.GuidedTourLinkTypes

This variable is initially bound to the single atom GuidedTour. The link types on this list will be the only legal link types users will be able to create in a guid

The NCHACKS Library Package

Xerox Corporation

Randy Trigg

[First written: 8/23/85 Randy Trigg]
[Last updated: 8/18/87 Peggy Irish]
[Code: {qv}<notecards>1.3k>library>nchacks.dcom]
[Doc: {qv}<notecards>1.3k>library>nchacks.ted]

This document describes a library package containing miscellaneous handy functions written using the programmer's interface for use in NoteCards Release 1.3k. It is hoped that you will find these useful not only for their functionality, but also as examples of the sort of code that can be written using the programmer's interface.

I encourage any readers who have created such functions themselves to send me the code and a short description and I'll incorporate into this package.

These functions all apply to groups of cards (and often to a notefile as well) and allow searching for text, global replacement of text, removal of deleted link icons, sorting of link icons, making link icons invisible, card collection by date, and creating chains of cards using a given link type.

The following functions are applicable to sets of cards and allow global searching, replacing, and chain creation. For those that take a CardsOrNoteFile argument, it can either be single card, a list of cards or a notefile. In the latter case, the entire notefile is used. To allow user selection of cards, pass (NCP.SelectCards) as the CardsOrNoteFile argument. (NCHACKS.MakeChain takes a Cards argument which can either be a single card or a list of cards, but not a notefile.)

(NCHACKS.TextSearch String WildCards? CardsOrNoteFile ReturnLocsFlg)

This goes through the list of cards looking for occurrences of text string String. WildCards? being non-nil means wild card characters may appear in String. The pound sign character '#' matches any single character, asterisk '*' matches any sequence of characters, and single quote, "'" can be used to quote one of the wildcard characters.

If ReturnLocsFlg is nil, then this just returns a list of cards containing at least one occurrence of String. If ReturnLocsFlg is non-nil, then this returns a list of lists. Each sublist has as first element a card ID followed by the locations of occurences of String. These are single integers if WildCards? is nil and correspond to the location of the first character of String in the text. If WildCards? is non-nil, then these are two element lists of start and end locations of the matching string.

Any cards in Cards that are not TEdit-based cards are ignored (i.e. their card type must inherit from the Text card type).

(NCHACKS.GlobalTextReplace String1 String2 WildCards? CardsOrNoteFile)

This goes through the list of cards replacing every occurrence of text string String1 by String2. WildCards? being non-nil means wild card characters may appear in String1. The pound sign character '#' matches any single character, asterisk '*' matches any sequence of characters, and single quote, "'" can be used to quote one of the wildcard characters.

Any cards in Cards that are not TEdit-based cards are ignored (i.e. their card type must inherit from the Text card type).

(NCHACKS.RemoveDeletedIconsFromTextCards CardsOrNoteFile)

This removes all deleted link icons from the TEdit-based cards in the list of cards.

(NCHACKS.ReorderLinkIconsInTextCards CardsOrNoteFile OrderingFn QuietFlg)

This moves link icons around in each of the TEdit-based cards in the list of cards so that they appear in sorted order. OrderingFn should be a function which accepts two link arguments (the destination cards of the links) and returns non-nil if the first should appear ahead of the second. OrderingFn can also be the atom ALPHABETIZE in which case NC.IDAlphOrder (orders by title of link's destination card) will be the ordering fn used. If OrderingFn is NIL, then the card's OrderingFn card prop is checked. If that card prop is NIL, then NC.IDAlphOrder is again used.

(NCHACKS.MakeLinkIconsInvisible CardsOrLinksOrNoteFile Invisiblity)

CardsOrLinksOrNoteFile should be one of: a card, a link, a list of cards and links, or a notefile. Make any link icons for given links or contained in given cards or notefile (but only those in TEdit-based cards) invisible or not depending on whether Invisibility is ON or OFF.

(NCHACKS.DateSearch DateString1 DateString2 CardsOrNoteFile)

This goes through the list of cards looking for occurrences of card parts modified between the dates DateString1 and DateString2. If DateString1 is NIL, then it defaults to a very early date. If DateString2 is NIL, then it defaults to current date. If you provide values for either DateString1 or DateString2, they should be strings in the same format as that returned by the (DATE) format, i.e. " 1-Jan-85 00:00:00".

This returns a list of lists, one sublist for each card modified between the given dates. This sublist consists of the card ID followed by two-element lists containing the card part name that was modified and the date of last modification. For example, if there was one hit, you might get something like (({CardObject}#56,164470 (SUBSTANCEDATE "23-Aug-85 19:15:27") (TITLEDATE "23-Aug-85 19:15:44")))

(NCHACKS.CardsModifiedBetweenDates CardsOrNoteFile CardTypes ModifiedRange LastModifiedRange CreatedRange)

This function goes through the list of cards or the notefile looking for cards of types in CardTypes which satisfy all of these criteria: they were modified in the range ModifiedRange, and last modified in the range LastModifiedRange, and created in the range CreatedRange. Each range should be a list of two dates, each in the form DD-Mon-YY (i.e. " 1-Jan-85"). Any of the ranges may be NIL, which means not to include that type of modification/creation. If the first (earlier) date in a range is NIL, then it defaults to the earliest known date. If the second (later) date in a range is NIL, then it defaults to the current date. CardTypes should be a list of card types. CardsOrNoteFile should be a list of cards or an open notefile. The cards which meet all of the given criteria are returned in a list.

Examples:

(NCHACKS.CardsModifiedBetweenDates (NCP.WNF) ’(Text FileBox) ("3-Jul-87" "10-Jul-87") NIL NIL)

would return all cards of type Text and FileBox in the notefile pointed to by the mouse which were modified between the dates of July 3, 1987 and July 10, 1987.

(NCHACKS.CardsModifiedBetweenDates (NCP.WNF) ’Text NIL ("1-Jun-87" NIL) (NIL "31-May-87"))

would return all cards of type Text in the notefile pointed to by the mouse which were created on or before May 31, 1987, and last modified after June 1, 1987.

(NCHACKS.MakeChain LinkType Cards Position AddCRFlg)

This creates links between successive cards in Cards each of type LinkType positioned at Position. Thus the first card in Cards will be linked to the second, the second to the third, etc. (No link will be built from the last card in Cards.) Position should be in the same form as that accepted by NCP.CardAddText, i.e. either an integer location or one of the litatoms START or END. If the AddCRFlg is non-nil and Position is one of START or END, then a carriage return is inserted after the link icon in the case of START or before the link icon in the case of END.

Any cards in Cards that are not TEdit-based cards are ignored (i.e. their card type must inherit from the Text card type).

