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>)

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

(NCP.SameLinkP <link1> <link2>)

Returns non-nil if <link1> is the same link as <link2>. Error if either arg is not a valid link.

(NCP.AllLinks <NoteFile>)

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>)

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>)

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

The following functions allow the user to manipulate link labels.

(NCP.CreateLinkType <linkType> <NoteFile> <QuietFlg>)

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>)

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>)

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>)

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.AddDefaulItNoteFileIconMiddleButtonItems <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]