16B

MORE ON FILES — NAVIGATION SERVICES

Includes Demonstration Program Files2

Introduction

Navigation Services, which was introduced with Mac OS 8.5[1] as an alternative to, and ultimately as a replacement for, the Standard File Package described at Chapter 16A, 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.
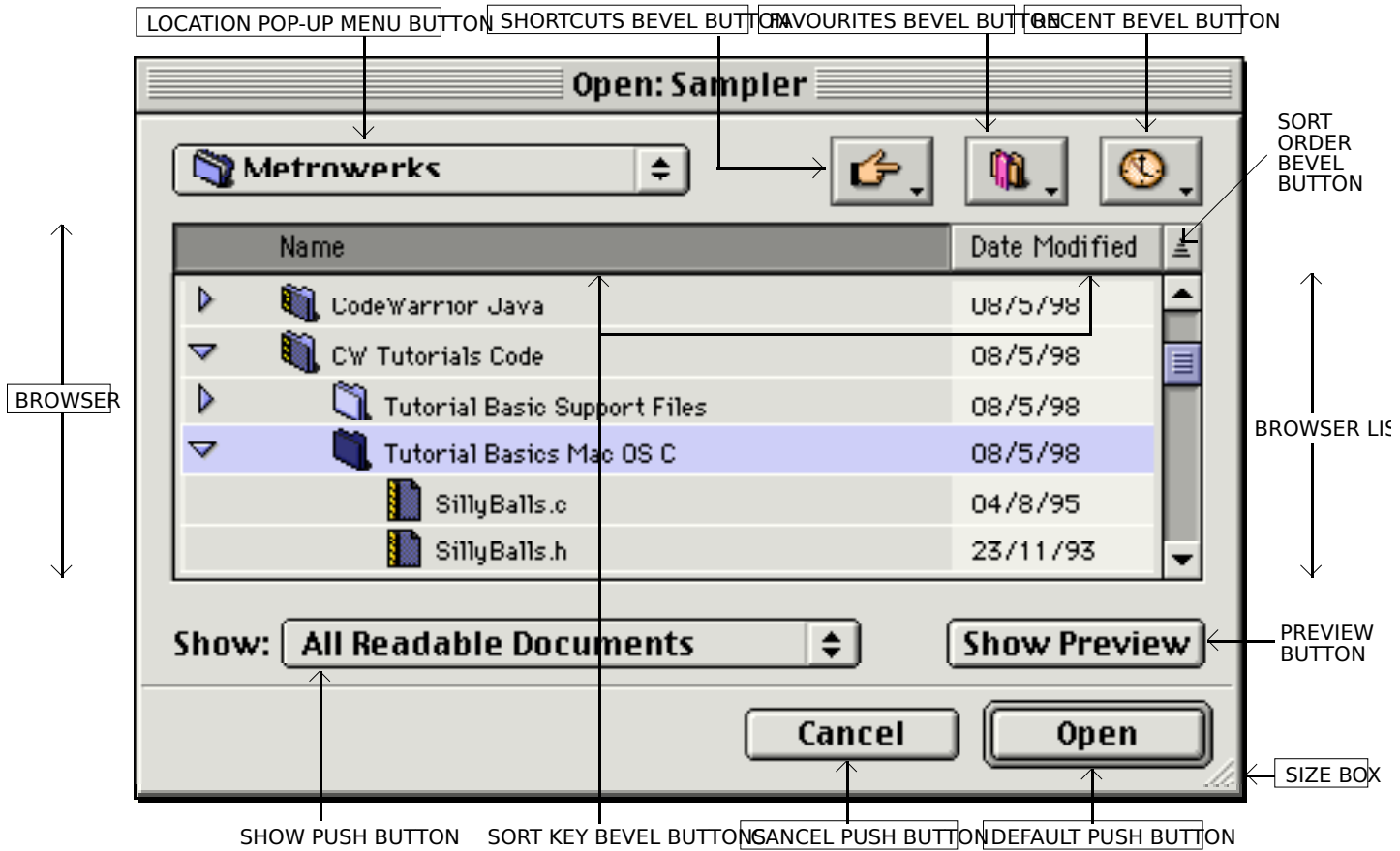
Note: The Open dialog box is shown. The names of the user interface elements common to all primary Navigation

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

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.

SHORTCUTS Allows quick navigation to any mounted storage volume, or straight to the desktop. If ejectable volumes are mounted, an Eject item appears at the bottom of the menu.

▦ **Desktop**

⌂ **Macintosh HD**
⌂ **SmartSet 3**

Eject "SmartSet 3"

RECENT The number of items in each section of the Recent menu will not exceed the number set in the Apple Menu Options dialog box.

📄 **Chapter_15**
📄 **chapter_8.pdf**

Documents section (Open dialog only)

📁 **Macintosh C**
📁 **Macintosh Pascal**
⌂ **SmartSet 2**

Folders and Volumes section

FAVOURITES The Add to Favorites command allows the the currently selected item to this menu. from Favorites command opens a dialog with the user to remove items from the menu.

Add to Favorites
Remove from Favorites...