NCKey

By: Ramana Rao (Rao.pa@Xerox.COM)

Stored: {qv}<notecards>1.3k>library>NCKey, .dcom, .ted
Last updated: July 13, 1987.
Requires: TEditKey Lispusers Package

INTRODUCTION

This NoteCards library package provides an extensible keyboard interface to useful Notecard Operations (useful means that I needed them at the time). In particular, NCKey provides keystroke commands for creating a link to a new text card using the current selection as either the title or the text of the new card (I was taking a large document and bashing it into a notecards hierarchy, so you can see why these were useful), entitling a card using the current selection, deleting all the "DeletedLinkIcon", and coercing the type and displaymode of all the links in the current card. However, the particular functions provided are not so important as the ability to easily bind your own functions to keys. Possibilities for functions as demonstrated by my bindings (the default bindings) include using available functions in NCHACKS or writing your own (or begging, asking, or demanding some one else to write) functions using the NC Programmer's Interface.

USING THE KEYBOARD INTERFACE

NCKey requires TEditKey but doesn't expect that it be previously loaded. NCKey actions are invoked by typing Meta-W followed by a single character. Currently, the single character can not be Meta-ed (but I realize that this is a pain since you want to be able to keep holding down the Meta key while striking the second character). The mapping from dispatch character to an action routine is made using *NCKEY-BINDINGS*. The default mapping is the following:

Inserts a Link Using Selection as Title	(J j)
Inserts a Link Using Selection as Text	(K k)
Removes Deleted Link Icons	(R r)
Sets Title Using Selection	(T t)
Coerces Links using Global Parameters	(L l)
Brings Up Help Menu	(?)

These items and their operation are described in greater detail below.

CUSTOMIZING OR EXTENDING THE KEYBOARD INTERFACE

The provided mechanism isn't particularly elaborate or elegant, but it's basically good enough for most purposes. You can edit the variable *NCKEY-BINDINGS* to remap the bindings to the provided action routines or to bind new action routines. This variable is INITVAR-red by this package, so users can put their own binding in their INIT files. Below is a specification of the syntax of *NCKEY-BINDINGS*:

NCKEY-BINDINGS = (<Binding Triple>*)

<Binding Triple> = (<Key Group> <Action-Fn> <Description String>) | NIL

A NIL <Binding Triple> is used to indicate a space in the Help menu.

<Key Group> = <Key> or (<Key>*)

<Action-Fn> = <function> or (<function> <Action-arg>*).

In the first case, the function is applied to a default set of parameters (<function> CARD NOTEFILE TEXTSTREAM SEL). In the second case, the form is eval-led.

<Action-arg> = The keywords CARD, NOTEFILE, TEXTSTREAM, TEXTOBJ, or SEL or an evalable form using these keywords as arguments. For example, either of the following may be be used as an Action-Fn Item:

(RandomFunction CARD NOTEFILE)

(AnotherRandomFunction (HelloFn SEL) TEXTOBJ).

<Description String> is used in the Help menu (Meta-W ?) to indicate what the key is bound to.

(NCKEY-BUILD-HLP)

After editting *NCKEY-BINDINGS*, you must call this function to update the Help menu (Meta-W ?).

PROVIDED ACTION-FNS AND PARAMETERS

NCKEY-DEFAULT-CARDTYPE = Text

NCKEY-DEFAULT-TITLE = "Untitled"

NCKEY-DEFAULT-LINKTYPE = Unspecified

NCKEY-DEFAULT-DISPLAYMODE = NIL

These four user parameters are used by the Provided Action-fns as default values. There is a tradeoff between just using default value and prompting the user for values. I opted for default values because 1) they're a little easier and 2) a major point of this was to get something keyboardishly quick. All of these user parameters are INITVAR-red by this package, so users can put their own bindings in their INIT files.

NCKEY-CREATE-LINK-WITH-TITLE

NCKEY-CREATE-LINK-WITH-TEXT

These create a new card and insert a link to this card into the current card. The current selection is used as either the title or the text of the new card, and is also deleted from the current card.

NCKEY-MAP-LINKS

This function coerces the type and displaymode of all the links in the current card using *NCKEY-DEFAULT-LINKTYPE* and *NCKEY-DEFAULT-DISPLAYMODE* .

(NCHACKS.RemoveDeletedIconsFromTextCards CARD)

This function is obviously a part of NCHACKS and demonstrates an Action-FN which takes arguments other than the default arguments. It removes all the delete icons from the current card.

NCKEY-ENTITLE

This function sets the title of the current card using the current selection. It leaves the current selection alone.

NCKEY-HELP

This function brings up the help menu of the current key bindings. Remember that (NCKEY-BUILD-HELP) should be called to update the help menu after editting the key bindings.

KNOWN PROBLEMS

As mentioned above, the dispatch character can’t be a META-character.

NCPath

By: David Newman (Newman.pasa@Xerox.COM)

NCPathParse, NCPathUse

INTRODUCTION

This file is a module designed to allow a programmer to specify a path through a notecards network and a root card, and obtain from these a data structure describing all the paths through the network from the root card that match the path specification. The function NCPath.FSM.PathCollect is the top level function. In conjunction with NCPathParse (which translates a simple path language into the path descriptor that NCPath.FSM.PathCollect requires), it returns a collection of paths for the programmer to manipulate. A typical call to the NCPath functions should look like the following.

```
(NCPath.FSM.PathCollect (NCPathParse <Path-description> ) <RootCard>)
```

This package and its functions are implemented on a bastardized finite state machine model. The key difference is that in order to allow the imposition of limits on loops in the finite state network (limits on the number of times a loop can be traversed), the number of times each loop is traversed must be recorded in the finite state machine. In addition, this system will not collect any paths which attempt to traverse the same link of a notefile in the same direction a second time.

The data structures of this system include the InterLisp datatypes NCPathFSM, NCPathFSMNode, and NCPathPathStep, and two list structures. A collection is a list of paths. A path is a list of pathsteps with the root card at the end. Paths contain the steps taken in reverse order, and the various paths in a collection share cons cells in order to conserve memory. Please note also the descriptions of the three records listed below.

RECORDS

NCPathFSM [Datatype]

The fields of an NCPathFSM are InitialState, CurrentState, Path, and LoopLimitAList. Each is a Lisp POINTER. InitialState and CurrentState contain objects of type NCPathFSMNode, and are the current and initial states of the finite state machine respectively. The Path field contains the intermediate

results of a traversal through the NoteCards network. Finally, LoopLimitAList is an a-list that keeps track of the number of times any loop in the finite state transition network has been traversed.

