The NoteCards Types Mechanism

Release 1.2

Frank G. Halasz

Xerox PARC

First Written: 22-Mar-85

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

Modified: 1-Aug-85 by Lissa Monty

1. Introduction

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

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

1.1 The Inheritance Hierarchy

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

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

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

Functions are inherited all or none. Often, however, a new NoteCardType will require only a minor addition to the corresponding function of its super-type. In this case, the new card type should define a new function, but this function can call the corresponding function of its super-type to do the bulk of the work. The following construction will accomplish this goal:

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

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

```
(DEFINEQ

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

1.2 Links and Link Icons

An integral part of NoteCards is the ability to create a link between two note cards. Presently, there are two kinds of links: *GlobalToGlobal* links and *LocalToGlobal* links. GlobalToGlobal links connect one entire card with another entire card and are stored separately from either card's substance.

GlobalToGlobal links are maintained (almost) entirely by the NoteCards system code and therefore do not vary across note card types.

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

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

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

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

1.3 Using the Types Mechanism

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

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

2. The NoteCardType

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

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

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

Inheritance Hierarchy Specifications

1) TypeName
2) SuperType
3) SubstanceType

Functions

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

Parameters

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

These fields are defined as follows:

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

2. SuperType: The TypeName of the NoteCardType that is the super-type for this NoteCardType. When a new NoteCardType is created, its SuperType must be an existing NoteCardType.

3. SubstanceType: The SubstanceName for the substance of this card type. When a new card is created, its SubstanceType must be the name of an existing SubstanceType (see Section 3.0 below). The basic NoteCards system includes the following substance types: TEXT, SKETCH, GRAPH which represent the the substances handled by the TEdit, Sketch, and Grapher packages respectively.

4. MakeCardFn: The name of a function to be applied to an ID, a Title, and a NoDisplayFlg. The function should create a new card of this type. The ID is the note card ID that will be assigned to the newly created card. It should be used to set the various properties of the new card. The title is a string specifying the title of the new card. It can be used in messages to the user or to set the title of any windows created. NoDisplayFlg determines whether the new card is to be displayed on the screen or not. If NoDisplayFlg is non-NIL, then the card is to be displayed in a window on the screen. If NoDisplayFlg is NIL, then the card is to be created but not displayed on the screen.

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

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

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

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

The EditCardFn should return the editor window.

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

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

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

The value returned by QuitCardFn is unspecified.

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

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

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

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

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

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

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

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

The value returned by the PutCardFn is unspecified.

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

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

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

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

The value returned by the CopyCardFn is unspecified.

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

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

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

The value returned by the MarkCardDirtyFn is unspecified.

The card specified by ID is guaranteed to be active.

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

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

The card specified by ID is guaranteed to be active.

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

> ReturnLinkIconsFlg and ReturnLocationsFlg both NIL: the list to be returned should be a list of link records.
>
> ReturnLinkIconsFlg is non-NIL, ReturnLocationsFlg is NIL: the list to be returned should be a list of link icons.
>
> ReturnLinkIconsFlg is NIL, ReturnLocationsFlg is non-NIL: the list to be returned should be a list of pairs where the first memeber of the pair is a link record and the second member of the pair is the "location" of the link icon for that link inside the substance.
>
> ReturnLinkIconsFlg and ReturnLocationsFlg both non-NIL: the list to be returned should be a list of pairs where the first memeber of the pair is a link icon and the second member of the pair is the "location" of that link icon.

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

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

The card specified by ID is guaranteed to be active.

13. DeleteLinksInCardFn: The name of a function to be applied to a "source" note card ID and a link record or "destination" note card ID. If the second argument is a link, the function should

remove from the substance of the card specified by "source" ID the link icon corresponding to link. If the second argument is a "destination" note card ID, the function should remove from the substance of the card specified by "source" ID all link icons corresponding to links pointing to the card specififed by "destination" ID.

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

The value returned by the DeleteLinksInCardFn is unspecified.

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

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

The value returned by the UpdateLinkIconsInCardFn is unspecified.

The "source" card is guaranteed to be active.

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

The value returned by the InsertLinkInCardFn is unspecified.

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

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

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

display the type of the link. If ATTACHBITMAPFLG is non-NIL, a bit map describing the type of the destination card will be attached to the right of the link icon.

Note: This property in NOT inherited.

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

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

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

Note: This property in NOT inherited.

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

3. The SubstanceType

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

1) SubstanceName
2) CreateSubstanceFn
3) EditSubstanceFn
4) QuitSubstanceFn
5) GetSubstanceFn
6) PutSubstanceFn
7) CopySubstanceFn
8) MarkSubstanceDirtyFn
9) SubstanceDirtyPFn
10) CollectLinksInSubstanceFn
11) DeleteLinksInSubstanceFn
12) UpdateLinkIconsInSubstanceFn
13) InsertLinkInSubstanceFn

14) TranslateWindowPositionToSubstancePositionFn

15) SubstanceDefaultWidth

16) SubstanceDefaultHeight

17) SubstanceLinkAnchorModesSupported

These fields are defined as follows:

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

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

15 Thru 17. Parameters: The parameters are identical to the corresponding parameters in the NoteCardType data structure. There are no parameters for the LinkDisplayMode and the DisplayInMenuFlg because these two parameters are not inherited. They must be specified separately for each card type.

4. Adding a New NoteCardType or SubstanceType to the System

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

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

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

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

5. Example: Defining the ProtectedText NoteCardType

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

· The function that creates the new ProtectedText card type:

```
(NC.AddProtectedTextCardType
  (LAMBDA NIL                          (* fgh: "26-Mar-85 15:48")
          (* * Create the ProtectedText card type)
    (NCP.CreateCardType (QUOTE ProtectedText)
                        (QUOTE Text)
                        (QUOTE TEXT)
                        (QUOTE ((MakeCardFn NC.MakeProtectedTextCard)
                                (EditCardFn NC.EditProtectedTextCard)))
                        (QUOTE ((LinkDisplayMode (T NIL NIL))
                                (CardDefaultHeight 300)
                                (CardDefaultWidth 400)
                                (CardLinkAnchorModesSupported T)
                                (CardDisplayInMenuFlg T))))))
```

· The MakeCardFn for the ProtectedText card type:

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

· The EditCardFn for the ProtectedText card type:

```
(NC.EditProtectedTextCard
  (LAMBDA (ID Substance Region/Position)
                                    (* fgh: "26-Mar-85 17:21")
          (* * Edit a Protected Text card, asking for the password first.)
    (PROG (ExactRegion Window Password Result)
          (* * Open a window for this card)
          (SETQ ExactRegion (NC.DetermineDisplayRegion ID
                                          Region/Position))
          (SETQ Window (CREATEW ExactRegion))
          (* * Get this card's password from the prop list)
          (SETQ Password (NCP.CardProp ID (QUOTE Password)))
          (COND
            ((EQUAL Password (NC.GetPassword Window))
              (* Password is okay.
                 Call the EditCardFn of my super-type)
              (SETQ Result (APPLY*
                  (NCP.CardTypeInheritedField
                    (NCP.CardTypeSuper (QUOTE ProtectedText))
                    (QUOTE EditCardFn))
                  ID Substance ExactRegion)))
            (T (* Password is bad. Express condolences)
               (NCP.PrintMsg Window T "Sorry." (CHARACTER
                              13)
                             "You do not know the password!!"
                             (CHARACTER 13)
                             "Bye."
                             (CHARACTER 13))
               (DISMISS 2000)))
          (* * Close the window you created.
               The super-types EdityCardFn will
               have created another window.)
          (CLOSEW Window)
          (RETURN Result))))
```

· A utility used by the MakeCardFn and the EditCardFn:

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