16B

MORE ON FILES — NAVIGATION SERVICES

Includes Demonstration Program Files2

Introduction

Navigation Services, which was introduced with Mac OS 8.5¹ as an alternative to, and ultimately as a replacement for, the Standard File Package described at Chapter 16, provides greatly enhanced functionality in the area of document management. Navigation Services provides for opening and saving documents, confirming saves and discarding changes, choosing a volume, folder, file, or file object, creating a new folder, file format translation, and easier navigation.

The dialog and alert boxes created by Navigation Services are all Appearance-compliant. Provided that your application provides an event-handling callback function, the primary dialog boxes are resizable and movable and the alert boxes are movable.

As is the case with the Standard File Package dialogs, the primary Navigation Services dialog boxes may be customised.

Navigation Services Dialog and Alert Boxes

The primary dialog boxes created by Navigation Services are as follows:

- Open.
- Save.
- Choose a folder.
- Choose a volume.
- Choose a file.
- Choose a file object.
- Create a new folder.

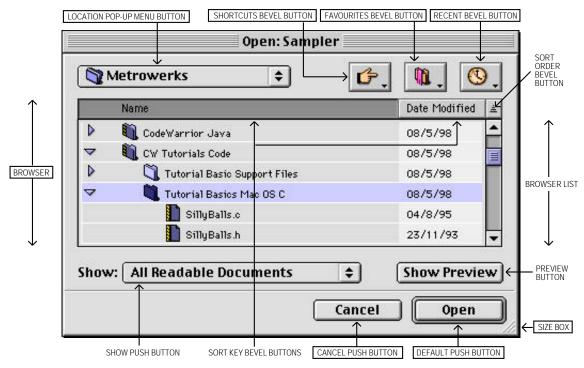
¹ Although introduced with Mac OS 8.5, Navigation Services may be used with Mac OS 7.5.5 or later provided Appearance Manager 1.0.1 or later is present. QuickTime is also required if previews of graphic documents are to be created and viewed. Macintosh Easy Open is required for document translation and the correct display of file types. On 680x0 systems, Navigation Services requires the CFM-68K Runtime Enabler.

The alert boxes created by Navigation Services are as follows:

- Save changes.
- Discard changes.

Standard User Interface Elements in Dialog Boxes

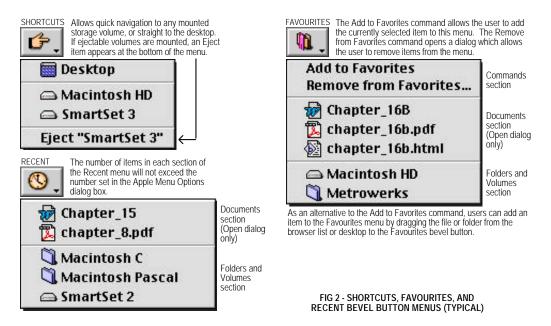
The standard user interface elements in Navigation Services primary dialog boxes are shown at Fig 1.



Note: The Open dialog box is shown. The names of the user interface elements common to all primary Navigation Services dialog boxes appear in boxes.

FIG 1 - STANDARD USER INTERFACE ELEMENTS IN NAVIGATION SERVICES DIALOG BOXES

The menus associated with the standard user interface elements known as the Shortcut, Favourites, and Recent bevel buttons are shown in more detail at Fig 2.



16B-2 Files

Moving and Resizing Navigation Services Dialog Boxes

Navigation Services primary dialog boxes are movable and resizable only if your application provides an application-defined event-handling callback function to handle update events. A universal procedure pointer to your event-handling function is passed in the eventProc parameter of Navigation Services functions such as NavGetFile (see below).

Browser List Expansion

When the user resizes the dialog box using the size box, the browser list expands proportionately. The date format in the browser list changes as the browser list expands.

Sort Keys and Sort Order

Clicking on one of the **sort key** bevel buttons (Name or Date) causes the browser list to be sorted on that key. The sort order (ascending or descending) may be toggled by clicking the **sort order** bevel button. Navigation Services tracks the sort key and sort order for each application.

Preview Area

Navigation Services provides a **preview area** in all dialog boxes which open files. This area can be toggled on or off by the user using the Show/Hide Preview push button. If the preview area is visible, Navigation Services will automatically display a preview of any file that contains a valid 'pnot' resource. You can request preview display by ensuring that the kNavAllowPreviews constant is set in the dialogOptionFlags field of the NavDialogOptions structure (see below).

Persistence

Persistence is the ability of Navigation Services to store information, and to store it on a per-application basis. For example, when a primary dialog box is displayed, the browser defaults to the directory location that was in use when that particular dialog box was last closed by the application. In addition, if a file or folder was selected when the dialog box was last closed, that file or folder is automatically selected when the dialog is re-opened. The size, position, sort key and sort order of dialog boxes are also memorised for each application.

Default Location and Selection

If you pass NULL in the default Location parameter of functions such as NavGetFile (see below), Navigation Services will display the **default location and selection**. You can override the default location and selection of any Navigation Services dialog box by passing a pointer to an Apple event descriptor (AEDesc) for the new location in the default Location parameter.

Opening Files

The function NavGetFile displays an Open dialog box that prompts the user to select a file (or files) to open.

Opening Multiple Files

By Shift-clicking in the Open dialog box's browser list, or by choosing the Select All item in the application's Edit menu, the user can open multiple files.²

² Folders and volumes cannot be multiple-selected.

The Show Pop-up Menu

The Show pop-up menu in the Open dialog box allows the user to choose the **file types** to be displayed by the browser list and opened by Navigation Services. The list of available file types is built from information supplied by your application when it calls NavGetFile, and by services in the Translation Manager. (See Fig 3.)

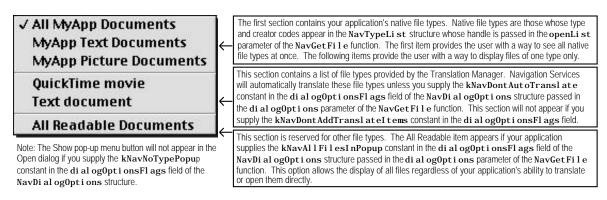
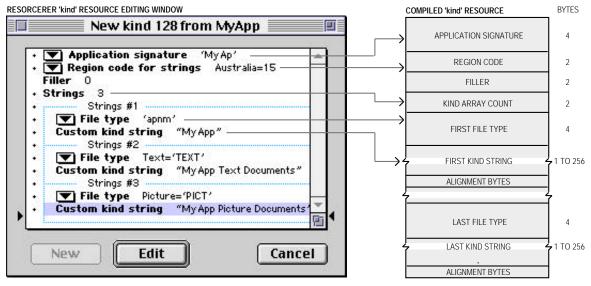


FIG 3 - THE SHOW POP-UP MENU AND FILE TYPE OPTIONS

Native File Types Section

The first item in the native file types section of the Show pop-up menu defaults to All Readable Files if you do not assign the name of your application to the clientName field of the NavDialogOptions structure (see below) passed in the dialogOptions parameter of the NavGetFile function.

The remaining items in the native file types section will default to <Application Name> Document unless you provide kind strings to describe the file types included in your NavTypeLi st structure. You can do this by including a kind resource (a resource of type 'kind') in your application's resource fork. Fig 4 shows the structure of a compiled 'kind' resource and such a resource being created using Resorcerer.³



Note: The special file type 'apnm' has been included so that, whenever Navigation Services encounters a document that belongs to your application, but whose file type has not been included in the 'kind' resource, a kind string in the form "<application name> document" will be generated.

FIG 4 - STRUCTURE OF A COMPILED 'kind' RESOURCE AND CREATING A 'kind' RESOURCE USING RESORCERER

16B-4 Files

 $^{^3}$ The kind strings from your application's 'kind' resource also appear in the Kind column in Finder window list views.

The NavDialogOptions and NavTypeList Structures

Fig 3 referred to the NavTypeList structure, which defines a list of file types that your application is capable of opening, and the NavDialogOptions structure, which contains dialog box configuration settings.

The NavTypeList Structure

The NavTypeLi st structure is as follows:

You can create your file type list dynamically or you can use an 'open' resource. Fig 5 shows the structure of a compiled 'open' resource and such a resource being created using Resorcerer.

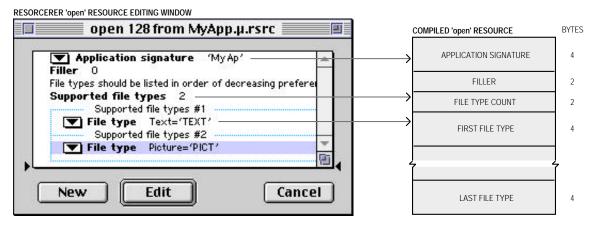


FIG 5 - STRUCTURE OF A COMPILED 'open' RESOURCE AND CREATING AN 'open' RESOURCE USING RESORCERER

The NavDialogOptions Structure

The NavDi al ogOptions structure is as follows:

```
struct NavDialogOptions
                           versi on;
  NavDi al ogOpti onFl ags
                           di al og0pti onFl ags;
  Poi nt
                           location;
  Str255
                           clientName;
  Str255
                           windowTitle:
  Str255
                           actionButtonLabel;
  Str255
                           cancel ButtonLabel;
  Str255
                           savedFileName;
  Str255
                           message;
  UInt32
                           preferenceKey;
  Handl e
                           popupExtension;
};
```

Field descriptions

dialogOptionsFlags One of the following constants of type NavDialogOptionFlags:

	Constant	Description	
	kNavDefaultNavDlogOptions	Use default options. The defaults are as follows:	
		No custom control titles.	
		No banner or prompt message. Automotive and allieurs for allieurs.	
		Automatic resolution of aliases. Support for file provious.	
		Support for file previews. No invisible file chiests are displayed.	
		No invisible file objects are displayed.Support for stationery.	
	kNavNoTypePopup	Don't show file type pop-up.	
	kNavDontAutoTranslate	Don't auto-translate on Open.	
	kNavDontAddTranslateItems	Don't add translation choices.	
	kNavAl l Fi l esI nPopup	Add "All Files" menu item.	
	kNavAllowStationery	Allow stationery files.	
	kNavAllowPreviews	Allow previews.	
	kNavAllowMultipleFiles	Allow multiple selection.	
	kNavAllowInvi si bleFiles	Show invisible objects.	
	kNavDontResol veAliases	Don't resolve aliases.	
	kNavSelectDefaultLocation	Make default location the browser selection.	
	kNavSelectAllReadableItem	Make All Readable Items default selection.	
location	di al og0pti onFl ags field is NULL or box appears in the same location a dialog box is persistent, but defau	dialog box, in global coordinates. If the the coordinate value is (-1, -1), then the dialog s when last closed. The size and location of the lts to opening in the middle of the main screen opened at the persistent location and size.	
clientName	A string that identifies your applic	ation in the dialog box window title.	
wi ndowTi tle	A string that you can provide to ov	verride the default window title.	
actionButtonLabel		dialog box's action push button. If you do not will use the default label (Open or Save, for	
cancel ButtonLabel	An alternative button title for the C	Cancel push button in dialog boxes.	
savedFileName	The default file name for a saved fi	ile.	
message		npt, below the browser list. This message can ctions for the user. If you don't provide a ll expand to fill that area.	
preferenceKey	An application-defined value that identifies which set of dialog box preferences Navigation Services should use. If your application maintains multiple sets of preferences for a particular type of dialog box, you can determine which set is active by specifying the appropriate value in the preferenceKey field. For example, an application may allow one set of preferences when it calls the function NavGetFile to open text files and a different set of preferences when opening movie files. If you do not wish to provide a preference key, specify zero for the preferenceKey value.		
popupExtensi on	menu items to the Show pop-up m up menu in Save dialog boxes.	res of type NavMenuItemSpec used to add extra nenu in an Open dialog box or the Format pop- Using NavMenuItemSpec structures allows your nment types to be opened or saved, or different out line breaks, for example).	

16B-6 Files

The function NavGetDefaultDialogOptions may be called to initialise a structure of type NavDialogOptions with the default dialog box options.

The NavReplyRecord Structure

The second parameter in the NavGetFile function, and in other Navigation Services functions, is a pointer to a structure of type NavReplyRecord. Navigation Services uses this structure to provide your application with information about the user's actions. The NavReplyRecord structure is as follows:

```
struct NavReplyRecord
  UInt16
                         version;
  Bool ean
                         validRecord;
  Bool ean
                         replacing;
  Bool ean
                         isStationery;
  Bool ean
                         translationNeeded;
  AEDescList
                         selection:
  Scri pt Code
                         keyScript;
  FileTranslationSpec
                         **fileTranslation;
```

Field Descriptions

val i dRecord

true if the user closes a dialog box by pressing Return or Enter, or by pressing the default button in an Open or Save dialog box. If this field is false, all other fields are unused and do not contain valid data.

replacing

true if the user chooses to save a file by replacing an existing file (thereby necessitating the removal or renaming of the existing file).

isStationery

A Boolean value informing your application that the file about to be saved should be saved as a stationery document.

translationNeeded

A Boolean value indicating that translation was or will be needed for files selected in Open and Save dialog boxes.