NCPathFSMNode [Datatype]
The fields of an NCPathFSMNode are Predicate, Card/Link, Direction, NextNodes, and LoopLimit. The first and fourth are POINTERs, the second and third are FLAGs, and the last is an INTEGER. The Predicate field points to a function or LAMBDA expression, which when applied to a notecard or link returns T or NIL; this indicates whether the item in question will satisfy the next step in the path description. The Card/Link flag indicates whether the Predicate is to be applied to links or cards, (T indicates links and is the default, while NIL indicates cards), and the Direction flag indicates whether a link is to be followed in a forward direction or a backward direction (T indicates forward, and is the default setting). NextNodes is a list of the next nodes in the finite state transition network, and hence is essentially a list of arcs away from the current node. Lastly, LoopLimit indicates the number of times that a loop beginning with the current node may be traversed; its default value is one.

NCPathPathStep [Datatype]
An NCPathPathStep has only two fields, both of which are POINTERS. the Link field contains a link in the notecards network, and the Direction field indicates the direction that this link was followed.

FUNCTIONS

NCPath.FSM.PathCollect *PathSpec RootCard*) [Function]
This function collects a list of complete paths starting at RootCard as specified by PathSpec The paths are really a network as they share CONS cells, and are actually reversed. The end of each is a pointer to the ID for RootCard The first item in each path is actually the NCPathFSM representing the remaining parts of the path to be collected. When the paths are complete, this is a NIL.

(NCPath.FSM.RealPathCollect *FSMInstance RootCard*) [Function]
This function does the real work of NCPath.FSM.PathCollect after that function has done the error checking. (FinishedPaths has an extra NIL at the front, so the CDR at the end removes it)

(NCPath.FSM.FirstStep *RootCard FSMInstance*) [Function]
This function takes care of the case where the first NCPathFSMNode has a list as its NextState.

(NCPath.FSM.RealFirstStep *RootCard FSMInstance*) [Function]
This function is specially intended to get the first set of links from a path specification (the NCPathFSM) and its root card. It constructs the first level or two of the tree of working paths. The function handles the special case where the first two FSMNodes in NCPathFSM include card predicates.

(NCPath.FSM.ListFirstSteps *RootCard FSMInstance*) [Function]
This function gets the first set of links from a path specification and a root card. End is bound specially so that all the paths created will share their last cons cells.

(NCPath.FSM.AddPotentialSteps *FSMInstance*) [Function]
Add all potential steps to the Path in *FSMInstance* without checking them against any specification. Returns a list of NCPathFSMs

(NCPath.FSM.ListMultiplePaths *FSMInstance*) [Function]
This function is intended to help out with the problem of true FSMs. It takes a NCPathFSM that has a list of FSMNodes as its CurrentState, and returns a list of NCPathFSMs each of which has one of the FSMNodes as its CurrentState.

(NCPath.FSM.ComputeMultilpeCollections *FSMInstance*) [Function]
This function handles the case where the NextNodes field of the NCPathFSMNode in the CurrentState field of *FSMInstance* is a list rather than an individual FSMNode.

(NCPath.FSM.AddNextSteps *FSMInstance*) [Function]
This function adds the next set of steps to a particular path. That is, it takes a path, gets the NCPathFSM representing the PathSpec and the last link of the path, and adds each of the appropriate next steps to that path - returning a list of several paths with common CONS cells. It also puts the next state of the NCPathFSM on the front of the Path for the next time around.

(NCPath.FSM.IncrementUseCount *FSMInstance*) [Function]
Increment the count kept in the AList on the FSM indicating how many times the CurrentState has been used in this path.

(NCPath.FSM.AddStep Step *FSMInstance*) [Function]
Given an old FSMInstance and a new step, this function adds the step to the path in the FSMInstance. This function also increments the CurrentState to the NextState

(NCPath.FSM.NextState *FSMInstance*) [Function]
This function is returns the next state in *FSMInstance* which defines a path specification. If the CurrentState of *FSMInstance* is NIL, we report the error and return NIL.

(NCPath.FSM.LoopLimitExceededP *FSMInstance*) [Function]
This predicate determines whether or not the FSMNodeInstance stored in the CurrentState of *FSMInstance* has been used too many time to get to the current place in the path. This function determines if a limited loop has been followed too many times.

(NCPath.FSMState.ComputeCollection *FSMInstance*) [Function]
This function takes an NCPathFSM as its single argument. The CurrentState of the NCPathFSM specifies a card rather than a link. If the last step of the path meets specification, and the next node in the NCPathFSM specifies a link, the path is returned, If the last step of the path meets the specification, and the next node in the NCPathFSM specifies a card, the intervening links are added, and the list of new paths is returned.

(NCPath.FSMState.ListNextSteps *PathStepORCard FSMNodeInstance*) [Function]

This function finds the next steps that can be taken from a particular point in a path. It accepts either a card or a link as the indicator of the current path position and it returns a list of links.

(NCPath.FSMState.SpecifiesCardP *FSMState*) [Function]
This function returns T iff the FSM-State is not NIL and it's CARD/LINK flag indicates that the

predicate is to be applied to a CARD.

(NCPath.FSMState.SpecifiesLinkP *FSMNodeInstance*) [Function]
This function returns T iff the FSM-State is not NIL and the LINK/CARD flag indicates that the predicate is to be applied to a LINK.

(NCPath.FSMState.TerminalP *FSMNode*) [Function]
This function determines whether or not a NCPathFSM node is a terminal node. That is, whether or not there is a transition to another node from this one.

(NCPath.Collection.ComputeNextCollection *PathCollection*) [Function]
This function computes a new list of FSMs from an old one. The new FSMs are created by taking an old FSM and adding one step to its path. This function also distinguishes the case where the next step is specified as a card predicate rather than a link predicate.

(NCPath.Collection.CollectMultiplePaths *Collection*) [Function]
This function takes a collection of NCPathFSMs and expands those that have multiple FSMNodes as their CurrentState into a list of NCPathFSMs each having one FSMNode as its current state.

(NCPath.Collection.ListRemovablePaths *Collection*) [Function]
List those paths of Collection which are complete, or loopy, or which have traversed the same loop of the FSM too many times.

(NCPath.Collection.ListFinishedPaths *Collection*) [Function]
List those paths in Collection that are complete as specified. These are the paths that the user wants to see.

(NCPath.Path.Create *Root FirstStepLink FSMInstance*) [Function]
This function creates a new path. Since a path is a list of NCPathPathStep in reverse order with a root card ID at the end, we create the first step and the root and cons them together.

(NCPath.Path.End *Path*) [Function]
This function returns the end card in a path by taking the last step of the path and using the Direction field to determine whether to return the source or destination of the link in the Link field.

(NCPath.Path.AddStep *Step Path*) [Function]
Given Path and a new step, this function adds the step to the Path

(NCPath.Path.LastStep *Path*) [Function]
Since a Path is a reversed list, we just take the CAR to get the last step.