Command section

📄 **Chapter_16B**
📄 **chapter_16b.pdf**
📄 **chapter_16b.html**

Document section (Open dial only)

⌂ **Macintosh HD**
📁 **Metrowerks**

Folders an Volumes section

As an alternative to the Add to Favorites command, item to the Favourites menu by dragging the file or browser list or desktop to the Favourites bevel butto

FIG 2 - SHORTCUTS, FAVOURITES, AND RECENT BEVEL BUTTON MENUS (TYPICAL)

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 <small>nil</small> in the <small>defaultLocation</small> parameter of functions such as <small>NavGetFile</small> (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 (<small>AEDesc</small>) for the new location in the <small>defaultLocation</small> parameter.

Opening Files

The function <small>NavGetFile</small> 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 <small>Select All</small> item in the application's <small>Edit</small> menu, the user can open multiple files.[2]

The Show Pop-up Menu

The <small>Show</small> 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 <small>NavGetFile</small>, and by services in the Translation Manager. (See Fig 3.)
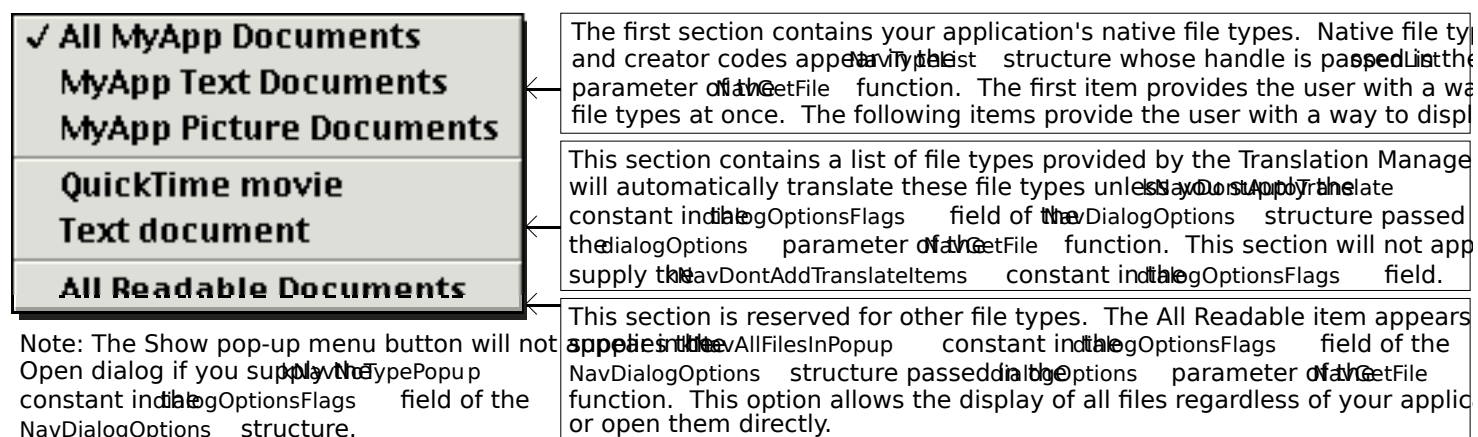
The first section contains your application's native file types. Native file types and creator codes appear in a <small>NavTypeList</small> structure whose handle is passed in the <small>typeList</small> parameter of the <small>NavGetFile</small> function. The first item provides the user with a way to file types at once. The following items provide the user with a way to display

This section contains a list of file types provided by the Translation Manager will automatically translate these file types unless you supply the <small>kNavDontAutoTranslate</small> constant in the <small>dialogOptionsFlags</small> field of the <small>NavDialogOptions</small> structure passed the <small>dialogOptions</small> parameter of the <small>NavGetFile</small> function. This section will not app supply the <small>kNavDontAddTranslateItems</small> constant in the <small>dialogOptionsFlags</small> field.

This section is reserved for other file types. The All Readable item appears applies the <small>kNavAllFilesInPopup</small> constant in the <small>dialogOptionsFlags</small> field of the <small>NavDialogOptions</small> structure passed in the <small>dialogOptions</small> parameter of the <small>NavGetFile</small> function. This option allows the display of all files regardless of your applic or open them directly.

Note: The Show pop-up menu button will not appear in the Open dialog if you supply the <small>kNavNoTypePopup</small> constant in the <small>dialogOptionsFlags</small> field of the <small>NavDialogOptions</small> structure.

**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 <small>All Readable Files</small> if you do not assign the name of your application to the <small>clientName</small> field of the <small>NavDialogOptions</small> structure (see below) passed in the <small>dialogOptions</small> parameter of the <small>NavGetFile</small> function.

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

---

[2] Folders and volumes cannot be multiple-selected.

[3] The kind strings from your application's <small>'kind'</small> resource also appear in the Kind column in Finder window list views.