0

An Apple event descriptor list (AEDescList) created from FSSpec references to items selected through the dialog box. Navigation Services creates this list, which is automatically disposed of when your application calls the function NavDisposeReply (see below). You can determine the number of items in the list by calling the Apple Event Manager function AECountItems. (Some dialog boxes may return one or more items; a Save dialog box will always return one.) Each selected item is described in an AEDesc structure by the descriptor type typeFSS. You can coerce this descriptor into an FSSpec structure to perform operations such as opening the file.

keyScri pt

The keyboard script system used for the filename.

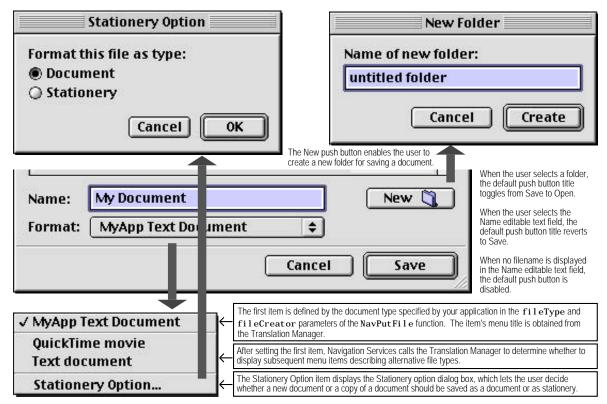
fileTranslation

A handle to a FileTranslationSpec structure. This structure contains a corresponding translation array for each file reference returned in the selection field. When opening files, Navigation Services will perform the translation automatically unless you set the kNavDontAutoTranslate flag in the dialogOptionFlags field of the NavDialogOptions structure. When Navigation Services performs an automatic translation, the FileTranslationSpec structure is strictly for the Translation Manager's use. If you turn off automatic translation, your application may use the FileTranslationSpec structure for its own translation scheme. If the user chooses a translation for a saved file, the FileTranslationSpec structure contains a single translation reference for the saved file and the translationNeeded field of the NavReplyRecord structure is set to true. The handle to the FileTranslationSpec structure is locked, so you can safely use dereferenced pointers.

When your application has finished using this structure, it should dispose of it by calling the function NavDi sposeRepl y.

Saving Files

The function NavPutFile displays the Save dialog box (see Fig 6).



Note: The Format pop-up menu button will not appear in the Save dialog if you supply the kNavNoTypePopup constant in the dialog0ptionsFlags field of the NavDialog0ptions Structure.

FIG 6 - THE SAVE DIALOG BOX (PARTIAL). THE FORMAT POP-UP MENU. THE STATIONERY OPTION DIALOG BOX. AND THE NEW FOLDER DIALOG BOX

You should always call the function NavCompleteSave to complete any save operation. Amongst other things, NavCompleteSave performs any needed translation.

Translating Files on Save

As stated at Fig 6, your application supplies its default file type and creator for saved files to the function NavPutFile and Navigation Services uses this information to build a pop-up menu of available translation choices obtained from the Translation Manager.

If the user selects an output file type that is different from the native type, Navigation Services prepares a translation specification and supplies a handle to it in the fileTranslation field of a NavReplyRecord structure. If you choose to provide your own translation, Navigation Services informs you that translation is required by setting the translationNeeded field of the NavReplyRecord structure to true.

If you wish to turn off automatic translation, set the value of the translationNeeded field of the NavReplyRecord structure to false before calling the NavCompleteSave function.

By default, the NavPutFile function saves translations as a copy of the original file. Your application can direct Navigation Services to replace the original with the translation by passing the kNavTranslateInPlace constant in the howToTranslate parameter of the NavCompleteSave function.

16B-8 Files

The function NavChooseFolder displays a dialog box that prompts the user to choose a folder (see Fig 7.)

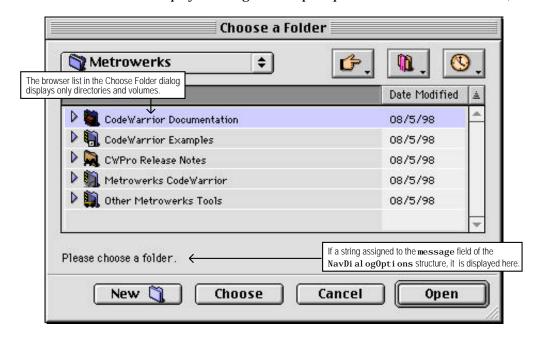


FIG 7 - CHOOSE A FOLDER DIALOG BOX

Several Navigation Services functions return Apple event descriptors for file objects. When Navigation Services passes your application an AEDesc structure of type typeFSS describing a file, the name field will contain the file's name and the parID field will contain the directory ID of the file's parent directory. However, when Navigation Services passes your application an AEDesc structure of type typeFSS describing a directory, the name field is empty and the parID field contains the directory ID of that directory.⁴

Choosing Volumes, Files, File Objects, and Creating a New Folder

The function NavChooseVolume displays a dialog box that prompts the user to choose a volume.

The function NavChooseFile displays a dialog box that prompts the user to choose a file. This file can be a preferences file, dictionary, or other specialised file.

The function NavChooseObject displays a dialog box that prompts the user to choose a file object. This function is useful when you need the user to select an object which might be one of several different types.

The function NavNewFolder displays a dialog box that prompts the user to create a new folder.

As with the Choose Folder dialog box, these dialog boxes will display a string assigned to the message field of the NavDi al ogOpti ons structure immediately below the browser list.

⁴ This means, incidentally, that you can use the name field to determine whether an object is a file or a folder.

Save Changes and Discard Changes Alert Boxes

Save Changes Alert Box

To display a standard Save Changes alert box, your application passes the document title to the function NavAskSaveChanges, which creates an alert box similar to that shown at Fig 8.

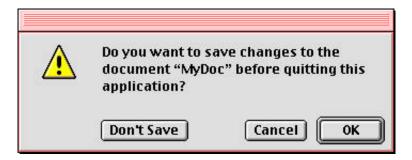


FIG 8 - STANDARD SAVE CHANGES ALERT BOX (QUITTING APPLICATION)

One of the following constants is passed in the action parameter of the NavAskSaveChanges function:

```
kNavSaveChangesClosingDocument = 1
kNavSaveChangesQuittingApplication = 2
kNavSaveChangesOther = 0
```

After the user closes the alert box, Navigation Services tells your application which push button the user clicked by returning one of the following constants:

```
kNavAskSaveChangesSave = 1
kNavAskSaveChangesCancel = 2
kNavAskSaveChangesDontSave = 3
```

You can display a customised Save Changes alert box using the function NavCustomAskSaveChanges. A typical customised Save Changes alert box might contain text such as "You have not saved your work for ten minutes. Do you want to save now?". The message field of the NavDialogOptions structure passed in the dialogOptions parameter of NavCustomAskSaveChanges is the only one you must supply with a value.

Save Changes alert boxes are movable only if your application provides an application-defined event-handling callback function to handle update events.

Discard Changes Alert Box

To support a Revert To Saved item in your application's File menu, Navigation Services provides the Discard Changes alert box (see Fig 9), which is created by the function NavAskDi scardChanges.

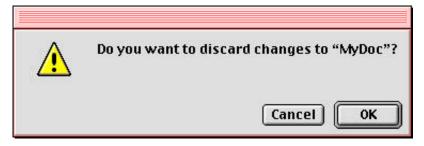


FIG 9 - DISCARD CHANGES ALERT BOX

16B-10 Files

After the user closes the alert box, Navigation Services tells your application which button the user clicked by returning one of the following constants:

```
kNavAskDiscardChanges = 1
kNavAskDiscardChangesCancel = 2
```

Discard Changes alert boxes are movable only if your application provides an application-defined event-handling callback function to handle update events.

Application-Defined Functions

Application-Defined Event Handling

As previously stated, Navigation Services primary dialog boxes are movable and resizable, and Navigation Services alert boxes are movable, only if your application provides an application-defined event-handling function to handle update events. A universal procedure pointer to your event-handling function is passed in the eventProc parameter of Navigation Services functions such as NavGetFile and NavAskSaveChanges. Your event-handling function must be defined like this:

```
pascal Boolean myEventFunction(NavEventCallbackMessage callBackSelector,
NavCbRecPtr callBackParms,
NavCallBackUserData callBackUD);
```

The formal parameter callBackSelector receives an event message constant indicating which type of call Navigation Services is making to the event-handling function. One such constant is kNavCBEvent, which indicates that an event has occurred, and which is the only message that needs to be processed by applications that do not customise the Open and Save dialog boxes. callBackParms is a pointer to a structure of type NavCBRec. The event's event structure resides in the eventData field of the NavCBRec structure.

Application-Defined Object Filtering

The process of choosing which files, folders and volumes to display in the browser list and the ShortCuts, Favourites, and Recent menus is known as object filtering. If your application needs simple, straightforward object filtering, and as previously described, you simply pass a pointer to a structure of type NavTypeList to the relevant Navigation Services function. If you desire more specific filtering, Navigation Services lets you implement an application-defined filter function. Filter functions give you more control over what can and cannot be displayed.

You can use both an <code>NavTypeList</code> structure and a filter function if you wish, but be aware that your filter function is directly affected by the <code>NavTypeList</code> structure. For example, if the <code>NavTypeList</code> structure contains only TEXT and PICT types, only TEXT and PICT files will be passed into your filter function.

You pass a universal procedure pointer to your filter function in the filterProc parameter of functions such as NavGetFile. Your filter function should return true if an object is to be displayed.

The following is an example of a simple filter function:

Application-Defined Previews

To override how previews are drawn and handled, you can create a preview function and pass a universal procedure pointer to it in the previewProc parameter of Navigation Services functions such as NavGetFile. Your preview function must be defined like this:

 $pascal\ Boolean\ my Preview Proc(Nav CBRecPtr\ call Back Parms, Nav Call Back User Data\ call Back UD);$

callBackParms A pointer to a NavCBRec structure that contains event data needed for your

function to draw the preview.

A value set by your application. When the NavGetFile function calls back your

event-handling function, the call BackUD value is passed back to your application.

Return: true if your preview function successfully draws the file preview. If your preview function returns false, Navigation Services will display the preview if the file contains a valid 'pnot' resource.

Your application can use the function <code>NavCustomControl</code> to determine if the preview area is visible and, if so, what its dimensions are.

Adding Controls to a Navigation Services Dialog Box

To add controls to a Navigation Services dialog box, you should:

- Provide an event-handling function to communicate with Navigation Services.
- Within the event-handling function, respond to the kNavCBCustomize event message constant, which your application can obtain from the param field of the NavCBRec structure. (See Application-Defined Event Handling, above.) The customRect field of the NavCBRec structure defines a rectangle in the local coordinates of the window. The top-left coordinates of this rectangle define the anchor point for the customisation rectangle, which is the area Navigation Services provides for your application to add custom dialog items. Your application responds by passing the values which will complete the dimensions of your required customisation rectangle. Navigation Services inspects the customRect field to determine if the requested dimensions can be accommodated in the screen space available. If not, the rectangle will be set to the largest size that can be accommodated and your application will be sent another kNavCBCustomize constant. Your application can continue to negotiate by examining the customRect field and requesting a different size until Navigation Services provides an acceptable rectangle value. The minimum dimensions for the customisation area are 400 pixels wide by 40 pixels high.⁵
- With the customisation rectangle established, check for the kNavCBStart event message constant in the param field of the NavCBRec structure. kNavCBStart indicates that Navigation Services is opening the dialog box. After you obtain this constant, you can add your interface elements to the customisation rectangle. You can do this by providing a 'DITL' resource (in local coordinates, relative to the anchor point of the customisation rectangle) and passing the kNavCtlAddControlList constant in the selector parameter of the function NavCustomControl. The following shows one way to do this:

```
gDitlList = GetResource('DITL', kControlListID);
theErr = NavCustomControl(callBackParms->context, kNavCtlAddControlList, gDitlList);
```

The advantage of using a 'DITL' resource is that the Dialog Manager will handle all control movement and tracking.

16B-12 Files

1

⁵ The customRect field contains an empty rectangle if customisation is not allowed (i.e. dialog boxes other than Open or Save). In this case, your application can simply ignore the call.

 When Navigation Services supplies the kNavCBTerminate event message constant in the param field of the NavCBRec structure (after the user closes the dialog box), dispose of the control or resource.