(NCPath.Path.StepInPathP *TestStep P*ath) [Function]
This predicate determines *if TestS*tep is *in P*ath or not.
(NCPath.Path.EQU*AL Path1 P*ath2) [Function]
This function checks for equality of two paths that are still reversed, and have the root card at the end. It is significantly faster than EQUALALL, but uses a number of CONS cells.
(NCPath.Path.Loop*sP P*ath) [Function]
This predicate returns T iff the last step in Path makes Path circular.
(NCPath.PathStep.E*nd PathS*tep) [Function]
This function returns the card at the appropriate end of PathStep, as indicated by the Direction field of the PathStep.
(NCPath.PathStep.PotentialSte*ps PathStep Direct*ion) [Function]
This function computes the possible next links from the link in PathStep.
(NCPath.PathStep.MeetsFSMCardSpecificatio*nP PathStep FSMN*ode) [Function]
This function determines whether or not the card specified *by PathS*tep meets the specification *of FSMN*ode.
(NCPath.App*ly FSMNodeInstance I*tem) [Function]
This function is anticipated to make the general path language easier to implement. It will look at the NCPathFSMNode and it will use APPLY appropriately, else it will perform the right parsing and whatnot and then use apply.
(Copy.NCPathF*SM FSMInsta*nce) [Function]
This function copies an NCPathFSM much faster than COPYALL does. It should save time in NCPath.FSM.IncrementCurrentState.
(Copy.NCPathFSMNo*de FSMNodeInsta*nce) [Function]
This function duplicates an NCPathFSMNode.
(NCPath.Link.ListPotentialSte*ps PreviousLink PreviousDirection Direct*ion) [Function]
This function takes a link, the direction it was followed, and another direction flag, and returns a list of all potential links that may be the next step in the path from the appropriate end of the Link in the. The second Direction flag determines which way the found links are followed.
(NCPath.NoteCard.ListPotentialSte*ps Card Direct*ion) [Function]
This function returns the potential links that might be of use from Card.
(NCPath.Link.GetC*ard PreviousLink Direct*ion) [Function]
Given a link and a flag, this function decides which end of the link (source card or destination card) to retu

NC Path Language Description

<path> ::= <path step>
| (<path step> ...)
| (<path>)
| (<path> <repeater exp>)

<path step> ::= <path step descriptor>
| (<path step op> <path step> ...)

<path step descriptor> ::= <literal descriptor>
| <functional descriptor>
| ANY
| <lisp-variable>

<literal descriptor> ::= <link-type>
| @<card-type>
|_<link-type>
|_@<card-type>

<functional descriptor> ::= #<lisp-link-predicate>
| @#<lisp-card-predicate>
|_#<lisp-link-predicate>
|_@#<lisp-card-predicate>

<path step op> ::= OR | AND | NOT

<repeater exp> ::= <repeater token>
| <repeater token> <depth cutoff>

<repeater token> ::= + | *

<depth cutoff> ::= <positive integer>

<positive integer> ::= <digit> | <digit> <positive integer>

<digit> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0

Notes:
1) The characters #, @, and _ are called the prefix of a path step descriptor, and their order is unimportant.
2) Note that <lisp-variable> `must have` a value parseable as a <path step descriptor>.

NCPathParse

By: David Newman (Newman.pasa@Xerox.COM)

NCPath, NCPathUse

INTRODUCTION

This package parses expressions of a simple path description language into NCPathFSM data structures for use by NCPath.FSM.PathCollect. The language is described by a BNF grammar in the file NCPathLanguage.TEdit. A simpler description of the language follows.

PATH DESCRIPTION LANGUAGE

The basic units of the language are path step descriptors. Each descriptor is either a literal descriptor or a functional descriptor. The literal descriptors consist of link types and card types currently existing in the active notefile. Functional descriptors are the names of lisp predicates that can be applied to a link or card, and which return T or NIL. A proposed new step in a path through a notefile is accepted when it is of the type specified by a literal descriptor or when application of the function specified by a functional descriptor returns T. Each path step descriptor has an optional prefix of several characters. The character @ indicates that the descriptor specifies the card at the end of the path step. The character # prefaces all functional descriptors. The character _ indicates that links followed in obtaining this step should be followed backwards. Path descriptors can be combined into other descriptors by the use of AND, OR, and NOT subject to the restriction that all descriptors in a combination must either apply to cards or links. Lastly, repetitive path structures may be specified using a number or Kleene star notation. See the examples below.

EXAMPLES

FOO is a function name, GOODP is a function name, and BAR is a variable with value @#GOODP.

```
(NCPathParse (QUOTE #FOO))          (NCPathParse (QUOTE @#FOO))
(NCPathParse (QUOTE _@#FOO))        (NCPathParse (QUOTE BAR))
```

```
(NCPathParse (QUOTE @Text))          (NCPathParse (QUOTE SubBox))
(NCPathParse (QUOTE (SubBox)))       (NCPathParse (QUOTE (@#FOO)))
(NCPathParse (QUOTE (SubBox FiledCard)))
(NCPathParse (QUOTE (@#FOO @#FOO))) (NCPathParse (QUOTE (AND @#FOO BAR)))
(NCPathParse (QUOTE (OR @#FOO (AND @Text BAR))))
(NCPathParse (QUOTE (#FOO *3)))      (NCPathParse (QUOTE ((#FOO #FOO) *3)))
```

FUNCTIONS

NCPathParse *Expression*) [Function]
This function is intended to be the top-level function that parses an expression into an FSM that NCPath.FSM.PathCollect can deal with.

(NCPathParse.FSMFromPathSpec *Expression*) [Function]
This function is intended to be the top-level function that parses an expression into an FSM that NCPath.FSM.PathCollect can deal with.

(NCPathParse.Path *Expression*) [Function]
This function parses an individual path as a list of steps or repeater expressions.

(NCPathParse.PathStep *Expression*) [Function]
This function parses an individual step of a path. A step is either a path step descriptor or some combination of descriptors.

(NCPathParse.PathStepDescriptor *Expression*) [Function]
Parses a path step descriptor. A descriptor is a literal descriptor, a functional descriptor, a "don't care" expression, or a variable pointing to a descriptor of one of the other types. The variable is checked for immediate circularity, but not for indirect circularity; either of these conditions could cause an infinite loop.

(NCPathParse.LiteralPathDescription *Expression*) [Function]
This function parses path descriptors that literally describe what kind of object qualifies as a step in the path sought through the notecards network.

(NCPathParse.FunctionalPathDescription *Expression*) [Function]
This function parses path descriptors that give function names which act as predicates to determine whether a link or card matches this step in the path descriptor.