RESORCERER 'kind' RESOURCE EDITING WINDOW
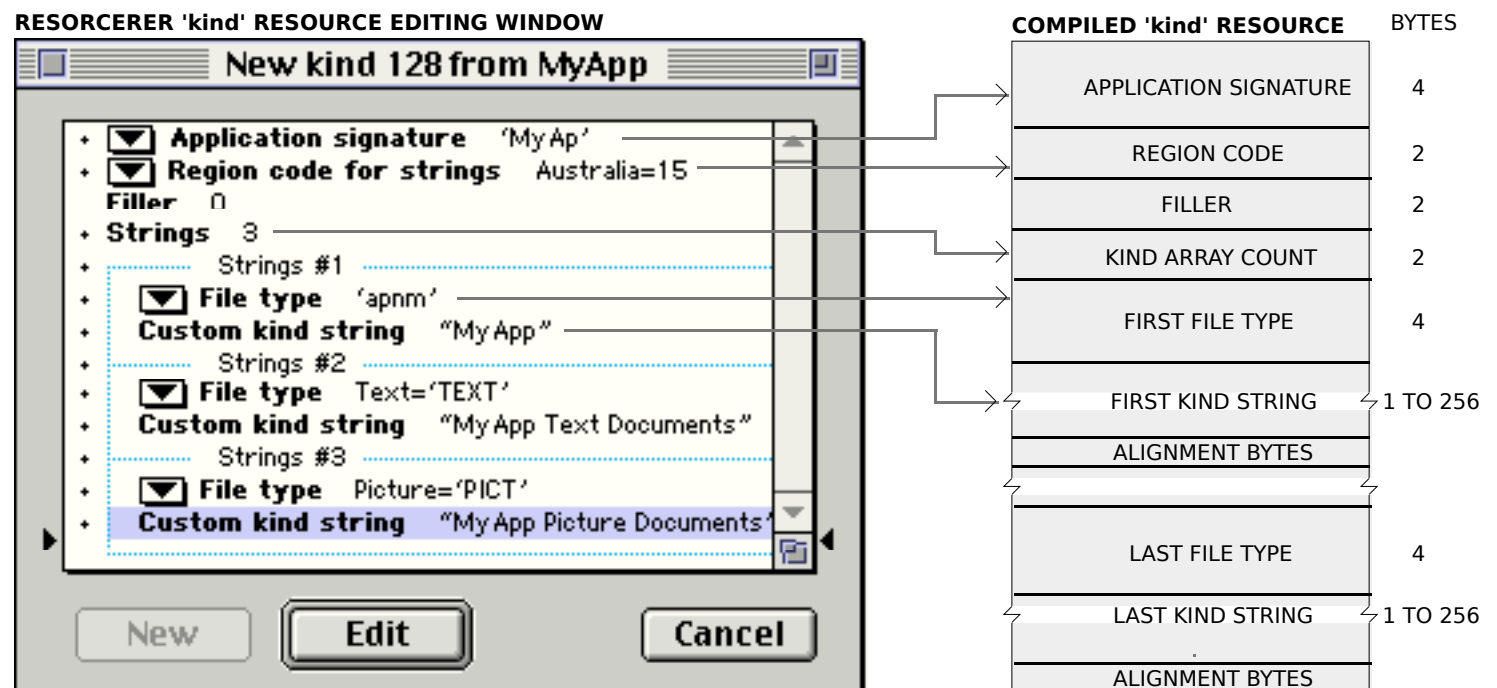
Note: The special file type 'apnm' has been included so that, whenever Navigation Services encounters a document that type has not been included in the 'kind' resource, a kind string in the form "<application name> document" will be genera

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

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 NavTypeList structure is as follows:

```
NavTypeList = RECORD
    componentSignature:     OSType;                 { Your application signature. }
    reserved :   INTEGER;
    osTypeCount:            INTEGER;                { How many file types will be defined. }
    osType:                 ARRAY [0..0] OF OSType; { A list of file types your application
                                                      can open.}

    END;
```

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.
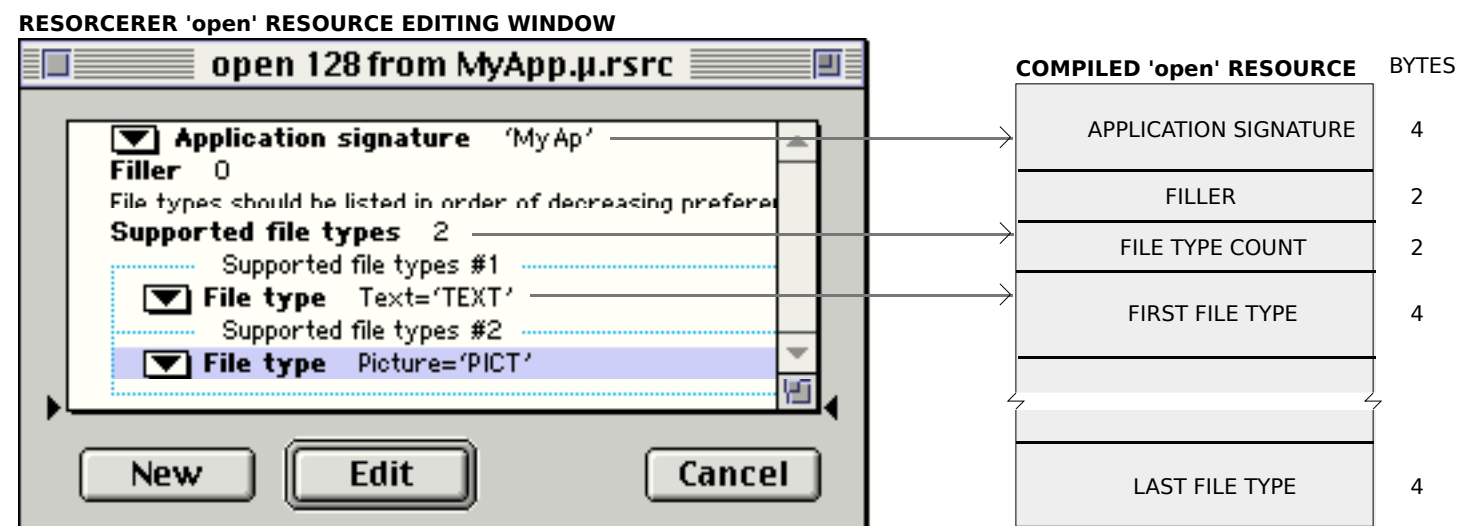
RESORCERER 'open' RESOURCE EDITING WINDOW