Main Navigation Services Constants, Data Types, and Functions

Constants

Configuration Options

```
kNavDefaultNavDlogOptions = 0x000000E4
                                                                                                                                       // Use defaults for all the options
                                                                              = 0x00000001 // Don't show file type/extension popup of the control of the contro
 kNavNoTypePopup
                                                                                                                                       // Don't show file type/extension popup on Open/Save
 kNavDontAutoTranslate
                                                                                                                                       // Don't add translation choices on Open/Save
 kNavDontAddTranslateItems = 0x00000004
kNavAllFilesInPopup = 0x00000010 // "All Files" menu item in the type popup on Open kNavAllowStationery = 0x00000020 // Allow saving of stationery files = 0x00000040 // Allow to show provious
 kNavAllowPreviews
                                                                                       = 0x00000040
                                                                                                                                         // Allow to show previews
 kNavAllowMultipleFiles = 0x00000080
                                                                                                                                        // Allow multiple items to be selected
kNavAllowInvisibleFiles = 0x00000100
                                                                                                                                         // Allow invisible items to be shown
                                                                                                                                         // Don't resolve aliases
 kNavDontResol veAliases
                                                                                      = 0x00000200
 kNavSelectDefaultLocation = 0x00000400
                                                                                                                                        // Make the default location the browser selection
 kNavSelectAllReadableItem = 0x00000800
                                                                                                                                        // Make dialog select "All Readable Documents" on Open
```

Save Changes Request

```
kNavSaveChangesClosingDocument = 1 \\ kNavSaveChangesQuittingApplication = 2 \\ kNavSaveChangesOther = 0
```

Save Changes Action

kNavAskSaveChangesSave = 1 kNavAskSaveChangesCancel = 2 kNavAskSaveChangesDontSave = 3

Discard Changes Action

kNavAskDi scardChanges = 1 kNavAskDi scardChangesCancel = 2

Event Message

kNavCBEvent	= 0
kNavCBCustomi ze	= 1
kNavCBStart	= 2
kNavCBTermi nate	= 3

Object Filtering

kNavFilteringBrowserList	=	0
kNavFilteringFavorites	=	1
kNavFilteringRecents	=	2
kNavFilteringShortCutVolumes	=	3

Data Types

NavDialog Options Structure

```
struct NavDialogOptions
  UInt16
                          version:
  NavDialogOptionFlags dialogOptionFlags;
  Poi nt
                         location;
  Str255
                         clientName;
  Str255
                          windowTitle:
  Str255
                          action Button Label \; ; \\
  Str255
                          cancel ButtonLabel;
                          savedFileName;
  Str255
  Str255
                          message;
```

```
Handl e
                                              popupExtension;
    char
                                              reserved[494];
};
typedef struct NavDialogOptions NavDialogOptions;
Nav Reply Structure
struct NavReplyRecord
     UI nt 16
                                              version;
     Bool ean
                                              validRecord;
                                              replacing;
     Bool ean
     Bool ean
                                              i sStationery
                                              translationNeeded;
    Bool ean
     AEDescList
                                              selection;
     ScriptCode
                                              keyScript;
                                             **fileTranslation;
    FileTranslationSpec
    UI nt 32
                                              reserved1;
                                             reserved[231];
    char
}:
typedef struct NavReplyRecord NavReplyRecord;
NavTypeList
struct NavTypeList
     0SType
                                              componentSi gnature;
    short
                                              reserved:
     short
                                              osTypeCount;
    0SType
                                              osType[1];
typedef struct NavTypeList NavTypeList;
typedef NavTypeList*NavTypeListPtr;
typedef NavTypeListPtr *NavTypeListHandle;
Functions
Calling Navigation Services
0SErr
             NavLoad(void);
0SErr
             NavUnload(void);
              NavGetDefaultDialog0ptions \,(NavDialog0ptions\ *dialog0ptions)\,;
0SErr
OSErr NavDi sposeRepl y(NavRepl yRecord *repl y);
Choosing Files, Folders, and Volumes
              NavGetFile(AEDesc *defaultLocation, NavReplyRecord *reply,
              NavDi al ogOpti ons *di al ogOpti ons, NavEventUPP eventProc, NavPrevi ewUPP previ ewProc,
              NavObjectFilterUPP filterProc, NavTypeListHandle typeList,
              NavCallBackUserData callBackUD);
              NavChooseFile(AEDesc *defaultLocation, NavReplyRecord *reply,
              NavDialogOptions *dialogOptions, NavEventUPP eventProc, NavPreviewUPP previewProc,
              Nav0bj\,ectFilter UPP\ filterProc, NavTypeListHandle\ typeList,
              NavCallBackUserData callBackUD);
0SErr
              NavChooseVolume(AEDesc *defaultSelection, NavReplyRecord *reply,
              NavDialogOptions *dialogOptions, NavEventUPP eventProc, NavObjectFilterUPP filterProc,
              NavCallBackUserData callBackUD);
0SErr
              NavChooseFolder(AEDesc *defaultLocation, NavReplyRecord *reply,
              NavDialogOptions *dialogOptions, NavEventUPP eventProc, NavObjectFilterUPP filterProc,
              NavCallBackUserData callBackUD);
0SErr
              NavChooseObject(AEDesc *defaultLocation, NavReplyRecord *reply,
              NavDialog 0 ptions * dialog 0 ptions, NavEventUPP \ eventProc, NavObjectFilterUPP \ filterProc, N
              NavCallBackUserData callBackUD);
              NavNewFolder(AEDesc *defaultLocation, NavReplyRecord *reply,
              NavDialogOptions *dialogOptions, NavEventUPP eventProc, NavCallBackUserData callBackUD);
Saving Files
              NavPutFile(AEDesc *defaultLocation, NavReplyRecord *reply,
              NavDialogOptions *dialogOptions, NavEventUPP eventProc, OSType fileType,
              OSType fileCreator, NavCallBackUserData callBackUD);
0SErr
              NavAskSaveChanges(NavDialogOptions *dialogOptions, NavAskSaveChangesAction action,
```

UInt32

preferenceKey;

16B-14 Files

NavAskSaveChangesResult *reply, NavEventUPP eventProc, NavCallBackUserData callBackUD);

Demonstration Program

```
// Files2.h
// ***
11
// This program demonstrates:
11
// .
      The use of Navigation Services rather than the Standard File Package to display
//
      Open, Save, and Choose a Folder dialog boxes.
11
// •
      The use of Navigation Services rather than the Dialog Manager to display Save
//
      Changes and Discard Changes alert boxes.
//
// •
      Application-defined file handling functions associated with:
//
//
          The user invoking the File menu Open..., Close, Save, Save As..., Revert to Saved,
//
          and Quit commands of a typical application.
//
//
          Handling of the required Apple events Open Application, Re-open Application,
//
          Open Documents and Quit Application.
11
      These functions are essentially the same as those in the demonstration program
11
      Files1 except that the safe-save methodology used in Files1 is not used, all
//
      saves being direct to the target file.
11
// To keep the code not specifically related to files and file-handling to a minimum, an
// item is included in the Demonstration menu which allows the user to simulate
// "touching" a window (that is, modifying the contents of the associated document).
// Choosing the first menu item in this menu sets the window-touched flag in the window's
// document structure to true and draws the text "WINDOW TOUCHED" in the window in a
/\!/ large font size, this latter so that the user can keep track of which windows have /\!/ been "touched".
\ensuremath{//} The program utilises the following resources:
11
// •
      An 'MBAR' resource, and 'MENU' resources for Apple, File, Edit and Demonstration
11
      menus (preload, non-purgeable).
11
     A 'WIND' resource (purgeable) (initially not visible).
//
// •
      A 'STR' resource containing the "missing application name" string, which is copied
      to all document files created by the program.
11
// .
      'STR#' resources (purgeable) containing error strings, the application's name (for
//
      certain Navigation Services functions), and a message string for the Choose a
//
      Folder dialog box.
//
//
      A 'kind' resource (purgeable) describing file types, which is used by Navigation
11
      Services to build the native file types section of the Show pop-up menu in the
11
      Open dialog box.
//
// •
      An 'open' resource (purgeable) containing the file type list for the Open dialog
11
//
// .
      A 'pnot' and associated 'PICT' resource (both purgeable), which provide the preview
//
      for the PICT file.
      The 'BNDL' resource (non-purgeable), 'FREF' resources (non-purgeable), signature
      resource (non-purgeable), and icon family resources (purgeable), required to
//
      support the built application.
11
// • A 'SIZE' resource with the acceptSuspendResumeEvents, isHighLevelEventAware, and
//
      is32BitCompatible flags set (non-purgeable).
```

```
..... i ncl udes
#include <Appearance.h>
#include <AERegistry.h>
#include <Devices. h>
#include <Folders.h>
#include <Fonts.h>
#include <LowMem. h>
#include <Navigation.h>
#include <Processes.h>
#include <Resources.h>
#include <StandardFile.h>
#include <TextUtils.h>
#include <ToolUtils.h>
// _____typedefs
typedef struct
  TEHandl e
             editStrucHdl;
             pictureHdl;
 Pi cHandl e
             fileRefNum;
  SInt 16
  FSSpec
             fileFSSpec;
             windowTouched;
  Bool ean
} docStructure, *docStructurePointer, **docStructureHandle;
typedef StandardFileReply *standardFileReplyPtr;
                                            defines
#define mApple
                              128
#define iAbout
#define mFile
                              129
#define iNew
                              1
#define iOpen
                              2
#define iClose
                              4
#define iSave
                              5
#define iSaveAs
#define i Revert
#define i Quit
                              12
#define mDemonstration
                              131
#define i TouchWindow
#define i ChooseAFolderDialog
                              1
                              3
#define rNewWindow
                              128
#define rMenubar
                              128
#define rRevertAlert
                              128
#define rCloseFileAlert
                              129
#define rCustomOpenDialog
                              130
#define iPopupItem
                              10
#define rSelectDirectoryDialog 131
#define iSelectButton
                              10
#define rErrorStrings
                              128
#define eInstallHandler
                              1000
#define eMaxWindows
                              1001
#define eFileIsOpen
                              opWrErr
#define eNoNavServices
                              1002
#define rMiscStrings
                              129
#define sApplicationName
                              1
#define sChooseAFolder
                              2
#define rOpenResource
                              128
#define kMaxWindows
                              10
                              0x7FFFFFFF
#define MAXLONG
#define MIN(a,b)
                              ((a) < (b) ? (a) : (b))
// ______function prototypes
void main
                            (void);
void eventLoop
                            (void);
     doI ni tManagers
                            (void);
voi d
     doInstallAEHandlers
voi d
                            (void);
voi d
     doEvents
                            (EventRecord *);
     doMouseDown
                            (EventRecord *);
voi d
voi d
     doUpdate
                            (EventRecord *);
voi d doMenuChoi ce
                            (SInt 32);
voi d
     doFileMenu
                            (SInt 16);
voi d
     doAdj ustMenus
                            (void);
     doErrorAl ert
                            (SInt 16);
voi d
```

16B-16 Files

```
void doCopyPString
                         (Str255, Str255);
     doConcat PStrings
                         (Str255, Str255);
voi d
voi d doTouchWi ndow
                         (void);
pascal OSErr doOpenAppEvent
                                (AppleEvent *, AppleEvent *, SInt32);
pascal OSErr
           doReopenAppEvent
                                (Appl eEvent *, Appl eEvent *, SI nt 32);
                                (Appl eEvent *, Appl eEvent *, SI nt 32);
(Appl eEvent *, Appl eEvent *, SI nt 32);
pascal OSErr doOpenDocsEvent
pascal OSErr doQuitAppEvent
0SErr
           doHasGotRequi redParams
                                (AppleEvent *);
OSErr doNewCommand
                         (void):
OSErr doOpenCommand
                         (void);
OSErr doCloseCommand
                         (NavAskSaveChangesAction);
OSErr doSaveCommand
                         (void):
OSErr doSaveAsCommand
                         (void);
OSErr doRevertCommand
                         (void);
OSErr doQuitCommand
                         (NavAskSaveChangesAction);
OSErr doNewDocWindow
                         (Boolean, OSType);
OSErr doOpenFile
                         (FSSpec, OSType);
OSErr doReadTextFile
                         (WindowPtr);
OSErr doReadPictFile
                         (WindowPtr);
OSErr doCloseFile
                         (WindowPtr, docStructureHandle, NavAskSaveChangesAction);
OSErr doWriteFile
                         (WindowPtr. Boolean):
OSErr doWriteTextData
                         (WindowPtr, SInt16);
                         (WindowPtr, SInt16);
OSErr doWritePictData
OSErr doCopyResources
                         (WindowPtr):
OSErr doCopyAResource
                         (ResType, SInt16, SInt16, SInt16);
pascal\ void\ navEventFunction\ (NavEventCallbackMessage, NavCBRecPtr, NavCallBackUserData);
OSErr doDirectorySelecDialog (FSSpec *);
// Files2.c
         // _____i i ncl udes
#include "Files2.h"
// ______global variables
               gDone;
Bool ean gI nBackground;
AEEventHandl erUPP doOpenAppEventUPP;
AEEventHandlerUPP doReopenAppEventUPP;
AEEventHandlerUPP doOpenDocsEventUPP;
AEEventHandlerUPP doQuitAppEventUPP;
SInt 16
               gAppResFileRefNum;
extern SInt16 gCurrentNumberOfWindows;
             gDestRect, gVi ewRect;
extern Rect
void main(void)
 Handl e
           menubarHdl;
 MenuHandle menuHdl;
 // _____initialise managers
 doInitManagers();
 // ..... cause the Appearance-compliant menu bar definition function to be called directly
 RegisterAppearanceClient();
                       if(NavServicesAvailable())
   NavLoad();
   doErrorAl ert(eNoNavServi ces);
 // _____create routine decriptors for Apple event handlers
 doOpenAppEventUPP = NewAEEventHandlerProc((ProcPtr) doOpenAppEvent);
 doReopenAppEventUPP = NewAEEventHandlerProc((ProcPtr) doReopenAppEvent);
```

```
doOpenDocsEventUPP = NewAEEventHandlerProc((ProcPtr) doOpenDocsEvent);
 doQuitAppEventUPP = NewAEEventHandlerProc((ProcPtr) doQuitAppEvent);
 // .....set application's resource fork as current resource file
 gAppResFileRefNum = CurResFile();
 // _____set up menu bar and menus
 menubarHdl = GetNewMBar(rMenubar);
 if(menubarHdl == NULL)
  doErrorAl ert(MemError());
 SetMenuBar(menubarHdl);
 DrawMenuBar();
 menuHdl = GetMenuHandle(mApple);
 if(menuHdl == NULL)
  doErrorAl ert(MemError());
 else
   AppendResMenu(menuHdl, 'DRVR');
                       ...... install required Apple event handlers
 doInstallAEHandlers();
 // _____enter event loop
 eventLoop();
voi d eventLoop(voi d)
 EventRecord eventStructure;
 gDone = false;
 while(!gDone)
   if(WaitNextEvent(everyEvent, &eventStructure, MAXLONG, NULL))
    doEvents(&eventStructure);
}
voi d doInitManagers(voi d)
 MaxAppl Zone();
 MoreMasters();
 InitGraf(&qd.thePort);
 InitFonts();
 InitWindows();
 InitMenus();
 TEI ni t();
 InitDialogs(NULL);
 InitCursor();
 FlushEvents(everyEvent, 0);
void doInstallAEHandlers(void)
 OSErr osError;
 osError = AEInstallEventHandler(kCoreEventClass, kAEOpenApplication, doOpenAppEventUPP,
                          OL, false);
 if(osError != noErr) doErrorAlert(eInstallHandler);
 osError = AEInstallEventHandler(kCoreEventClass, kAEReopenApplication,
                          doReopenAppEventUPP, OL, false);
 if(osError != noErr) doErrorAlert(eInstallHandler);
```