(NCPathParse.CreateFSMNode *Item Link/CardFlag DirectionFlag FunctionFlag*) [Function]
This function creates the NCPathFSMNodes that will become part of the NCPathFSM being created by NCPathParse. It contains a call to the code which creates the LAMBDA expressions to be used as predicates if necessary.

(NCPathParse.CreatePredicateForm *Type Link/CardFLAG*) [Function]

This function creates a LAMBDA expression that will serve as a predicate. See NCPathParse.CombinePredicates.

(NCPathParse.RepeaterExpression *RepeaterExpression LoopSteps*) [Function]
This function parses repeater expressions.

(NCPathParse.RepeaterToken *RepeaterExpression LoopSteps*) [Function]
Creates a repeater loop with no limit, or with infinite limit

(NCPathParse.LimitedRepeaterToken *RepeaterExpression LoopSteps*) [Function]
This function parses repeater tokens that have an integral limit.

(NCPathParse.LoopDecision *Expression LoopSteps MinimumRepeat Symbol*) [Function]
This function decides what kind of loop is to be created (requiring one or zero pases through the loop) and then creates a description of that loop.

(NCPathParse.CreateLoop *LoopSteps MinTimes MaxTimes*) [Function]
Here we create an expression describing a loop that NCPathParse.CoalesceStates can transform into an actual loop.

(NCPathParse.PathStepOperation *Expression*) [Function]
Here we parse combinations of pathsteps. Combinations are created using AND, OR, and NOT, to combine other pathsteps.

(NCPathParse.FunctionP *Function*) [Function]
This function returns T if its argument names an appropriate function for use as a predicate in an NCPathFSMNode.

(NCPathParse.CheckAndComputeFlag *StepList FlagName*) [Function]
This function computes the Direction and Card/Link flags when parsing a combination FSMNode. It also determines if the flags involved match, and notifies the user if they do not. It returns NIL if the flags do not match.

(NCPathParse.CombinePredicates *StepList Operation*) [Function]
This function builds the LAMBDA expression that will be the predicate in a combination FSMNode.

(NCPathParse.CombinationExpressions *StepList Operation*) [Function]
This function combines pathstep specifications using AND, OR, or NOT into a single NCPathFSMNode.

(NCPathParse.CoalesceStates *NodeList*) [Function]
This function takes a lisp-style list of NCPathFSMNodes and turns them into a linked list. The linked list will include circularities where repeater expressions exist.

(NCPathParse.CoalesceRepeaterStates *LoopList Next* Limit) [Function]
This function creates the circular linked list structure that repeater expressions need.

(LOGEQUAL . *U*) [Function]
This function is a logical operator. It determines if the arbitrary number of arguments are logically equal or not.

(NAND . *Args*) [Function]
This function is a logical operator similar to AND or OR, except that it performs the logical operation of `(NOT (AND Args))`.

NCPathUse

By: David Newman (Newman.pasa@Xerox.COM)

NCPath, NCPathParse

INTRODUCTION

This package is intended to provide some functions useful to the programmer using the NCPath package. These functions allow the programmer to transform the data returned by NCPath.FSM.PathCollect into something more useful to a particular application. Also, three functions intended for use as Print definitions for the three major datatypes in NCPath are included in this file.

FUNCTIONS

(NCPath.GetCardPathListsFromPathCollection *PathCollection*) [Function]
This function collects a list of cards traversed by each path in PathCollection. Thus, it returns a list of lists. Each sublist is a list of all the cards traversed in a particular path.

(NCPath.GetTerminalsFromPathCollection *PathCollection*) [Function]
This function returns a list of the terminal cards in a collection of paths. It is intended to assist the user in getting the results desired from what NCPath.FSM.PathCollect returns.

(NCPath.GetUniqueTerminalsFromPathCollection *PathCollection*) [Function]
This function is very similar to NCPath.GetTerminalsFromPathCollection, but it collects only unique terminal cards, where the other function can collect multiple copies of the same terminal card.

(NCPath.ReverseAndSeparatePaths *PathCollection*) [Function]
The results of NCPath.FSM.PathCollect is a list of lists, each sublist being a path in the collection. These paths share cons cells for efficiency, and are stored with the steps in the paths in reverse order. This function accepts the result of NCPath.FSM.PathCollect, and returns a list of paths with their steps in the correct order, and broken apart such that no cons cells are shared between paths.

(PathStepPrint *Instance Stream*) [Function]
This function is intended to be a print definition for the NCPathPathStep datatype.

(PrintFSM *Instance Stream*) [Function]
This function is intended for use as a print definition for the NCPathFSM datatype.

(PrintFSMNode *Instance Stream*) [Function]
This function is to be a print definition function for FSMNodes.

NCScreen

By: Randy Trigg (Trigg.pa@Xerox.COM)

Stored: {qv}<notecards>1.3k>library>NCScreen, .dcom, .ted
Last updated: May 4, 1987.

INTRODUCTION

This NoteCards library package provides a set of user-callable functions that can be used to manipulate cards on the screen. Other NoteCards library packages including NCCluster use NCScreen functions.

FUNCTIONS FOR LAYING OUT CARDS

(SCREEN.LayoutCardsInSquare <StartPos> <Cards>)

Lay out the cards appearing in the list <Cards> in a square arrangement with the upper left corner of the first card at the position <StartPos>. The number of columns is equal to the floor of the square root of the number of cards. Card windows are arranged in row order attempting to fit all windows on the screen without overlapping. However, some windows may overlap in order to fit each window on the screen.

(SCREEN.LayoutCardsInCascade <StartPos> <Cards>)

Lay out the cards in <Cards> in a cascaded overlapping deck with the upper left corner of the first, deepest card at the position <StartPos>. Calls SCREEN.GetCascadePosition.

(SCREEN.GetCascadePosition <Window>)

This function is used to determine the position for the next card in a cascade. It takes one argument which is a window and returns the position of the upper left corner for the next cascaded window. The spacing between cards in the cascade is determined by the global variables SCREEN.CascadeXSpace and SCREEN.CascadeYSpace (initially set to 3 and 8, respectively).

(SCREEN.LayoutCardsInSurround <CenterCard> <Cards>)

Arranges the cards in the list <Cards> around three sides of the card <CenterCard>. Cards are placed in order down the right side, the left side, and along the bottom. Note: this may not display all cards in <Cards>. The number of cards displayed is returned.

(SCREEN.SurroundCardWithFileBoxContents <Card> <Box>)

Arrange the child cards of the filebox <Box> around the card <Card>. If either arg is NIL, then the user is asked to select. This calls SCREEN.LayoutCardsInSurround. If some cards in <Box> could not be displayed, then a statement to that effect is printed in the prompt window.

