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.