16B-18 Files

```
osError = AEInstallEventHandler(kCoreEventClass, kAEOpenDocuments, doOpenDocsEventUPP,
                               OL, false);
 if(osError != noErr) doErrorAlert(eInstallHandler);
 osError = AEInstallEventHandler(kCoreEventClass, kAEQuitApplication, doQuitAppEventUPP,
                               OL, false);
 if(osError != noErr) doErrorAlert(eInstallHandler);
  void doEvents(EventRecord *eventStrucPtr)
 SInt8 charCode;
 switch(eventStrucPtr->what)
   case kHighLevelEvent:
     AEProcessAppl eEvent (eventStrucPtr);
     break;
   case mouseDown:
     doMouseDown(eventStrucPtr);
     break;
   case keyDown:
   case autoKey:
     charCode = eventStrucPtr->message & charCodeMask;
     if((eventStrucPtr->modifiers & cmdKey) != 0)
       doAdj ustMenus();
       doMenuChoi ce(MenuEvent(eventStrucPtr));
     break;
   case updateEvt:
     doUpdate(eventStrucPtr);
     break;
   case osEvt:
     switch((eventStrucPtr->message >> 24) & 0x000000FF)
       case suspendResumeMessage:
         gInBackground = (eventStrucPtr->message & resumeFlag) == 0;
         break;
     HiliteMenu(0);
     break:
 }
      void doMouseDown(EventRecord *eventStrucPtr)
 WindowPtr windowPtr;
 SInt16
         partCode;
 partCode = FindWindow(eventStrucPtr->where, &windowPtr);
 switch(partCode)
   case inMenuBar:
     doAdj ustMenus();
     doMenuChoi ce(MenuSel ect(eventStrucPtr->where));
     break;
   case inContent:
     if(windowPtr != FrontWindow())
       Sel ectWi ndow(wi ndowPtr);
     break;
   case inDrag:
     DragWi ndow(wi ndowPtr, eventStrucPtr->where, &qd. screenBi ts. bounds);
     break;
   case inGoAway:
```

```
if(TrackGoAway(windowPtr, eventStrucPtr->where) == true)
       doCloseCommand(kNavSaveChangesClosingDocument);
     break:
}
                          ************** doUpdate
voi d doUpdate(EventRecord *eventStrucPtr)
 WindowPtr
                    windowPtr;
 docStructureHandle
                    docStrucHdl;
                    oldPort:
 GrafPtr
                    destRect;
 Rect
 windowPtr = (WindowPtr) eventStrucPtr->message;
 docStrucHdl = (docStructureHandle) GetWRefCon(windowPtr);
  GetPort(&oldPort);
 SetPort(windowPtr);
 Begi nUpdate(wi ndowPtr);
 if((*docStrucHdl)->pictureHdl)
   destRect = (*(*docStrucHdl)->pictureHdl)->picFrame;
   OffsetRect(&destRect, 170, 54);
   HLock((Handle) (*docStrucHdl)->pictureHdl);
   DrawPicture((*docStrucHdl)->pictureHdl, &destRect);
   HUnlock((Handle) (*docStrucHdl)->pictureHdl);
 else if((*docStrucHdl)->editStrucHdl)
   HLock((Handle) (*docStrucHdl)->editStrucHdl);
   TEUpdate(&gDestRect, (*docStrucHdl) - >editStrucHdl);
   HUnlock((Handle) (*docStrucHdl)->editStrucHdl);
 i f((*docStrucHdl)->wi ndowTouched)
   TextSize(48);
   MoveTo(30, 170);
   DrawString("\pWINDOW TOUCHED");
   TextSize(12);
 EndUpdate((WindowPtr)eventStrucPtr->message);
 SetPort(oldPort);
  void doMenuChoice(SInt32 menuChoice)
 SInt16 menuID, menuItem;
 Str255 itemName;
 SInt16 daDriverRefNum;
 0SErr
         osError;
 FSSpec fileSpec;
 Rect
         theRect;
 Str255 theString, numberString;
 menuID = Hi Word(menuChoi ce);
 menuItem = LoWord(menuChoice);
 if(menuID == 0)
   return;
 switch(menuID)
   case mApple:
     GetMenuItemText(GetMenuHandle(mApple), menuItem, itemName);
     daDri verRefNum = OpenDeskAcc(itemName);
     break:
   case mFile:
```

16B-20 Files

```
doFileMenu(menuItem);
     break;
   case mDemonstration:
     if(menuItem == iTouchWindow)
       doTouchWindow();
     else if(menuItem == iChooseAFolderDialog)
       doErrorAl ert(osError);
       else
         if(FrontWindow())
          SetPort(FrontWindow());
          TextSize(10);
          SetRect(&theRect, 0, 271, 600, 300);
          EraseRect(&theRect);
          if(osError != userCanceledErr)
            doCopyPString(fileSpec.name, theString);
            doConcatPStrings(theString, "\p Volume Reference Number: ");
            NumToString((SInt32) fileSpec. vRefNum, numberString);
            doConcatPStrings(theString, numberString);
            doConcatPStrings(theString, "\p Parent Directory ID: ");
            NumToString((SInt32) fileSpec.parID, numberString);
            doConcatPStrings(theString, numberString);
            MoveTo(10, 290);
            DrawString(theString);
        }
       }
     break;
 }
 HiliteMenu(0);
      void doFileMenu(SInt16 menuItem)
 OSErr osError;
 switch(menuItem)
 {
   case iNew:
     if(osError = doNewCommand())
       doErrorAl ert(osError);
     break:
   case iOpen:
     if(osError = doOpenCommand())
       doErrorAl ert(osError);
     break;
   case iClose:
     if((osError = doCloseCommand(kNavSaveChangesClosingDocument)) &&
         osError != kNavAskSaveChangesCancel)
       doErrorAl ert(osError);
     break;
   case iSave:
     if(osError = doSaveCommand())
       doErrorAl ert(osError);
     break;
   case iSaveAs:
     if(osError = doSaveAsCommand())
       doErrorAl ert(osError);
     break;
   case iRevert:
     if(osError = doRevertCommand())
       doErrorAl ert(osError);
     break;
```

```
case iQuit:
     if((osError = doQuitCommand(kNavSaveChangesQuittingApplication)) &&
         osError != kNavAskSaveChangesCancel)
       doErrorAl ert(osError);
     if(osError != kNavAskSaveChangesCancel)
       NavUnl oad();
       gDone = true;
     break;
 }
}
   void doAdjustMenus(void)
                    menuHdl;
  MenuHandl e
  WindowPtr
                    windowPtr;
 docStructureHandle docStrucHdl;
 windowPtr = FrontWindow();
 docStrucHdl = (docStructureHandle) GetWRefCon(windowPtr);
  menuHdl = GetMenuHandle(mFile);
 if(gCurrentNumber0fWindows > 0)
   menuHdl = GetMenuHandle(mFile);
   EnableItem(menuHdl, iClose);
   if((*docStrucHdl)->windowTouched)
   {
     EnableItem(menuHdl, iSave);
     EnableItem(menuHdl, iRevert);
   }
   else
   {
     Di sableItem(menuHdl, iSave);
     Di sableItem(menuHdl, i Revert);
   EnableItem(menuHdl, iSaveAs);
   menuHdl = GetMenuHandle(mDemonstration);
   if((*docStrucHdl)->windowTouched == false)
     EnableItem(menuHdl, i TouchWindow);
     Di sableItem(menuHdl, i TouchWindow);
 }
 else
   menuHdl = GetMenuHandle(mFile);
   DisableItem(menuHdl, iClose);
   Di sableItem(menuHdl, iSave);
   Di sabl eI tem(menuHdl, i SaveAs);
   Di sableItem(menuHdl, i Revert);
   menuHdl = GetMenuHandle(mDemonstration);
   Di sableItem(menuHdl, i TouchWindow);
 }
 DrawMenuBar();
void doErrorAlert(SInt16 errorCode)
 AlertStdAlertParamRec paramRec;
 Str255
                      errorString, theString;
 SInt 16
                      itemHit;
  paramRec. movable
                        = true;
  paramRec. helpButton
                        = false;
 paramRec.filterProc
                        = NULL;
  paramRec. defaultText
                        = (StringPtr) kAlertDefaultOKText;
  paramRec. cancel Text
                        = NULL;
  paramRec. otherText
                        = NULL;
```

16B-22 Files

```
paramRec. defaultButton = kAlertStdAlertOKButton;
  paramRec. cancel Button = 0;
                        = kWindowDefaultPosition:
  paramRec. position
  if(errorCode == eInstallHandler)
   GetIndString(errorString, rErrorStrings, 1);
  else if(errorCode == eMaxWindows)
   GetIndString(errorString, rErrorStrings, 2);
  else if(errorCode == eFileIsOpen)
   GetIndString(errorString, rErrorStrings, 3);
  else if(errorCode == eNoNavServices)
   GetIndString(errorString, rErrorStrings, 4);
  else
   GetIndString(errorString, rErrorStrings, 5);
   NumToString((SInt32) errorCode, theString);
   doConcatPStrings(errorString, theString);
 }
  if(errorCode != memFullErr && errorCode != eNoNavServices)
   StandardAlert(kAlertCautionAlert, errorString, NULL, &paramRec, &itemHit);
   StandardAl ert(kAl ertStopAl ert, errorString, NULL, &paramRec, &i temHit);
   ExitToShell();
voi\,d\quad do Copy PS tring (Str255\ source String,\,Str255\ destination String)
 SInt16 stringLength;
  stringLength = sourceString[0];
 BlockMove(sourceString + 1, destinationString + 1, stringLength);
  destinationString[0] = stringLength;
  void doConcatPStrings(Str255 targetString, Str255 appendString)
 SInt16 appendLength;
  appendLength = MIN(appendString[0], 255 - targetString[0]);
  if(appendLength > 0)
   BlockMoveData(appendString+1, targetString+targetString[0]+1, (SInt32) appendLength);
   targetString[0] += appendLength;
                              ******* doTouchWi ndow
void doTouchWindow(void)
 Wi ndowPtr
                    windowPtr;
  docStructureHandle docStrucHdl;
  wi ndowPtr = FrontWi ndow();
  docStrucHdl = (docStructureHandle) GetWRefCon(windowPtr);
  SetPort(windowPtr);
 TextSize(48);
  MoveTo(30, 170);
  DrawString("\pWINDOW TOUCHED");
  TextSize(12);
  (*docStrucHdl)->windowTouched = true;
pascal OSErr doOpenAppEvent(AppleEvent *appEvent, AppleEvent *reply, SInt32 handlerRefCon)
```

```
OSErr osError;
 osError = doHasGotRequiredParams(appEvent);
 if(osError == noErr)
   osError = doNewCommand();
 return(osError);
  pascal OSErr doReopenAppEvent(AppleEvent *appEvent, AppleEvent *reply,
                            SInt32 handlerRefCon)
 OSErr osError;
 osError = doHasGotRequiredParams(appEvent);
 if(osError == noErr)
   if(!FrontWindow())
     osError = doNewCommand();
 return(osError);
   pascal OSErr doOpenDocsEvent(AppleEvent *appEvent, AppleEvent *reply, SInt32 handlerRefcon)
 FSSpec
            fileSpec:
 AEDescList docList;
  0SErr
            osError, ignoreErr;
            index, numberOfItems;
 SInt32
            actual Si ze;
 Si ze
  AEKeyword
            keyWord;
            returnedType;
 DescType
 FInfo
            fileInfo;
 osError = AEGetParamDesc(appEvent, keyDirectObject, typeAEList, &docList);
 if(osError == noErr)
   osError = doHasGotRequiredParams(appEvent);
   if(osError == noErr)
     AECountItems(&docList, &numberOfItems);
     if(osError == noErr)
     {
       for(index=1; index<=number0fItems; index++)</pre>
         osError = AEGetNthPtr(&docList, index, typeFSS, &keyWord, &returnedType,
                           (Ptr) &fileSpec, sizeof(fileSpec), &actualSize);
         if(osError == noErr)
         {
          osError = FSpGetFInfo(&fileSpec, &fileInfo);
          if(osError == noErr)
          {
            if(osError = doOpenFile(fileSpec, fileInfo.fdType))
              doErrorAl ert(osError);
         }
          doErrorAl ert(osError);
     }
   }
   el se
     doErrorAl ert(osError);
   i gnoreErr = AEDi sposeDesc(&docLi st);
 }
 else
   doErrorAl ert(osError);
 return(osError);
```