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

The NavDialogOptions Structure

The NavDialogOptions record is as follows:

```
NavDialogOptions = RECORD
    version:                    UInt16;
    dialogOptionFlags:          NavDialogOptionFlags;
    location:                   Point;
    clientName:                 Str255;
    windowTitle:                Str255;
    actionButtonLabel:          Str255;
    cancelButtonLabel:          Str255;
    savedFileName:              Str255;
    message:                    Str255;
    preferenceKey:              UInt32;
    popupExtension:             Handle;
    reserved:                   PACKED ARRAY [0..493] OF CHAR;
END;
```

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. • Automatic resolution of aliases. • Support for file previews. • 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.
kNavAllFilesInPopup	Add "All Files" menu item.
kNavAllowStationery	Allow stationery files.
kNavAllowPreviews	Allow previews.
kNavAllowMultipleFiles	Allow multiple selection.
kNavAllowInvisibleFiles	Show invisible objects.
kNavDontResolveAliases	Don't resolve aliases.
kNavSelectDefaultLocation	Make default location the browser selection.
kNavSelectAllReadableItem	Make All Readable Items default selection.

location   The upper-left location of the dialog box, in global coordinates. If the dialogOptionFlags field is NULL or the coordinate value is (-1,-1), then the dialog box appears in the same location as when last closed. The size and location of the dialog box is persistent, but defaults to opening in the middle of the main screen if any portion is not visible when opened at the persistent location and size.

clientName   A string that identifies your application in the dialog box window title.

windowTitle   A string that you can provide to override the default window title.

actionButtonLabel   An alternative button title for the dialog box's action push button. If you do not specify a label, the push button will use the default label (Open or Save, for example.)

cancelButtonLabel   An alternative button title for the Cancel push button in dialog boxes.

savedFileName   The default file name for a saved file.

message   The string for the banner, or prompt, below the browser list. This message can provide more descriptive instructions for the user. If you don't provide a message string, the browser list will 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.

popupExtension   A handle to one or more structures of type NavMenuItemSpec used to add extra menu items to the Show pop-up menu in an Open dialog box or the Format pop-up menu in Save dialog boxes. Using NavMenuItemSpec structures allows your application to add additional document types to be opened or saved, or different ways of saving a file (with or without line breaks, for example).

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:

```
NavReplyRecord = RECORD
    version:                UInt16;
    validRecord:            BOOLEAN;
    replacing:              BOOLEAN;
    isStationery:           BOOLEAN;
    translationNeeded:      BOOLEAN;
    selection:              AEDescList;
    keyScript:              ScriptCode;
    fileTranslation:        ^FileTranslationSpecPtr;
    reserved1:              UInt32;
    reserved:               PACKED ARRAY [0..230] OF CHAR;
END;
```

Field Descriptions

validRecord   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.

keyScript — 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 NavDisposeReply.

Saving Files

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

The New push button enables the user to create a new folder for saving a document.

When the user selects a f[...]
the default push button tit[...]
toggles from Save to Ope[...]

When the user selects the [...]
Name editable text field, t[...]
default push button title r[...]
to Save.

When no filename is displ[...]
in the Name editable text [...]
the default push button is [...]
disabled.

The first item is defined by the document type specified by fileType and fileCreator parameters of the NavPutFile function. The item's menu title is obtai[...] the Translation Manager.

After setting the first item, Navigation Services calls the Translation Manager [...] display subsequent menu items describing alternative file types.

The Stationery Option item displays the Stationery option dialog box, which [...] whether a new document or a copy of a document should be saved as a do[...]