MISCELLANEOUS HANDY FUNCTIONS

(SCREEN.MoveCardToPos <Card> <Pos> <Corner>)

Move the <Card> card's window such that its <Corner> corner is at the position <Pos>. <Corner> should be one of the litatoms UL, LL, UR, or LR. The default is UL. If the card is not currently on the screen, then it will be brought up. This calls SCREEN.MoveWinToPos.

(SCREEN.MoveWinToPos <Win> <Pos> <Corner>)

Move the <Win> window so that its <Corner> is at the position <Pos>. <Corner> should be one of UL, LL, UR, or LR as above. If the window doesn't completely fit on the screen at that position, then it is moved so it does.

(SCREEN.WinLLCorner <Win>)

(SCREEN.WinLRCorner <Win>)

(SCREEN.WinULCorner <Win>)

(SCREEN.WinURCorner <Win>)

These functions return the positions of the various corners of the window <Win>.

(SCREEN.WinShrinkAndPlace <Win>)

Shrink the window <Win> (if not already shrunk) and place so its upper left corner is at the upper left corner of <Win>.

NCTableTopCard

By: Randy Trigg (Trigg.pa@Xerox.COM)[1]

Stored: {qv}<notecards>1.3L>Library>NCTableTopCard, .lcom, .ted
Written: July 30, 1988 by Randy Trigg
Last updated: May 5, 1989 by Peggy Irish.
See also NCGuidedTourCard package.

[1] The first version of TableTops was written by Jeff Shrager and subsequently modified by George Cole.

INTRODUCTION

Tabletop cards are a means in NoteCards of capturing the layout of some set of cards on the screen. A tabletop is a snapshot which includes the list of cards, the shapes of their windows, their positions on the screen, the scrolled locations (vertically and possibly horizontally) of the windows' contents, and the order in which to open the windows so that the original (possibly) overlapping arrangement can be preserved. Tabletops have three main uses: (1) as a means of storing card layout across notefile closings/openings. In this case, the system constructs and maintains the tabletop automatically. (2) as aids for online demonstrations of notefiles and to help in creating hard copy screen snapshots for offline presentations. Here, the user creates the tabletop card manually. (3) as "stops" along a NoteCards guided tour. In what follows, I describe the system support for the first two cases. For a description of the use of tabletop cards in guided tours, see the documentation on the NCGuidedTourCard package.

LOADING THE NCTABLETOPCARD PACKAGE

Loading the NCTableTopCard package is done in the same way as loading any other NoteCards card type library package. For example, you can type the following at an Interlisp exec:

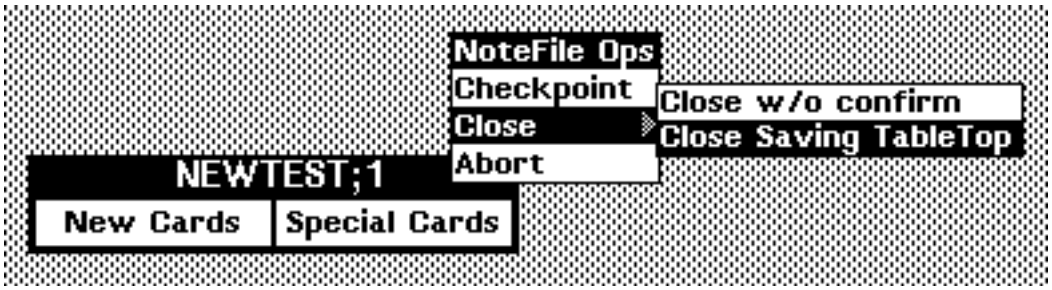
(FILESLOAD (FROM NOTECARDS) NCTABLETOPCARD)

Alternatively, you can add the atom NCTABLETOPCARD to the global var NOTECARDSLIBRARYFILES in your init file. This will ensure that NCTABLETOPCARD is loaded when bringing up a fresh sysout. Finally, if a link to a

tabletop card is followed and the NCTABLETOPCARD package is not currently loaded, then it will be auto-loaded by the system.

SAVING A TABLETOP AT NOTEFILE CLOSING

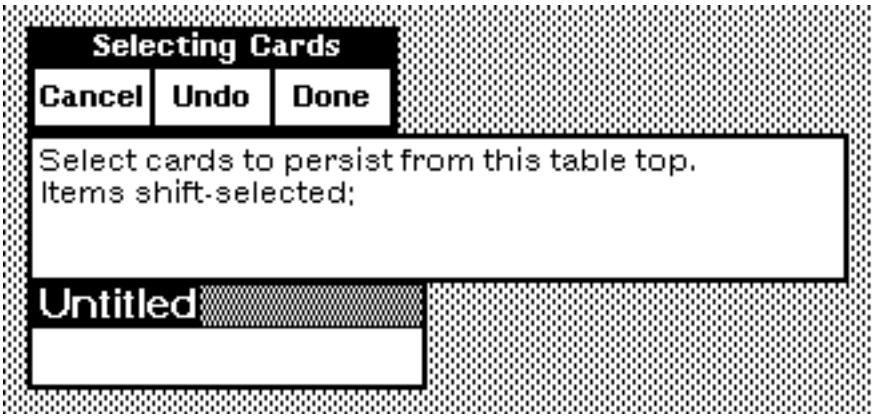
Using a tabletop, one can save the screen layout of cards from a notefile in such a way that the layout can be reinstated the next time the notefile is opened. This allows for greater continuity of work across notefile closings and reopenings. When this package is loaded, the "NoteFile ops" left-button menu available from the notefile icon is augmented slightly. The new pull-across menu for "Close" appears as follows



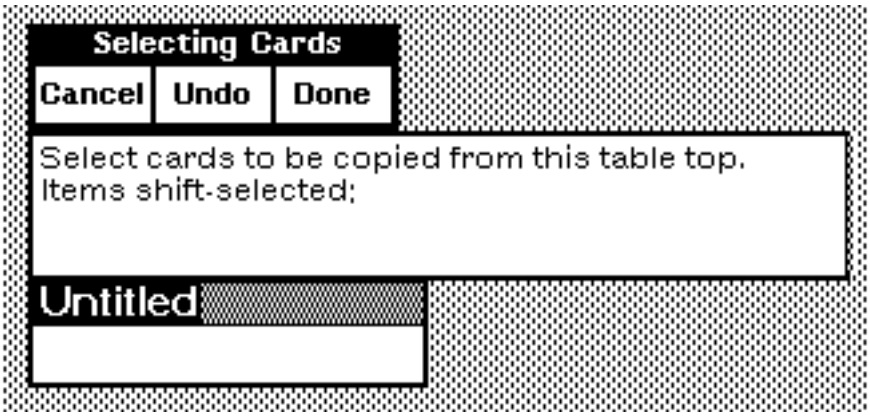
Choosing the option **Close Saving TableTop** causes the current layout of cards with open (or shrunken) windows to be saved in a special tabletop and then the notefile to be closed. The next time the notefile is opened, those same cards will be brought up and arranged (or shrunken) according to the saved layout. (The NCTABLETOPCARD package will be auto-loaded if necessary.)