16B-24 Files

```
pascal OSErr doQuitAppEvent(AppleEvent *appEvent, AppleEvent *reply, SInt32 handlerRefcon)
 0SErr
        osError;
 osError = doHasGotRequiredParams(appEvent);
 if(osError == noErr)
 {
   while(FrontWindow())
     osError = doCloseCommand(kNavSaveChangesQuittingApplication);
     if(osError != noErr && osError != kNavAskSaveChangesCancel)
      doErrorAl ert(osError);
     if(osError == kNavAskSaveChangesCancel)
      eventLoop();
   }
 }
 NavUnload();
 gDone = true;
 return(osError);
  ********* doHasGotRequiredParams
OSErr doHasGotRequiredParams(AppleEvent *appEvent)
 DescType returnedType;
          actual Si ze;
 Si ze
 0SErr
          osError:
 osError = AEGetAttri butePtr(appEvent, keyMi ssedKeywordAttr, typeWi l dCard, &returnedType,
                       NULL, 0, &actual Size);
 if(osError == errAEDescNotFound)
   return(noErr);
 else if(osError == noErr)
   return(errAEParamMissed);
// NewOpenCloseSave.c
#include "Files2.h"
                                   .....global variables
WindowPtr
          gWindowPtr;
          gCurrentNumberOfWi ndows = 0;
SInt 16
Rect
          gDestRect, gVi ewRect;
extern\ SInt16\ gAppResFileRefNum;
OSErr doNewCommand(void)
 0SErr
        osError;
 OSType documentType = 'TEXT';
 osError = doNewDocWindow(true, documentType);
 return(osError);
                     *********** doOpenCommand
// ************
OSErr doOpenCommand(void)
 NavDi al ogOptions
                  di al og0pti ons;
 NavTypeLi stHandle
                  fileTypeListHdl = NULL;
 NavEventUPP
                  navEventFunctionUPP;
 0SErr
                  osError = noErr;
```

```
NavRepl yRecord
                     navReplyStruc;
  AEDesc
                     resultDesc;
  SInt32
                     index, count;
  FSSpec
                     fileSpec;
                     fileInfo;
  FInfo
                     documentType;
  OSType
  NavGetDefaultDialogOptions(&dialogOptions);
  GetIndString((unsigned char*) &dialogOptions.clientName, rMiscStrings, sApplicationName);
  fileTypeListHdl = (NavTypeListHandle) GetResource('open', r0penResource);
  navEventFunctionUPP = NewNavEventProc((ProcPtr) navEventFunction);
  osError = NavGetFile(NULL, &navReplyStruc, &dialogOptions, navEventFunctionUPP, NULL, NULL,
                    fileTypeListHdl, 0);
  DisposeRoutineDescriptor(navEventFunctionUPP);
  if(navReplyStruc.validRecord && osError == noErr)
   if(osError == noErr)
   {
     osError = AECountItems(&(navReplyStruc.selection), &count);
     for(i ndex=1; i ndex<=count; i ndex++)</pre>
       resultDesc.dataHandle = 0L;
        if((osError = AEGetNthDesc(&(navReplyStruc. selection), index, typeFSS, NULL,
          &resultDesc)) == noErr)
         BlockMoveData(*resultDesc.dataHandle, &fileSpec, sizeof(FSSpec));
         if((osError = FSpGetFInfo(&fileSpec, &fileInfo)) == noErr)
           documentType = fileInfo.fdType;
           osError = doOpenFile(fileSpec, documentType);
         AEDi sposeDesc(&resultDesc);
     }
   }
    NavDi sposeRepl y(&navRepl yStruc);
  }
  if(fileTypeListHdl != NULL)
   ReleaseResource((Handle) fileTypeListHdl);
  if(osError == userCanceledErr)
   osError = noErr;
  return(osError);
OSErr doCloseCommand(NavAskSaveChangesAction action)
  WindowPtr
                     windowPtr:
  SInt 16
                     wi ndowKi nd;
  docStructureHandle docStrucHdl;
                     osError = noErr;
  0SErr
  windowPtr = FrontWindow();
  windowKind = ((WindowPeek) windowPtr)->windowKind;
  switch(windowKind)
   case kApplicationWindowKind:
     docStrucHdl = (docStructureHandle) GetWRefCon(windowPtr);
     osError = doCloseFile(windowPtr, docStrucHdl, action);
```

16B-26 Files

```
if(osError == kNavAskSaveChangesCancel)
       return(kNavAskSaveChangesCancel);
      else if(osError == noErr)
       Di sposeWi ndow(wi ndowPtr);
       gCurrentNumberOfWindows --;
     break;
    case kDi al ogWi ndowKi nd:
      // Hide or close modeless dialog, as required.
     break:
 }
 return(osError);
OSErr doSaveCommand(void)
  Wi ndowPtr
                     windowPtr;
  docStructureHandle docStrucHdl;
  0SErr
                     osError = noErr;
  windowPtr = FrontWindow();
  docStrucHdl = (docStructureHandle) GetWRefCon(windowPtr);
  if((*docStrucHdl)->fileRefNum)
  {
    osError = doWriteFile(windowPtr, false);
    SetPort(windowPtr):
    EraseRect(&windowPtr->portRect);
   Inval Rect(&windowPtr->portRect);
 }
  el se
    osError = doSaveAsCommand();
  return(osError);
                               ******** doSaveAsCommand
OSErr doSaveAsCommand(void)
 NavDi al ogOptions
                     di al og0pti ons;
  Wi ndowPtr
                     windowPtr:
  docStructureHandle docStrucHdl;
  NavEvent UPP
                     navEventFunctionUPP;
  0SType
                     fileType;
  0SErr
                     osError = noErr;
  NavRepl yRecord
                     navReplyStruc;
  AEDesc
                     resultDesc:
  FSSpec
                     fileSpec;
  SInt16
                     fileRefNum;
  NavGetDefaultDi al ogOptions(&di al ogOptions);
  windowPtr = FrontWindow();
  docStrucHdl = (docStructureHandle) GetWRefCon(windowPtr);
  GetWTitle(windowPtr, dialog0ptions.savedFileName);
  GetIndString((unsigned\ char^*)\ \&dialog0ptions.\ clientName,\ rMiscStrings,\ sApplicationName);
  navEventFunctionUPP = NewNavEventProc((ProcPtr) navEventFunction);
  if((*docStrucHdl)->editStrucHdl)
   fileType = 'TEXT';
  else if((*docStrucHdl)->pictureHdl)
   fileType = 'PICT';
  osError = NavPutFile(NULL, &navReplyStruc, &dialogOptions, navEventFunctionUPP,
                      fileType, 'kkkB', NULL);
  Di sposeRouti neDescri ptor(navEventFuncti onUPP);
  if(navReplyStruc.validRecord && osError == noErr)
```

```
resultDesc.dataHandle = 0L;
    if((osError = AEGetNthDesc(&(navReplyStruc.selection), 1, typeFSS, NULL, &resultDesc))
      BlockMoveData(*resultDesc.dataHandle, &fileSpec, sizeof(FSSpec));
      if(!navReplyStruc.replacing)
        osError = FSpCreate(&fileSpec, 'kkkB', fileType, navReplyStruc.keyScript);
        if(osError != noErr)
          AEDi sposeDesc(&resultDesc);
          NavDi sposeRepl y(&navRepl yStruc);
          return(osError);
      }
      (*docStrucHdl)->fileFSSpec = fileSpec;
      if((*docStrucHdl)->fileRefNum != 0)
      {
        osError = FSClose((*docStrucHdl)->fileRefNum);
        (*docStrucHdl)->fileRefNum = 0;
      if(osError == noErr)
        osError = FSpOpenDF(&(*docStrucHdl)->fileFSSpec, fsRdWrPerm, &fileRefNum);
      if(osError == noErr)
        (*docStrucHdl)->fileRefNum = fileRefNum:
        SetWTitle(windowPtr, fileSpec. name);
        osError = doWriteFile(windowPtr,!navReplyStruc.replacing);
      AEDi sposeDesc(&resul tDesc);
      NavCompleteSave (\&navReplyStruc, kNavTranslateInPlace);\\
   NavDi sposeRepl y(&navRepl yStruc);
  SetPort(windowPtr);
  EraseRect(&windowPtr->portRect);
 Inval Rect(&windowPtr->portRect);
  if(osError == userCanceledErr)
    osError = noErr;
  return(osError);
                              ******** doRevertCommand
// ************
OSErr doRevertCommand(void)
                          navEventFunction UPP;\\
  NavEvent UPP
  WindowPtr
                          wi ndowPtr
  docStructureHandle
                          docStrucHdl;
  Str255
                          fileName:
  NavDi al ogOpti ons
                          di al og0ptions;
  NavAskSaveChangesResult reply;
                          osError = noErr;
  0SErr
  navEventFunctionUPP = NewNavEventProc((ProcPtr) navEventFunction);
  windowPtr = FrontWindow();
  docStrucHdl = (docStructureHandle) GetWRefCon(windowPtr);
 SetPort(windowPtr);
  GetWTitle(windowPtr, fileName);
  BlockMoveData(fileName, dialogOptions.savedFileName, fileName[1]);
```

16B-28 Files

```
osError = NavAskDi scardChanges(&di al ogOpti ons, &repl y, navEventFuncti onUPP, 0);
  if(reply == kNavAskDiscardChanges)
    EraseRect(&windowPtr->portRect);
   if((*docStrucHdl)->editStrucHdl)
   {
     osError = doReadTextFile(windowPtr);
   else if((*docStrucHdl)->pictureHdl)
   {
     KillPicture((*docStrucHdl)->pictureHdl);
     (*docStrucHdl) - >pi ctureHdl = NULL;
     osError = doReadPictFile(windowPtr);
    (*docStrucHdl)->windowTouched = false;
    Inval Rect(&wi ndowPtr->portRect);
  return(osError);
// ******************* doQuitCommand
OSErr doQuitCommand(NavAskSaveChangesAction action)
 OSErr osError = noErr;
  while(FrontWindow())
  {
    osError = doCloseCommand(action);
   if(osError != noErr)
     return(osError);
 return(osError);
// ******************* doNewDocWi ndow
OSErr doNewDocWindow(Boolean showWindow, OSType documentType)
  docStructureHandle docStrucHdl;
 if(gCurrentNumberOfWindows == kMaxWindows)
    return(eMaxWi ndows);
 if(!(gWindowPtr = GetNewCWindow(rNewWindow, NULL, (WindowPtr)-1)))
    return(MemError());
  SetPort(gWindowPtr);
  if(!(docStrucHdl = (docStructureHandle) NewHandle(sizeof(docStructure))))
   Di sposeWi ndow(gWi ndowPtr);
    return(MemError());
  SetWRefCon(gWindowPtr, (SInt32) docStrucHdl);
  (*docStrucHdl)->editStrucHdl = NULL;
  (*docStrucHdl) - >pictureHdl = NULL;
  (*docStrucHdl) ->fileRefNum = 0;
  (*docStrucHdl)->windowTouched = false;
  if(documentType == 'TEXT')
    gDestRect = gWindowPtr->portRect;
    InsetRect(&gDestRect, 6, 6);
    gViewRect = gDestRect;
    MoveHHi ((Handle) docStrucHdl);
    HLock((Handle) docStrucHdl);
```

```
if(!((*docStrucHdl)->editStrucHdl = TENew(&gDestRect, &gViewRect)))
   {
     Di sposeWi ndow(gWi ndowPtr);
     DisposeHandle((Handle) docStrucHdl);
     return(MemError());
   HUnlock((Handle) docStrucHdl);
 }
  if(showWindow)
   ShowWi ndow(gWi ndowPtr);
 gCurrentNumberOfWindows ++;
  return(noErr);
                        ************** do0penFile
OSErr doOpenFile(FSSpec fileSpec, OSType documentType)
 0SErr
                     osError;
 SInt 16
                     fileRefNum;
 docStructureHandle docStrucHdl;
 if(osError = doNewDocWindow(false, documentType))
   return(osError);
 SetWTitle(gWindowPtr, fileSpec.name);
 if(osError = FSp0penDF(&fileSpec, fsRdWrPerm, &fileRefNum))
   Di sposeWi ndow(gWi ndowPtr);
   gCurrentNumberOfWindows --;
   return(osError);
  docStrucHdl = (docStructureHandle) GetWRefCon(gWindowPtr);
  (*docStrucHdl)->fileRefNum = fileRefNum;
  (*docStrucHdl)->fileFSSpec = fileSpec;
 if(documentType == 'TEXT')
   if(osError = doReadTextFile(gWindowPtr))
     return(osError);
 else if(documentType == 'PICT')
   if(osError = doReadPictFile(gWindowPtr))
     return(osError);
 ShowWi ndow(gWi ndowPtr);
 return(noErr);
  OSErr doCloseFile(WindowPtr windowPtr, docStructureHandle docStrucHdl,
                  NavAskSaveChangesAction action)
  NavAskSaveChangesResult reply = 0;
 NavEventUPP
                        navEventFunctionUPP;
 NavDi al ogOpti ons
                        di al og0pti ons;
 Str255
                        fileName:
 0SErr
                        osError:
 i f((*docStrucHdl)->wi ndowTouched)
   GetWTitle(windowPtr, fileName);
   BlockMoveData(fileName, dialogOptions.savedFileName, fileName[0]+1);
   navEventFunctionUPP = NewNavEventProc((ProcPtr) navEventFunction);
   osError = NavAskSaveChanges(&dialogOptions, action, &reply, navEventFunctionUPP, 0);
```