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

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

Choosing A Folder

The function NavChooseFolder displays a dialog box that prompts the user to choose a folder (see Fig 7.)
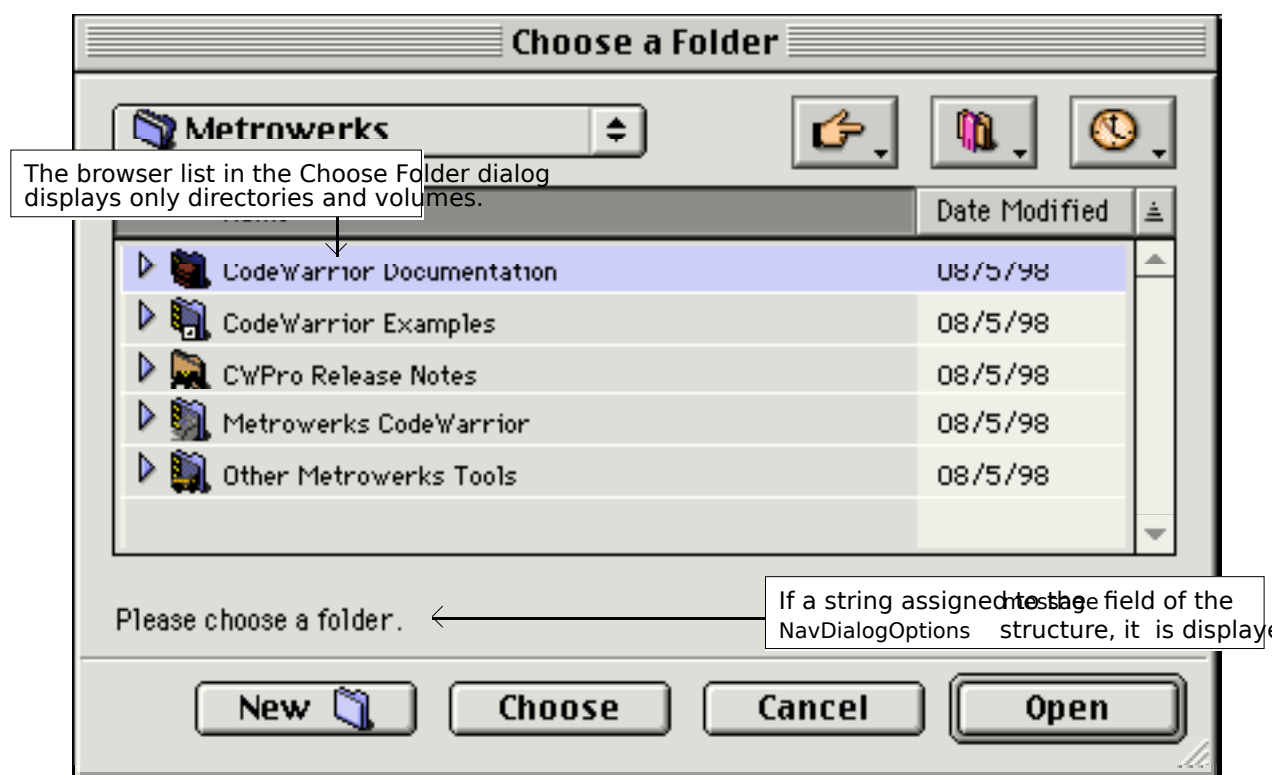
FIG 7 - CHOOSE A FOLDER DIALOG BOX

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 NavDialogOptions 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.
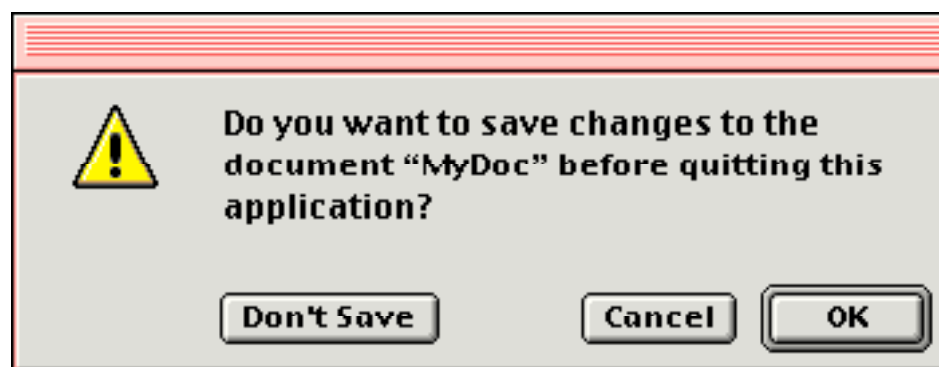
FIG 8 - STANDARD SAVE CHANGES ALERT BOX (QUITTING APPL

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 NavAskDiscardChanges.
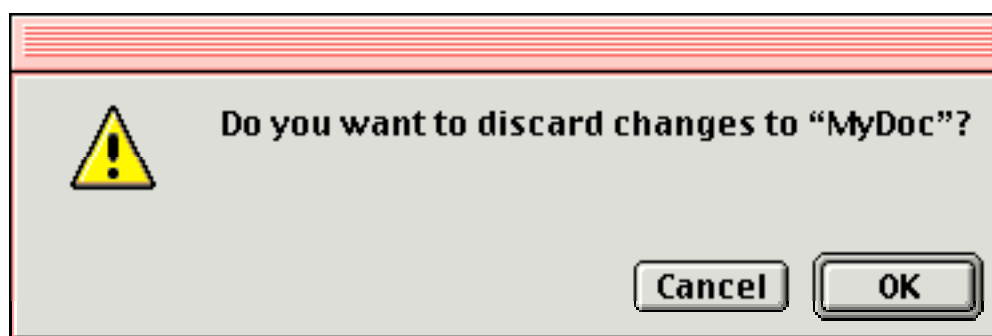
FIG 9 - DISCARD CHANGES ALERT BOX

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:

```
procedure  myEventFunction(callBackSelector : NavEventCallbackMessage;
                           callBackParms : NaVCBRecPtr;
                           callBackUD : NavCallBackUserData);
```

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 NavTypeList structure and a filter function if you wish, but be aware that your filter function is directly affected by the NavTypeList structure. For example, if the NavTypeList 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:

```
function MyFilterFunction(var theItem: AEDesc; info: UNIV Ptr;
                          callBackUD: NavCallBackUserData;
                          filterMode: NavFilterModes): boolean;

     var
     theErr : OSErr;
     display : boolean;
     theInfo : NavFileOrFolderInfoPtr;

     begin
     theErr := noErr;
```

```
        display := true;
        theInfo := NavFileOrFolderInfoPtr(info)
        if (theItem.descriptorType = typeFSS) then
                begin
            if not (theInfo^.isFolder) then
                    begin
                if (theInfo^.fileAndFolder.fileInfo.finderInfo.fdType <> 'TEXT') then
                        begin
                        display := false;
                        end;
                    end;
                end;

    MyFilterFunction := display;
    end;
```

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:

```
function MyPreviewFunction(callBackParms: NavCBRecPtr;
                        callBackUD: NavCallBackUserData): boolean;
```

callBackParms   A pointer to a NavCBRec record that contains event data needed for your function to draw the preview.

callBackUD   A value set by your application. When the NavGetFile function calls back your event-handling function, the callBackUD 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 NavCustomControl 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]

---

[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.