BUILDING AND MANIPULATING TABLETOPS BY HAND

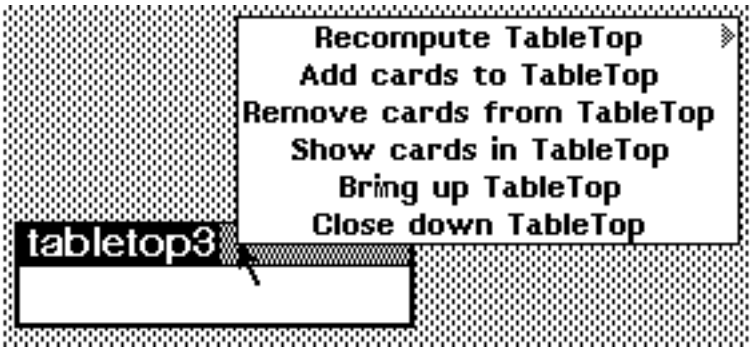
You can create a tabletop card in the usual way, namely by middle-buttoning the NewCards menu on the notefile icon and choosing **TableTop**. You will be asked if you want to use a template tabletop for the creation of this new tabletop. Frequently a user will want several tabletops to share one or more windows, so this option allows one tabletop to be modeled easlily after another. Selecting Yes allows you to pick a tabletop to be used as a template for the new tabletop. Then you will be asked to select the cards to persist from the template, ie. whch cards should actually exist in both tabletops.



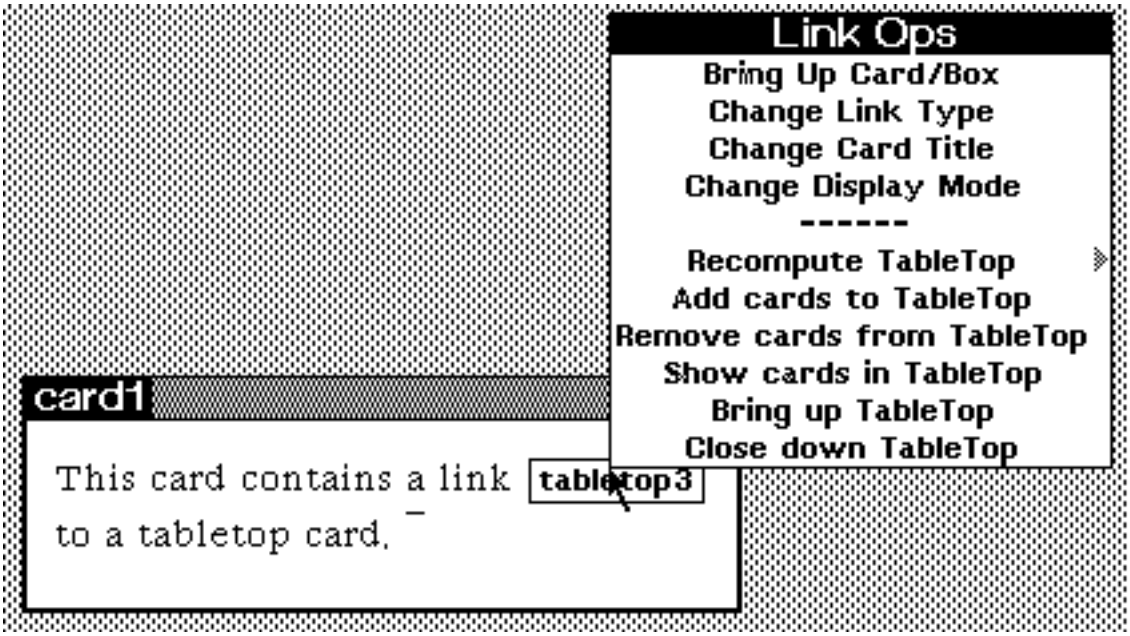
You will also be asked to select the cards to copy from the template. These cards will be copied for the new tabletop, preserving the title, size, shape, location, and default looks of the original cards.



If you don't use a template tabletop, you'll be asked to select a set of cards to form the contents of the tabletop. Select these in the usual way, by holding the shift-key down and left-buttoning in the card windows' titlebars. You'll notice that the tabletop card's window is rather small and has no directly editable substance. Nonetheless, the contents of a tabletop card can be easily modified through a set of tabletop operations available from the card's middle-button titlebar menu.



In addition, each of these operations is available from (the middle-button menu of) any link icon pointing at the tabletop card.



The operations available for tabletop cards are:

Recompute TableTop - Before choosing this operation, you presumably will have rearranged the cards in the tabletop changing some aspect of their layout. Choosing Recompute causes the new layout to be installed in the tabletop card.

Recompute TableTop pull across menu -

New Cards - This operation requires you to specify a new set of cards. That is, the tabletop is recomputed from scratch.

New Cards In Order - This operation requires you to specify a new set of cards, like the **New Cards** menu item. The order in which you choose the new cards is the order in which the cards will appear when you bring up the tabletop.

With Template - This operation asks you to select an existing tabletop to be used as a template for this tabletop.

Add cards to Table Top - This lets you add one or more cards to the tabletop by shift-selecting their windows' titlebars (or links icons pointing at them) in the usual manner. Once the selection is made, the tabletop is recomputed so that the new cards will be ordered correctly relative to the cards already in the tabletop.

Remove cards from Table Top - This brings up a menu containing the titles of all cards in the tabletop. Choose a card to remove by selecting from this menu. A second menu will then pop up containing the remaining cards. In this way, as many cards as desired can be removed from the tabletop. You'll be asked to confirm the removal of all selected cards as soon as you click outside of one of the card title menus.

Show cards in Table Top - This brings up a menu containing the titles of all cards in the tabletop. You can then select from this menu to bring up a single card in the tabletop at its assigned position/shape and scrolled properly.

Bring up Table Top - This opens all cards in the tabletop and positions, shapes, scrolls, shrinks and orders their windows according to the layout information stored in the tabletop. Any cards already open are moved into place and scrolled and shaped (or shrunk) appropriately.

Close down Table Top - This closes any open cards in the tabletop.