16B-30 Files

```
Di sposeRouti neDescri ptor(navEventFuncti onUPP);
   if(reply == kNavAskSaveChangesCancel)
     return((0SErr) reply);
    else if(reply == kNavAskSaveChangesSave)
     if(osError = doSaveCommand())
       return(osError);
 }
 if((*docStrucHdl)->fileRefNum != 0)
   if(!(osError = FSClose((*docStrucHdl)->fileRefNum)))
     osError = FlushVol(NULL, (*docStrucHdl)->fileFSSpec.vRefNum);
      (*docStrucHdl)->fileRefNum = 0;
  if((*docStrucHdl)->editStrucHdl)
    TEDi spose((*docStrucHdl)->editStrucHdl);
  if((*docStrucHdl)->pictureHdl)
    KillPicture((*docStrucHdl)->pictureHdl);
  DisposeHandle((Handle) docStrucHdl);
 return(osError);
// *************** dowriteFile
OSErr doWriteFile(WindowPtr windowPtr, Boolean newFile)
  docStructureHandle docStrucHdl;
 0SErr
                     osError;
  SInt32
                     fileRefNum;
  docStrucHdl = (docStructureHandle) GetWRefCon(windowPtr);
  fileRefNum = (*docStrucHdl)->fileRefNum;
  if((*docStrucHdl)->editStrucHdl)
   osError = doWriteTextData(windowPtr, fileRefNum);
  else if((*docStrucHdl)->pictureHdl)
    osError = doWritePictData(windowPtr, fileRefNum);
 if(osError == noErr)
   if(newFile)
     osError = doCopyResources(windowPtr);
  return(osError);
                              OSErr doReadTextFile(WindowPtr windowPtr)
  docStructureHandle docStrucHdl;
  SInt 16
                     fileRefNum;
  TEHandl e
                     textEditHdl;
  SInt32
                     numberOfBytes;
                    textBuffer;
  Handl e
  0SErr
                     osError;
  docStrucHdl = (docStructureHandle) GetWRefCon(windowPtr);
  fileRefNum = (*docStrucHdl)->fileRefNum;
  textEditHdl = (*docStrucHdl)->editStrucHdl;
  (*textEditHdl) - > txSize = 10;
  (*textEditHdl)->lineHeight = 15;
  SetFPos(fileRefNum, fsFromStart, 0);
  GetE0F(fileRefNum, &number0fBytes);
  if(numberOfBytes > 32767)
    numberOfBytes = 32767;
```

```
if(!(textBuffer = NewHandle((Size) numberOfBytes)))
   return(MemError());
  osError = FSRead(fileRefNum, &numberOfBytes, *textBuffer);
  if(osError == noErr || osError == eofErr)
    MoveHHi (textBuffer);
    HLockHi (textBuffer);
   TESetText(*textBuffer, numberOfBytes, (*docStrucHdl) ->editStrucHdl);
   HUnlock(textBuffer);
   Di sposeHandl e(textBuffer);
  else
   return(osError);
 return(noErr);
// ******************* doReadPictFile
OSErr doReadPictFile(WindowPtr windowPtr)
  docStructureHandle docStrucHdl;
  SInt 16
                     fileRefNum:
  SInt32
                     numberOfBytes;
  0SErr
                     osError;
  docStrucHdl = (docStructureHandle) GetWRefCon(windowPtr);
  fileRefNum = (*docStrucHdl)->fileRefNum;
  GetE0F(fileRefNum, &number0fBytes);
  SetFPos(fileRefNum, fsFromStart, 512);
  numberOfBytes -= 512;
 if(!((*docStrucHdl)->pictureHdl = (PicHandle) NewHandle(numberOfBytes)))
   return(MemError());
  osError = FSRead(fileRefNum, &numberOfBytes, *(*docStrucHdl)->pictureHdl);
  if(osError == noErr || osError == eofErr)
   return(noErr);
  else
   return(osError);
OSErr doWriteTextData(WindowPtr windowPtr, SInt16 tempFileRefNum)
  docStructureHandle docStrucHdl;
                     textEditHdl;
  TEHandl e
 Handl e
                     editText:
  SInt32
                     numberOfBytes;
  SInt 16
                     vol Ref Num;
  0SErr
                     osError:
 docStrucHdl = (docStructureHandle) GetWRefCon(windowPtr);
textEditHdl = (*docStrucHdl)->editStrucHdl;
  editText = (*textEditHdl)->hText;
  numberOfBytes = (*textEditHdl)->teLength;
  osError = SetFPos(tempFileRefNum, fsFromStart, 0);
  if(osError == noErr)
    osError = FSWrite(tempFileRefNum, &numberOfBytes, *editText);
  if(osError == noErr)
   osError = SetE0F(tempFileRefNum, number0fBytes);
  if(osError == noErr)
   osError = GetVRefNum(tempFileRefNum, &vol RefNum);
  if(osError == noErr)
   osError = FlushVol(NULL, vol RefNum);
  if(osError == noErr)
    (*docStrucHdl)->windowTouched = false;
 return(osError);
```

16B-32 Files

```
******** doWritePictData
OSErr doWritePictData(WindowPtr windowPtr, SInt16 tempFileRefNum)
  docStructureHandle docStrucHdl;
                       pi ctureHdl;
  Pi cHandl e
  SInt32
                       numberOfBytes, dummyData;
  SInt16
                       vol Ref Num;
  0SErr
                       osError:
  docStrucHdl = (docStructureHandle) GetWRefCon(windowPtr);
  pictureHdl = (*docStrucHdl)->pictureHdl;
  numberOfBytes = 512;
  dummyData = 0;
  osError = SetFPos(tempFileRefNum, fsFromStart, 0);
  if(osError == noErr)
    osError = FSWrite(tempFileRefNum, &numberOfBytes, &dummyData);
  numberOfBytes = GetHandleSize((Handle) (*docStrucHdl)->pictureHdl);
  if(osError == noErr)
    HLock((Handle) (*docStrucHdl)->pictureHdl);
    osError = FSWrite(tempFileRefNum, &numberOfBytes, *(*docStrucHdl) ->pictureHdl);
    HUnlock((Handle) (*docStrucHdl)->pictureHdl);
  if(osError == noErr)
    osError = SetEOF(tempFileRefNum, 512 + numberOfBytes);
  if(osError == noErr)
    osError = GetVRefNum(tempFileRefNum, &volRefNum);
  if(osError == noErr)
    osError = FlushVol (NULL, vol RefNum);
  if(osError == noErr)
    (*docStrucHdl)->windowTouched = false;
  return(osError);
                                             ****** doCopyResources
OSErr doCopyResources(WindowPtr windowPtr)
  docStructureHandle docStrucHdl;
  0SType
                       fileType;
  0SErr
                       osError;
  SInt16
                       fileRefNum;
  docStrucHdl = (docStructureHandle) GetWRefCon(windowPtr);
  if((*docStrucHdl)->editStrucHdl)
    fileType = 'TEXT';
  else if((*docStrucHdl)->pictureHdl)
    fileType = 'PICT';
  FSpCreateResFile(&(*docStrucHdl)->fileFSSpec, 'kkkB', fileType, smSystemScript);
  osError = ResError();
  if(osError == noErr)
    fileRefNum = FSpOpenResFile(&(*docStrucHdl)->fileFSSpec, fsRdWrPerm);
  if(fileRefNum > 0)
    osError = doCopyAResource('STR', -16396, gAppResFileRefNum, fileRefNum);
    if(fileType == 'PICT')
    {
      \label{lem:convergence} $$doCopyAResource('pnot', 128, gAppResFileRefNum, fileRefNum); $$doCopyAResource('PICT', 128, gAppResFileRefNum, fileRefNum); $$
  }
  else
    osError = ResError();
```

```
if(osError == noErr)
        CloseResFile(fileRefNum);
    osError = ResError();
    return(osError);
      OSErr doCopyAResource(ResType resourceType, SInt16 resourceID, SInt16 sourceFileRefNum,
                                              SInt16 destFileRefNum)
    Handle sourceResourceHdl;
    Str255 sourceResourceName;
    ResType ignoredType;
    SInt16 ignoredID;
    UseResFile(sourceFileRefNum);
    sourceResourceHdl = GetResource(resourceType, resourceID);
    if(sourceResourceHdl != NULL)
        GetResInfo(sourceResourceHdl, &ignoredID, &ignoredType, sourceResourceName);
        DetachResource(sourceResourceHdl);
        UseResFile(destFileRefNum);
        AddResource(sourceResourceHdl, resourceType, resourceID, sourceResourceName);
        if(ResError() == noErr)
            UpdateResFile(destFileRefNum);
    Rel easeResource(sourceResourceHdl);
    return(ResError());
                     pascal\ void\ navEventFunction (NavEventCallbackMessage\ callBackSelector, navEventCallbackMessage\ callBackS
                                                            NavCBRecPtr callBackParms, NavCallBackUserData callBackUD)
    WindowPtr windowPtr:
    windowPtr = (WindowPtr) callBackParms->eventData.event->message;
    switch(callBackSelector)
        case kNavCBEvent:
            switch(callBackParms->eventData.event->what)
                 case updateEvt:
                    if(((WindowPeek) windowPtr)->windowKind != kDialogWindowKind)
                        doUpdate((EventRecord *) callBackParms->eventData.event);
            break;
// ChooseAFol derDi al og. c
#include "Files2.h"
// ********* doChooseAFol derDi al og
OSErr doChooseAFolderDialog(FSSpec *theFileSpec)
    NavDi al ogOptions di al ogOptions;
    NavEvent UPP
                                         navEventFunctionUPP;
    0SErr
                                         osError:
    NavRepl yRecord
                                         navReplyStruc;
    FSSpec
                                          fileSpec;
```

16B-34 Files

```
AEDesc
                  resultDesc;
osError = NavGetDefaultDialogOptions(&dialogOptions);
GetIndString(dialog0ptions.message, rMiscStrings, sChooseAFolder);
navEventFunctionUPP = NewNavEventProc((ProcPtr) navEventFunction);
osError = NavChooseFolder(NULL, &navReplyStruc, &dialogOptions, navEventFunctionUPP, NULL, 0);
Di sposeRouti neDescri ptor(navEventFuncti onUPP);
if((navReplyStruc.validRecord) && (osError == noErr))
  if((osError = AECoerceDesc(&(navReplyStruc.selection),typeFSS,&resultDesc)) == noErr)
 {
    BlockMoveData(*resultDesc. dataHandle, &fileSpec, sizeof(FSSpec));
   FSMakeFSSpec(fileSpec. vRefNum, fileSpec. parID, fileSpec. name, theFileSpec);
 }
  AEDi sposeDesc(&resul tDesc):
 NavDi sposeReply(&navReplyStruc);
return(osError);
```

Demonstration Program Comments

Note: Navigation Services requires Mac OS 7.5 or later and Appearance Manager 1.0.1 or later. The Navigation Services shared library must be installed in the Extensions folder. On 680x0 systems, OpenTransportLib68K 1.3 or later must be installed in the Extensions folder.