- 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.

- 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	= $000000E4	{ Use defaults for all the options }
kNavNoTypePopup	= $00000001	{ Don't show file type/extension popup on Open/Save }
kNavDontAutoTranslate	= $00000002	{ Don't automatically translate on Open }
kNavDontAddTranslateItems	= $00000004	{ Don't add translation choices on Open/Save }
kNavAllFilesInPopup	= $00000010	{ "All Files" menu item in the type popup on Open }
kNavAllowStationery	= $00000020	{ Allow saving of stationery files }
kNavAllowPreviews	= $00000040	{ Allow to show previews }
kNavAllowMultipleFiles	= $00000080	{ Allow multiple items to be selected }
kNavAllowInvisibleFiles	= $00000100	{ Allow invisible items to be shown }
kNavDontResolveAliases	= $00000200	{ Don't resolve aliases }
kNavSelectDefaultLocation	= $00000400	{ Make the default location the browser selection }
kNavSelectAllReadableItem	= $00000800	{ 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

kNavAskDiscardChanges	= 1
kNavAskDiscardChangesCancel	= 2

Event Message

kNavCBEvent	= 0
kNavCBCustomize	= 1
kNavCBStart	= 2
kNavCBTerminate	= 3

Object Filtering

kNavFilteringBrowserList	= 0
kNavFilteringFavorites	= 1
kNavFilteringRecents	= 2

kNavFilteringShortCutVolumes = 3

Data Types

NavDialog Options Structure

```
NavDialogOptions = RECORD
    version:                UInt16;
    dialogOptionFlags:      NavDialogOptionFlags;
    location:               Point;
    clientName:             Str255;
    windowTitle:            Str255;
    actionButtonLabel:      Str255;
    cancelButtonLabel:      Str255;
    savedFileName:          Str255;
    message:                Str255;
    preferenceKey:          UInt32;
    popupExtension:         Handle;
    reserved:               PACKED ARRAY [0..493] OF CHAR;
END;
```

Nav Reply Structure

```
NavReplyRecord = RECORD
    version:                UInt16;
    validRecord:            BOOLEAN;
    replacing:              BOOLEAN;
    isStationery:           BOOLEAN;
    translationNeeded:      BOOLEAN;
    selection:              AEDescList;
    keyScript:              ScriptCode;
    fileTranslation:        ^FileTranslationSpecPtr;
    reserved1:              UInt32;
    reserved:               PACKED ARRAY [0..230] OF CHAR;
END;
```

NavTypeList

```
NavTypeList = RECORD
    componentSignature:     OSType;
    reserved:               INTEGER;
    osTypeCount:            INTEGER;
    osType:                 ARRAY [0..0] OF OSType;
END;
```

Functions

Calling Navigation Services

```
FUNCTION NavLoad: OSErr;
FUNCTION NavUnload: OSErr;
FUNCTION NavGetDefaultDialogOptions(VAR dialogOptions: NavDialogOptions): OSErr;
FUNCTION NavDisposeReply(VAR reply: NavReplyRecord): OSErr;

FUNCTION NavCustomControl(context: NavContext; selector: NavCustomControlMessage; parms: UNIV Ptr): OSErr;
FUNCTION NavServicesCanRun: BOOLEAN;
```

Choosing Files, Folders, and Volumes

```
FUNCTION NavGetFile(VAR defaultLocation: AEDesc; VAR reply: NavReplyRecord;
        VAR dialogOptions: NavDialogOptions; eventProc: NavEventUPP;
        previewProc: NavPreviewUPP; filterProc: NavObjectFilterUPP;
        typeList: NavTypeListHandle; callBackUD: NavCallBackUserData): OSErr;
FUNCTION NavChooseFile(VAR defaultLocation: AEDesc; VAR reply: NavReplyRecord;
        VAR dialogOptions: NavDialogOptions; eventProc: NavEventUPP;
        previewProc: NavPreviewUPP; filterProc: NavObjectFilterUPP;
        typeList: NavTypeListHandle; callBackUD: NavCallBackUserData): OSErr;
FUNCTION NavChooseFolder(VAR defaultLocation: AEDesc; VAR reply: NavReplyRecord;
        VAR dialogOptions: NavDialogOptions; eventProc: NavEventUPP;
        filterProc: NavObjectFilterUPP; callBackUD: NavCallBackUserData): OSErr;
FUNCTION NavChooseVolume(VAR defaultSelection: AEDesc; VAR reply: NavReplyRecord;
        VAR dialogOptions: NavDialogOptions; eventProc: NavEventUPP;
```

```
          filterProc: NavObjectFilterUPP; callBackUD: NavCallBackUserData): OSErr;
FUNCTION NavChooseObject(VAR defaultLocation: AEDesc; VAR reply: NavReplyRecord;
          VAR dialogOptions: NavDialogOptions; eventProc: NavEventUPP;
          filterProc: NavObjectFilterUPP; callBackUD: NavCallBackUserData): OSErr;
FUNCTION NavNewFolder(VAR defaultLocation: AEDesc; VAR reply: NavReplyRecord;
          VAR dialogOptions: NavDialogOptions; eventProc: NavEventUPP;
          callBackUD: NavCallBackUserData): OSErr;
```