The above description covers the standard operation of tabletops. There are, however, two exceptional cases having to do with unopened cards in the tabletop. First, you might try to add a card to the tabletop whose window is not currently open

(or shunken). In this case, the card's most recent position is used and it will appear in the bottom-most position in the window overlap ordering.
Second, you might try to recompute your tabletop even though one or more cards are not currently open. That causes something like the following interactio 

ÓÑ
Of the three choice**s, Can**cel aborts the Recompute operation**. K**eep causes the card to remain in the tabletop. (Note however, that it will be moved to bottom-most in the overlap ordering.**) Rem**ove causes the card to be removed from the tablet**op.**

INFORMATION STORED IN A TABLETOP CARD

As mentioned above, the tabletop card stores layout-related information for each of its cards. This includes: the card window region (i.e. shape and position); the scrolling positions (both vertically and horizontally if any); and whether the card's window is shrunken and if so, the position of its icon. Furthermore, the tabletop keeps the list of cards sorted according to the window overlapping ordering (a.k.a. the TOTOPW ordering). If t**he Recomp**ute sub-co**mmand New Cards In** Order was used to select the cards in the tabletop, then the list of cards is kept in the order specified.
In addition, extra information is kept for certain card types. For browser-based cards, tabletops record information about the shrunken browser overview window (if and where it is attached and its region) and whether the links legend and Editor menu windows are currently open. For Sketch-based cards, tabletops record whether the Sketch editing menu is atta**ched.**

PROGRAMMER'S INTERFACE TO TABLETOP CARDS

To create a new tabletop card programmatically, call NCP.CreateCard as fol**lows:**
(NCP.CreateCard 'TableTop NoteFile Title NoDisplayFlg Props ParentFileBoxes ListOfCards InterestedWindow)
Here, all the parameters are as per the documentation of NCP.CreateCard except that ListOfCards should be a list of (open or shrunken) cards whose layout is to be the tabletop card's initial substance.
The following functions can be used to programmatically manipulate tabletop cards and their cont**ents.**

(NCTableTop.BringUpTableTop TableTopCardOrWindow DoNotUncacheFlg)
The tabletop is brought up, that is, its cards opened and layed out appropriately on the screen. TableTopCardOrWindow can be either a tabletop card object or the window for same. Normally, if the tabletop card itself isn't part of the tabletop being displayed, it is cached, the cards in the tabletop are displayed, and the tabletop card is uncached. A non-NIL DoNotUncacheFlg will prevent the uncaching of the tabletop card (this is useful, for example, when control of caching is desired at the Guided Tour level -- see the NCGuidedTour package documentation).
NCTableTop.BringUpTableTop returns the list of cards it brough**t up.**

(NCTableTop.OpenCardInTableTop TableTopCardOrWindow InterestedWindow)
This causes a single card within a tabletop to be opened and positioned, shaped, etc.

at its proper location. A pop-up menu is used to elicit the card to be opened from the **user.**

(NCTableTop.CloseDownTableTop TableTopCardOrWindow)
Close any open cards in the tabl**etop.**

(NCTableTop.RemakeTableTop TableTopCardOrWindow ListOfCards RetainClosedCardsFlg NoToTopOrder)
Rebuild the tabletop to contain the cards appearing **in ListO**fCards**. If RetainClosedCa**rdsFlg is non-nil, then even closed car**ds in ListO**fCards will be included in the tabletop**. If NoToTo**pOrder is non-NIL, the cards are ordered in the tabletop according to the order in which they appe**ar in ListO**fCards, instead of by their overlapping order on the screen, which is the def**ault.**

(NCTableTop.RecomputeTableTop TableTopCardOrWindow NoToTopOrder)
Recompute the tabletop contents. Normally, the cards in the tabletop are open and arranged in a new arrangement when this call is made**. If NoToTo**pOrder is non-NIL, the order of the cards in the tabletop is not altered according to their overlapping order on the screen, which is the def**ault.**

(NCTableTop.RemoveCardsFromTableTop TableTopCardOrWindow CardsToRemove)
Remove cards in the **list Card**sToRemove from the tabletop's substance. Returns the cards actually **removed.**

(NCTableTop.AddCardsToTableTop TableTopCardOrWindow CardsToAdd InterestedWindow)
Add cards appe**aring in C**ardsToAdd to the substance of the t**abletop.**

(NCTableTop.CardsInTableTop Table TopCard)
Return the list of cards currently appearing in the t**abletop.**

(NCTableTop.TableTopBasedP CardOrCardType)
Return no**n-nil if CardO**rCardType is either a tabletop card or a card of a type that inherits from tabletop card. (It does the obvious **thing if CardO**rCardType is a car**d type.)**

(NCTableTop.CollectCards RootTableTopCards LinkTypes MaxDepth FollowCrossFileLinksFlg)
This function is analogous to NCP.CollectCards. It starts from a list of tabletop cards **given by RootTabl**eTopCards and computes the transitive closure by following links of types appe**aring on** LinkTypes to a maximum depth spec**ified by** MaxDepth where the depth 1 cards are the cards directly in the tabletop su

TextCardKeys

by Kirk Kelley (Kelley.pa@Xerox.COM)

{QV}<NoteCards>1.3k>library>TextCardKeys.dcom
.tedit
Last updated: Jan 29, 1988

Introduction

Ever wish you could capture the structure of your ideas before they escaped short-term memory?

TextCardKeys sets special keys on your keyboard to create with one chord a new text card linked from your caret location.

Initial Settings

For a See link type: META asterisk (META SHIFT 9)
For an Include link type: META open paren (META SHIFT 9)

To close a card: META close paren (META SHIFT 0)

The new cards come up with default titles ready to be edited.

For See links, the title is an asterisk.
For Include links, the title is empty.

The new cards also come up already positioned on the screen.

The best way to see how it works is to load it and try it out. I use this with ForcedFiling turned off and find my hands rarely need to leave the keyboard.

Philosophy

TextCardKeys facilitate one of the proposed new ways of keeping track of cards. In this new way cards would always be linked somewhere. It is hoped this will eliminate the need for forced filing, Orphans, the To Be Filed box, un-filed cards, and all of the concepts and mechanisms surrounding them.

Changing Settings

The following is the nitty gritty for those that want to program different behavior or settings.

NC.CreateLinkedTextNoteChar [variable]
This is the key for creating a "See" link. It comes set to 298 (Meta asterisk).

NC.CreateLinkedTextNoteLinkType [variable]
This is a string containing the type of link for the above character. It comes set to "See".

NC.CreateLinkedTextNoteTitle [variable]
This is a string containing the initial title for the above link. It comes set to "".

NC.CreateLinkedTextInclusionChar [variable]
This is the key for creating an "include" link. It comes set to 296 (Meta open-paren).

NC.NC.CreateLinkedTextInclusionLinkType [variable]

This is a string containing the type of link for the above character. It comes set to "Include".

NC.CreateLinkedTextNoteTitle [variable]
This is a string containing the initial title for the above link. It comes set to "".

NCP.NewTextCardRegion [function]
Alter this to change where, and what shape, cards will automatically have when positioned on the screen. To turn off auto-positioning, have this return NIL.

If you change any of the above settings, I would be interested to know how you change them