When the program is run, the user should:

- Exercise the File menu by opening the supplied TEXT and PICT files, saving those files, saving those files under new names, closing files, opening the new files, attempting to open files which are already open, attempting to save files to new files with existing names, making open windows "touched" by choosing the first item in the Demonstration menu, reverting to the saved versions of files associated with "touched" windows, choosing Quit when "touched" and non-" touched" windows are open, and so on.
- Choose, via the Show pop-up menu button, the file types required to be displayed in the Open dialog.
- Choose the Choose a Folder item from the Demonstration menu to display the Choose a Folder dialog, and choose a folder using the Choose button at the bottom of the dialog. (The name of the chosen folder will be drawn in the bottom-left corner of the front window.)

The program may be run from within CodeWarrior to demonstrate responses to the File menu commands and the Choose a Folder dialog.

The built application, together with the supplied TEXT and PICT files, may be used to demonstrate the additional aspect of integrating the receipt of required Apple events with the overall file handling mechanism. To prove the correct handling of the required Apple events, the user should:

- Open the application by double-clicking the application icon, noting that a new document window is opened after the application is launched and the Open Application event is received.
- Double click on a document icon, or select one or more document icons and either drag those icons to the application icon or choose Open from the Finder's File menu, noting that the application is launched and the selected files are opened when the Open Documents event is received.
- Close all windows and double-click the application icon, noting that the application responds to the Re-open Application event by opening a new window.
- With several documents open, some with "touched" windows, choose Restart or Shut Down from the Finder's Special menu (thus invoking a Quit Application event), noting that,

Tiles 16B-35

for "touched" windows, the Save Changes alert box is presented asking the user whether the file should be saved before the shutdown process proceeds.

Files.h

#typedef

Each window created by the program will have an associated document structure, accessed via the window structure's refCon field. The docStructure structure will be used for document structures.

The editStrucHdl field will be assigned a handle to a TextEdit edit structure ('TEXT' files). The pictureHdl field will be assigned a handle to a Picture structure ('PICT' files). The fileRefNum and fileFSSpec fields will be assigned the file reference number and the file system specification structure of the file associated with the window. The windowTouched field will be set to true when a window has been made "touched", that is, when the associated document in memory has been modified by the user.

#define

After the usual constants relating to menus, windows, and alert boxes are established, additional constants are established a 'STR#' resource containing error strings, four specific error conditions, a 'STR#' resource containing the application's name and the message string for the Choose a Folder dialog box, and the 'open' resource containing the file types list. kMaxWindows is used to limit the number of windows the user can open.

Files.c

Files.c is simply the basic "engine" which supports the demonstration. There is little in this file which has not featured in previous demonstration programs.

Global Variables

gAppResFileRefNum will be assigned the file reference number of the application's resource fork.

main

The call to NavServicesAvailable determines whether the Navigation Services shared library is installed and running on the user's system. If it is, NavLoad is called to load the library, otherwise, an error alert is presented and the program terminates. (The call to NavLoad is optional. If the call is not made, the Navigation Services shared library will not load until your application calls a Navigation Services function, and will unload after the call completes. If the NavLoad call is made, you must call the NavUnload function before quitting so as to release the reserved memory.)

Routine descriptors for the required Apple events (less the Print Documents event) are created and a call is made to the application-defined function which installs the handlers. Also, the file reference number of the application's resource fork (which is opened automatically at application launch) is assigned to the global variable gAppResFileRefNum.

dolnstallAEHandlers

doInstallAEHandlers installs handlers for the Open Application, Re-Open Application, Open Documents, and Quit Application events. (Note that, so as to avoid the necessity to include application-defined printing functions in this program, a handler for the Print Documents event is not included in this demonstration.)

doMouseDown

Note that, in the inGoAway case, the constant kNavSaveChangesClosingDocument is passed in the call to doCloseCommand. This affects the text in the Save Changes alert box.

doUpdate

doUpdate performs such window updating as is necessary for the satisfactory execution of the demonstration aspects of the program.

16B-36 Files

doMenuChoice

If the Choose a Folder item in the Demonstration menu is chosen, the application-defined function which presents the Choose a Folder dialog box is called. This function returns userCanceledErr if the user clicked the Cancel push button in the dialog box. If an error other than userCanceledErr is returned, an error alert box is presented and the else block is bypassed.

The doChooseAFolderDialog function fills in the file system specification structure whose address is passed in its parameter. If a window is open, a rectangle in the bottom corner of the front window is erased and, if the user did not click the Cancel push button, the chosen folder's name, volume reference number, and parent directory ID are extracted from the file system specification structure and drawn in the bottom of the window.

doFileMenu

At the iClose case, kNavSaveChangesClosingDocument is passed in the call to doCloseCommand. This affects the wording in the Save Changes alert box. If doCloseCommand returns an error, and if that error is not kNavAskSaveChangesCancel (the user clicked the Cancel push button in the Save Changes alert box), an error alert box is presented.

At the iQuit case, kNavSaveChangesQuittingApplication is passed in the call to doQuitCommand. This affects the wording in the Save Changes alert box. If doQuitCommand returns an error, and if that error is not kNavAskSaveChangesCancel (the user clicked the Cancel push button in the Save Changes alert box), an error alert box is presented. If kNavAskSaveChangesCancel was not returned, NavUnload is called to release the memory reserved for the Navigation Services shared library, and gDone is set to true to cause program termination.

doErrorAlert

doErrorAlert handles errors, invoking an appropriate alert box (caution or stop) advising of the nature of the problem by error code number or straight text. Note that he program will only be terminated if the Navigation Services library is not installed and running, or in the case of the memFullErr error (no more space in the application heap).

doTouchWindow

doTouchWindow is called when the user chooses the Touch Window item in the Demonstration menu. Changing the content of the in-memory version of a file is only simulated in this program. The text "WINDOW TOUCHED" is drawn in window and the windowTouched field of the document structure is set to true.

doOpenAppEvent, doOpenDocsEvent, and doQuitAppEvent

The handlers for the required Apple events are essentially identical to those in the demonstration program at Chapter 10 - Required Apple Events.

Most programs should simply open a new untitled window on receipt of an Open Application event. Accordingly, doOpenAppEvent simply calls the same function (doNewCommand) as is called when the user chooses New from the File menu.

On receipt of a Re-Open Application event, if no windows are currently open, doNewCommand is called to open a window.

The demonstration program supports both 'TEXT' and 'PICT' files. On receipt of an Open Application event, it is thus necessary to determine the type of each file specified in the event. Accordingly, within doOpenDocsEvent, the call to FSpGetFInfo returns the Finder information from the volume catalog entry for the file relating to the specified FSSpec structure. The fdType field of the FInfo structure "filled-in" by FSpGetFInfo contains the file type. This, together with the FSSpec structure, is then passed in the call to doOpenFile. (doOpenFile is also called when the user chooses Open from the File menu.)

Within the function doQuitAppEvent, the while loop entered at repeats for each open window. Within the loop, doCloseCommand is called, passing kNavSaveChangesQuittingApplication (which affects the wording in the Save Changes alert box). doCloseCommand, in turn, calls doCloseFile. doCloseFile presents the Save Changes alert box. If an error is returned by this sequence, and if the user did not click the Cancel push button in the alert box, the error handler is called. If the user clicked the Cancel button, it is necessary to interrupt the sequence of closing all open windows and re-enter the main event loop.

When the while loop eventually exits, NavUnload is called to release the memory reserved for the Navigation Services shared library, and gDone is set to true, causing the program to terminate.

NewOpenCloseSave.c

Global Variables

gWindowPtr is assigned the pointer to the graphics port of each new window as it is opened. gCurrentNumberofWindows keeps a count of the number of windows opened. gDestRect and gViewRect are used to set the destination and view rectangles for the edit structures associated with 'TEXT' files.

doNewCommand

doNewCommand is the first of the file-handling functions. It is called when the user chooses New from the File menu and when an Open Application or Re-Open Application event is received.

Since this demonstration does not support the actual entry of text or the drawing of graphics, the document type passed to doNewDocWindow is immaterial. The document type 'TEXT' is passed in this instance simply to keep doNewDocWindow happy.

doOpenCommand

doOpenCommand is called when the user chooses Open from the File menu. This function uses Navigation Services functions.

NavGetDefaultDialogOptions initialises the specified NavDialogOptions structure with the

GetIndString assigns the application's name to the clientName field of the NavDialogOptions structure. This will then appear in the dialog box's title bar. The next line reads in the 'open' resource containing the file type list and assigns the handle to a variable of type NavTypeListHandle.

NewNavEventProc creates a routine descriptor for an application-defined event-handling function, which is included so as to make the Open dialog box movable and resizable.

The call to NavGetFile displays the Open dialog box. NULL in the first parameter means that the dialog box will open at the last location visited during the last call to NavGetFile. The second parameter will receive the information required by the application. Since the default options are being used, multiple file selection is allowed. The universal procedure pointer to the event-handling function is passed in the fourth parameter. No preview function is used. No filter function is used. The handle to the file type list is passed in the second last parameter.

When the Open dialog is dismissed by the user and NavGetFile returns, the first action is to dispose of the routine descriptor.

The main if block executes only if the user clicked the Open push button (or pressed the Return or Enter keys) and no error was returned. The first action is to call AECountItems to count the number of descriptor structures in the descriptor list returned in the selection field of the NavReplyRecord structure, and which is created from FSSpec references to items selected in the Open dialog box.

The following for loop repeats for each of the descriptor structures. AEGetNthDesc gets a descriptor structure of type file system specification into the AEDesc structure resultDesc. BlockMoveData copies the data from the dataHandle field of that structure to a local variable of type FSSpec. This file system specification is then passed in the first parameter of a call to FspGetFInfo, allowing the file type to be ascertained. The file system specification and file type are then passed in a call to the application-defined function doOpenFile, which creates a new window and reads in the file. Before the next iteration of the loop (if any), AEDisposeDesc is called to deallocate the memory occupied by the descriptor structure.

With all of the selected files read in, NavDisposeReply must be called to release the memory allotted for the NavReplyReply structure. The call to ReleaseResource frees the memory occupied by the file type list.

If the user clicks the Cancel push button in a Navigation Services dialog box, the relevant Navigation Services function returns userCanceledErr. In this event, the variable osError is set to noErr before doOpenCommand returns.

doCloseCommand

doCloseCommand is called when the user chooses Close from the File menu or clicks in the window's go-away box. It is also called successively for each open window when a Quit Application event is received.

16B-38 Files

The first two lines get the WindowPtr for the front window and establish whether the front window is a document window or a modeless dialog box.

If the front window is a document window, the handle to the window's document structure is retrieved from the window structure's refCon field. The WindowPtr and this handle are then passed to the application-defined function doCloseFile, together with the Navigation Services constant received in doCloseCommand's action parameter. If the window is "touched", doCloseFile presents the Save Changes alert box asking the user whether the document should be saved before it is closed. If the user clicks the Cancel push button of that alert box, doCloseFile returns kNavAskSaveChangesCancel, in which case doCloseCommand returns kNavAskSaveChangesCancel. If the user clicks either the OK or Don't Save push buttons in the alert box, and if doCloseFile returns no error, the window is closed as the final act in closing the file, and the global variable which keeps track of the number of open windows is decremented.

No modeless dialog boxes are used by this program. However, if the front window was a modeless dialog box, the appropriate action would be taken at the second case.

doSaveCommand

doSaveCommand is called when the user chooses Save from the File menu. It may also be called by doCloseFile if the user is attempting to close a "dirty" window.

The first two lines get the WindowPtr for the front window and retrieve the handle to that window's document structure. If a file currently exists for the document in this window, the application-defined function doWriteFile is called, otherwise the application-defined function doSaveAsCommand is called.

doSaveAsCommand

doSaveAsCommand is called when the user chooses Save As... from the File menu. It is also called by doSaveCommand if the user chooses Save when the front window contains a document for which no file currently exists. This function uses Navigation Services functions.

 $NavGetDefault Dialog 0 ptions\ initialises\ the\ specified\ NavDialog 0 ptions\ structure\ with\ the\ defaults.$

The next two lines get the handle to the window's document structure, which will be required later. GetWTitle gets the window's title into the savedFileName field of a NavDialogOptions structure. This will be the default name for the saved file and will appear in the Name edit text field in the Save dialog box. The call to GetIndString copies the application's name to the clientName field of the NavDialogOptions structure. This will then appear in the dialog box's title bar.

NewNavEventProc creates a routine descriptor for an application-defined event-handling function, which is included so as to make the Save dialog box movable and resizable. The next four lines retrieve the file type from the document structure for the front window.

The call to NavPutFile displays the Open dialog box. NULL in the first parameter means that the dialog box will open at the last location visited during the last call to NavPutFile. The second parameter will receive the information required by the application. The file type and creator are passed in the fifth and sixth parameters. When the user dismisses the dialog box, NavPutFile returns, at which point the routine descriptor for the event-handling function is disposed of.