Saving Files

```
FUNCTION NavPutFile(VAR defaultLocation: AEDesc; VAR reply: NavReplyRecord;
          VAR dialogOptions: NavDialogOptions; eventProc: NavEventUPP;
          fileType: OSType; fileCreator: OSType; callBackUD: NavCallBackUserData): OSErr;
FUNCTION NavAskSaveChanges(VAR dialogOptions: NavDialogOptions;
          action: NavAskSaveChangesAction; VAR reply: NavAskSaveChangesResult;
          eventProc: NavEventUPP; callBackUD: NavCallBackUserData): OSErr;
FUNCTION NavCustomAskSaveChanges(VAR dialogOptions: NavDialogOptions;
          VAR reply: NavAskSaveChangesResult; eventProc: NavEventUPP;
          callBackUD: NavCallBackUserData): OSErr;
FUNCTION NavAskDiscardChanges(VAR dialogOptions: NavDialogOptions;
          VAR reply: NavAskDiscardChangesResult; eventProc: NavEventUPP;
          callBackUD: NavCallBackUserData): OSErr;
FUNCTION NavCompleteSave(VAR reply: NavReplyRecord;
          howToTranslate: NavTranslationOptions): OSErr;
```

Demonstration Program

```
{ ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊
  Files2Program.p
 ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊
//
// This program demonstrates:
//
// •   The use of Navigation Services rather than the Standard File Package to display
//     Open, Save, and Choose a Folder dialog boxes.
//
// •   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.
//
//     These functions are essentially the same as those in the demonstration program
//     Files1 except that the safe-save methodology used in Files1 is not used, all
//     saves being direct to the target file.
//
// 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".
//
// The program utilises the following resources:
//
// •   An 'MBAR' resource, and 'MENU' resources for Apple, File, Edit and Demonstration
//     menus (preload, non-purgeable).
//
// •   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.
//
// •   '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
//     Services to   build the native file types section of the Show pop-up menu in the
//     Open dialog box.
```

```
//
// •  An 'open' resource (purgeable) containing the file type list for the Open dialog
//     box.
//
// •  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.
//
// •  A 'SIZE' resource with the acceptSuspendResumeEvents, isHighLevelEventAware, and
//     is32BitCompatible flags set (non-purgeable).
//
// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ }

program Files2Program;

//
.........................................................................................................................................
.......................................... includes

uses

    { Other project files. }
    Files2;

//
.........................................................................................................................................
.................... global variables

var

    mainMenubarHdl : Handle;
    mainMenuHdl : MenuHandle;
    mainErr : OSErr;

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ main

begin

    // ...................................................................................................................................
initialise global variables

    gCurrentNumberOfWindows := 0;
    gCurrentType := 1;

    //
.........................................................................................................................................
.........initialise managers

    DoInitManagers;

    // .................................................................... check for Navigation Services, and pre-load (optional)

{$ifc GENERATINGPOWERPC}
    if ((SInt32(@NavLibraryVersion) <> kUnresolvedCFragSymbolAddress) & NavServicesCanRun) then
{$elsec}
    if NavServicesCanRun then
{$endc}
        begin
        mainErr := NavLoad;
        end
    else begin
        DoErrorAlert(eNoNavServices);
        end;

    // ............................................................................... create routine decriptors for Apple event handlers

    doOpenAppEventUPP := NewAEEventHandlerProc(@DoOpenAppEvent);
    doReopenAppEventUPP := NewAEEventHandlerProc(@DoReopenAppEvent);
    doOpenDocsEventUPP := NewAEEventHandlerProc(@DoOpenDocsEvent);
    doQuitAppEventUPP := NewAEEventHandlerProc(@DoQuitAppEvent);

    // ............................................................ set application's resource fork as current resource file

    gAppResFileRefNum := CurResFile;

    // ................................................................................................................... set
up menu bar and menus
```

```
   mainMenubarHdl := GetNewMBar(rMenubar);
   if (mainMenubarHdl = nil) then
      begin
      DoErrorAlert(MemError);
      end;
   SetMenuBar(mainMenubarHdl);
   DrawMenuBar;

   mainMenuHdl := GetMenuHandle(mApple);
   if (mainMenuHdl = nil) then
      begin
      DoErrorAlert(MemError);
      end
   else begin
      AppendResMenu(mainMenuHdl, 'DRVR');
      end;
```

// .... install required Apple event handlers

```
   DoInstallAEHandlers;
```

// .... enter event loop

```
   EventLoop;

end.
   { of main program block }
```

{ ◊◊ }

{ ◊◊
  Files2.p
 ◊◊ }

unit Files2;

interface
```

// ................................................................................................................................................................................................................ includes

```
uses

   { Universal Interfaces. }
   Appearance, AERegistry, Devices, Dialogs, Folders, Fonts, LowMem, Navigation, Processes,
   Resources, TextUtils, ToolUtils;
```

// .... constants