The main if block executes only if the user clicked the Save push button (or pressed the Return or Enter keys) and no error was returned. A descriptor list is returned in the selection field of the NavReplyRecord structure. AEGetNthDesc is called to get the descriptor structure (of type file system specification) from that descriptor list into the AEDesc structure resultDesc. If this call is successful, BlockMoveData is called to copy the data from the dataHandle field of that structure to a local variable of type FSSpec.

If the value in the replacing field of the NavReplyRecord structure indicates that the file is not being replaced, FSpCreate is called to create a new file of the specified type and with the application's signature as the specified creator. If this call is not successful, the descriptor structure is disposed of, the NavReplyRecord structure is disposed of, and the function returns.

The file system specification structure returned by the FSpCreate call is assigned to the fileFSSpec field of the window's document structure. If a file currently exists for the document, that file is closed by the call to FSClose. The data fork of the newly created file is then opened by a call to FSpOpenDF, the fileRefNum field of the document structure is assigned the file reference number returned by FSpOpenDF, the window's title is set to the new file's name, and the application-defined function doWriteFile is called to write the document to the new file.

AEDisposeDesc is called to deallocate the memory occupied by the descriptor structure. NavCompleteSave is called to complete the save operation. With the save completed, NavDisposeReply must be called to release the memory allotted for the NavReplyReply structure.

If the user clicks the Cancel push button in a Navigation Services dialog box, the relevant Navigation Services function returns userCanceledErr. In this event, the variable osError is set to noErr before doOpenCommand returns.

doRevertCommand

doRevertCommand is called when the user chooses Revert to Saved from the File menu. This function uses a Navigation Services function.

The first line creates a routine descriptor for an application-defined event handling function, which will be used so as to cause the Discard Changes alert box to be movable.

The next two lines get the WindowPtr for the front window and the handle to that window's document structure. The call to GetWTitle gets the window's title (that is, the filename). BlockMoveData copies that filename to the savedFileName field of a NavDialogOptions structure. This will be used in the Discard Changes alert box's text.

The call to NavAskDiscardChanges displays the Discard Changes alert box. The information required by the application is received in the reply parameter. When the user dismisses the alert box, the routine descriptor is disposed of.

If the user clicked the OK push button, the window's content area is erased and the appropriate application-defined function (doReadTextFile or doReadPictFile) is called depending on whether the file type is 'TEXT' or 'PICT'. In addition, the window's "touched" field in the document structure is set to false and InvalRect is called to force a redraw of the window's content region.

doQuitCommand

doQuitCommand is called when the user chooses Quit from the File menu and when a Quit Application event is received.

The while loop continues to execute until no more windows remain open. On each pass through the loop, doCloseCommand is called to manage the process of closing (and, where necessary, saving) all documents and disposing of the associated windows.

doNewDocWindow

doNewDocWindow is called by doNewCommand, doOpenFile and the Open Application event handler. It creates a new window and associated document structure.

If the current number of open windows is the maximum allowable by this program, the function immediately exits, passing an error code which will cause an advisory error alert box to be displayed.

The call to GetNewCWindow opens a new window. SetPort sets that window's graphics port as the current port for drawing.

The call to NewHandle allocates memory for the window's document structure. If this call is not successful, the window is disposed of and the function returns with the error code returned by MemError.

The call to SetWRefCon assigns the handle to the document structure to the window structure's refCon field. The next four lines initialise fields of the document structure.

If the document type is 'TEXT', the if block executes, creating a TextEdit edit structure and assigning a handle to that structure to the editRec field of the document structure. (Note that the processes here are not explained in detail because TextEdit and edit structures are not central to the demonstration. For the purposes of the demonstration, it is sufficient to understand that the text data retrieved from, and saved to, disk is stored in a TextEdit edit structure. TextEdit is addressed in detail at Chapter 19 - Text and TextEdit.)

If the Boolean value passed to doNewDocWindow was set to true, the call to ShowWindow makes the window visible, otherwise the window is left invisible. The penultimate line increments the global variable which keeps track of the number of open windows.

doOpenFile

doOpenFile is called by doOpenCommand and the Open Documents event handler, which pass to it the file system specification structure and document type. doOpenFile opens a new document window and calls the application-defined functions which read in the file.

The call to doNewDocWindow opens a new window and creates an associated document structure. SetWTitle sets the window's title. FSpOpenDF opens the file's data fork. If this call is not successful, the window is disposed of and the function returns. The next three lines assign the file reference number and file system specification structure to the relevant fields of the document structure.

The next block calls the appropriate function for reading in the file, depending on whether the file type is of type 'TEXT' or 'PICT'. If the file is read in successfully, ShowWindow makes the window visible.

doCloseFile

doCloseFile is called by doCloseCommand. doCloseFile does not allow a "touched" window to be closed without offering the user the option of first saving the associated document to file. This function uses a Navigation Services function.

The first if block executes only if the window has been "touched". First, GetWTitle gets the window's title, which BlockMoveData copies to the savedFileName field of a NavDialogOptions structure. This will be used in the alert box's text. A routine descriptor is created for an event-handling function, which is used so as to cause the Save Changes alert box to be movable.

NavAskSaveChanges is then called to display the Save Changes alert box. The information required by the application is received in the reply parameter. The routine descriptor is disposed of when NavAskSaveChanges returns.

If the Cancel push button was clicked, NavAskSaveChanges returns kNavAskSaveChangesCancel, in which case doCloseFile returns, returning kNavAskSaveChangesCancel to the calling function. If OK push button was clicked, the application-defined function doSaveCommand is called to save the file.

If the document has a file, FSClose closes the file, and FlushVol stores to disk all unwritten data currently in the volume buffer.

If the document is a text document, the text edit structure is disposed of. If it is a picture document, the Picture structure is disposed of. Finally, the document structure is disposed of.

doWriteFile

doWriteFile is called by doSaveCommand and doSaveAsCommand. In conjunction with two supporting application-defined functions, it writes the document to disk. Note that, unlike the function of the same name in the demonstration program Files1, the "safe-save" methodology is not used.

The first two lines retrieve a handle to the window's document structure and the file reference number from the document structure.

At the next four lines, the appropriate application-defined function is called to write the document's data to its file.

If the file is a newly created file, the application-defined function doCopyResources is called to copy the missing application name string resource from the resource fork of the application file to the resource fork of the new document file. If the file type is 'PICT', a 'pnot' resource and associated 'PICT' resource is also copied to the resource fork.

doReadTextFile

doReadTextFile is called by doOpenFile and doRevertCommand to read in data from an open file of type 'TEXT'.

The first two lines retrieve the file reference number from the document structure.

The next three lines retrieve the handle to the TextEdit edit structure from the document structure and modify the text size and line height fields of the edit structure.

SetFPos sets the file mark to the beginning of the file. GetEOF gets the number of bytes in the file. If the number of bytes exceeds that which can be stored in a TextEdit edit

structure (32,767), the number of bytes which will be read from the file is restricted to 32,767.

NewHandle allocates a buffer equal to the size of the file (or 32,767 bytes if the preceding if statement executed). FSRead reads the data from the file into the buffer. MoveHHi and HLockHi move the buffer high in the heap and lock it preparatory to the call to TESetText. TESetText copies the text in the buffer into the existing hText handle of the TextEdit edit structure. The buffer is then unlocked and disposed of.

(Note: TextEdit is addressed in detail at Chapter 19 - Text and TextEdit.)

doReadPictFile

doReadPictFile is called by doOpenFile and doRevertCommand to read in data from an open file of type 'PICT'.

The first two lines retrieve the file reference number from the document structure. GetEOF gets the number of bytes in the file. SetFPos sets the file mark 512 bytes (the size of a 'PICT' file's header) past the beginning of the file, and the next line subtracts the header size from the total size of the file. NewHandle allocates memory for the Picture structure and FSRead reads in the file's data.

doWriteTextData

doWriteTextData is called by doWriteFile to write text data to the specified file.

The first two lines retrieve the handle to the TextEdit edit structure from the document structure. The number of bytes of text is then retrieved from the teLength field of the text edit structure.

SetFPos sets the file mark to the beginning of the file. FSWrite writes the specified number of bytes to the file. SetEOF adjusts the file's size. FlushVol stores to disk all unwritten data currently in the volume buffer.

The penultimate line sets the windowTouchedfield of the document structure to indicate that the document data on disk equates to the document data in memory.

doWritePictData

 ${\tt doWritePictData}\ is\ called\ by\ doWriteFile\ to\ write\ picture\ data\ to\ the\ specified\ file.$

The first two lines retrieve the handle to the relevant Picture structure from the document structure. SetFPos sets the file mark to the start of the file. FSWrite writes zeros in the first 512 bytes (the size of a 'PICT' file's header). GetHandleSize gets the size of the Picture structure and FSWrite writes the bytes in the Picture structure to the file. SetEOF adjusts the file's size and FlushVol stores to disk all unwritten data currently in the volume buffer.

The penultimate line sets the windowTouchedfield of the document structure to indicate that the document data on disk equates to the document data in memory.

doCopyResources

doCopyResources is called by doWriteFile when a newly created file has been written to for the first time. It copies the missing application name string resource from the resource fork of the application file to the resource fork of the new file. If the file type is PICT, a 'pnot' resource and associated 'PICT' resource is also copied.

The first line retrieves a handle to the file's document structure. The next four lines establish the file type involved. FSpCreateResFile creates the resource fork in the new file and FSpOpenResFile opens the resource fork. The application-defined function for copying specified resources between specified files (doCopyAResource) is then called to copy the missing application name string resource from the resource fork of the application file to the resource fork of the new file. If the file type is 'PICT', a 'pnot' resource and associated 'PICT' resource is also copied so as to provide a preview for 'PICT' files in the Open dialog box. (Of course, in a real application, the 'pnot' and 'PICT' resource would be created by the application for each separate 'PICT' file.)

CloseResFile closes the resource fork of the new file.

doCopyAResource

doCopyAResource copies specified resources between specified files. In this program, it is called only by doCopyResources.

UseResFile sets the application's resource fork as the current resource file. GetResource reads the specified resource into memory.

GetResInfo, given a handle, gets the resource type, ID and name. (Note that this line is included only because of the generic nature of doCopyResource. The calling function has passed doCopyResource the type and ID in this instance.)

DetachResource removes the resource's handle from the resource map without removing the resource from memory, and converts the resource handle into a generic handle. UseResFile makes the new file's resource fork the current resource file. AddResource makes the now arbitrary data in memory into a resource, assigns a resource ID, type and name to that resource, and inserts an entry in the resource map for the current resource file. UpdateResFile then writes the resource map and data to disk.

navEventFunction

 $navEventFunction\ is\ the\ event-handling\ function\ used\ by\ the\ Navigation\ Services\ dialog\ and\ alert\ boxes.$

The formal parameter callBackSelector is a constant indicating which type of call Navigation Services is making to navEventFunction. One such constant is kNavCBEvent, which indicates that an event has occurred. callBackParms is a pointer to a structure of type NavCBRec. The event's event structure resides in the eventData field of the NavCBRec structure.

At the first line, the window's WindowPtr is retrieved from the event structure's message field. At the kNavCBEvent case, the event type is extracted from the event structure's what field. If it is an update event, and if it is not for a Navigation Services dialog box (the application does not open any other dialog boxes), the application's window updating function doUpdate is called.

ChooseAFolderDialog.c

doChooseAFolderDialog

 $do Choose A Folder \, Dialog \,\, is \,\, called \,\, when \,\, the \,\, user \,\, chooses \,\, the \,\, Choose \,\, a \,\, Folder \,\, Dialog \,\, item \,\, in \,\, the \,\, demonstration \,\, menu. \,\, This \,\, function \,\, uses \,\, Navigation \,\, Services \,\, functions.$

NavGetDefaultDialogOptions initialises the specified NavDialogOptions structure with the defaults. GetIndString copies a string to the message field of a NavDialogOptions structure. This will appear immediately below the browser list in the dialog box.

The next line creates a routine descriptor for the event-handling function.

NavChooseFolder displays the Choose a Folder dialog box. When the user dismisses the dialog box, NavChooseFolder returns, at which time the routine descriptor is disposed of.

The if block executes if the user clicked the Choose push button. AECoerceDesc coerces the descriptor structure in the descriptor list returned in the selection field of the NavReplyRecord structure to a descriptor of type file system specification, the resulting descriptor being assigned to the AEDesc structure resultDesc. If this call is successful, BlockMoveData is called to copy the data from the dataHandle field of that structure to a local variable of type FSSpec.

When Navigation Services passes your application an AEDesc structure of type typeFSS describing a directory, the name field is empty and the parID field contains the directory ID of that directory, not the ID of the parent directory. In this demonstration, the volume reference number and directory ID are passed in a call to FSMakeFSSpec, which fills in the fields of the FSSpec record pointed to by the fourth parameter. In the function doMenuChoice, the contents of the fields of this FSSpec structure (the directory name, its parent directory ID, and the volume reference number) are drawn in the bottom of the front window.