```
const

mApple = 128;
iAbout = 1;
mFile = 129;
iNew = 1;
iOpen = 2;
iClose = 4;
iSave = 5;
iSaveAs = 6;
iRevert = 7;
iQuit = 12;
mDemonstration = 131;
iTouchWindow = 1;
iChooseAFolderDialog = 3;
rNewWindow = 128;
rMenubar = 128;
rRevertAlert = 128;
rCloseFileAlert = 129;
rCustomOpenDialog = 130;
iPopupItem = 10;
rSelectDirectoryDialog = 131;
iSelectButton = 10;
```

```
rErrorStrings = 128;
eInstallHandler = 1000;
eMaxWindows = 1001;
eFileIsOpen = opWrErr;
eNoNavServices = 1002;
rMiscStrings = 129;
sApplicationName = 1;
sChooseAFolder = 2;
rOpenResource = 128;
kMaxWindows = 10;
MAXLONG = $7FFFFFFF;
```

```
//
........................................................................................................................................................................
.................... type definitions
```

```
type
```

```
DocRecord = record
    editStrucHdl : TEHandle;
    pictureHdl : PicHandle;
    fileRefNum : SInt16;
    fileFSSpec : FSSpec;
    windowTouched : boolean;
    end;
```

```
DocRecordPointer = ^DocRecord;
DocRecordHandle = ^DocRecordPointer;
```

```
//
........................................................................................................................................................................
.................. global variables
```

```
var
```

```
gDone : boolean;
gInBackground : boolean;
doOpenAppEventUPP : AEEventHandlerUPP;
doReopenAppEventUPP : AEEventHandlerUPP;
doOpenDocsEventUPP : AEEventHandlerUPP;
doQuitAppEventUPP : AEEventHandlerUPP;
gAppResFileRefNum : SInt16;
```

```
gCurrentNumberOfWindows : SInt16;
gDestRect, gViewRect : Rect;
gDirectorySelectionFlag : boolean;
```

```
gCurrentType : SInt16;
gPrevSelectedName : Str255;
gWindowPtr : WindowPtr;
```

```
//
........................................................................................................................................................................
........... routine prototypes
```

```
procedure EventLoop;
procedure DoInitManagers;
procedure DoInstallAEHandlers;
procedure DoEvents({const} var theEvent : EventRecord);
procedure DoMouseDown({const} var theEvent : EventRecord);
procedure DoUpdate({const} var theEvent : EventRecord);
procedure DoMenuChoice(menuChoice : SInt32);
procedure DoFileMenu(menuItem : SInt16);
procedure DoAdjustMenus;
procedure DoErrorAlert(error : SInt16);
procedure DoTouchWindow;
function  DoOpenAppEvent(var appEvent, reply : AppleEvent; handlerRefcon : UInt32) : OSErr;
function  DoReopenAppEvent(var appEvent, reply : AppleEvent; handlerRefcon : UInt32) : OSErr;
function  DoOpenDocsEvent(var appEvent, reply : AppleEvent; handlerRefcon : UInt32) : OSErr;
function  DoQuitAppEvent(var appEvent, reply : AppleEvent; handlerRefcon : UInt32) : OSErr;
function  DoHasGotRequiredParams(var theEvent : AppleEvent) : OSErr;
```

```
implementation
```

```
uses
```

```
    ChooseAFolderDialog, NewOpenCloseSave;
```

```
// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ EventLoop

procedure EventLoop;
   var
   theEvent : EventRecord;

   begin
   gDone := false;

   while not gDone do
      begin
      if WaitNextEvent(everyEvent, theEvent, MAXLONG, nil) then
         begin
         DoEvents(theEvent);
         end;
      end;
   end;
      { of procedure EventLoop }


// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoInitManagers

procedure DoInitManagers;
   var
   osError : OSErr;

   begin
   MaxApplZone;
   MoreMasters;

   InitGraf(@qd.thePort);
   InitFonts;
   InitWindows;
   InitMenus;
   TEInit;
   InitDialogs(nil);

   InitCursor;
   FlushEvents(everyEvent, 0);

   osError := RegisterAppearanceClient;

   end;
      { of procedure DoInitManagers }


// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoInstallAEHandlers

procedure DoInstallAEHandlers;
   var
   osError : OSErr;

   begin
   osError := AEInstallEventHandler(kCoreEventClass, kAEOpenApplication,
                         doOpenAppEventUPP, 0, false);
   if (osError <> noErr) then
      begin
      DoErrorAlert(eInstallHandler);
      end;

   osError := AEInstallEventHandler(kCoreEventClass, kAEReopenApplication,
                         doReopenAppEventUPP, 0, false);
   if (osError <> noErr) then
      begin
      DoErrorAlert(eInstallHandler);
      end;

   osError := AEInstallEventHandler(kCoreEventClass, kAEOpenDocuments,
                         doOpenDocsEventUPP, 0, false);
   if (osError <> noErr) then
      begin
      DoErrorAlert(eInstallHandler);
      end;

   osError := AEInstallEventHandler(kCoreEventClass, kAEQuitApplication,
                         doQuitAppEventUPP, 0, false);
   if (osError <> noErr) then
      begin
      DoErrorAlert(eInstallHandler);
      end;
   end;
      { of procedure DoInstallAEHandlers }
```

// ⟨⟨⟩ DoEvents

```
procedure DoEvents({const} var theEvent : EventRecord);
    var
    ignoredErr : OSErr;
    charCode : SInt8;

    begin
    case (theEvent.what) of

        kHighLevelEvent: begin
            ignoredErr := AEProcessAppleEvent(theEvent);
            end;

        mouseDown: begin
            DoMouseDown(theEvent);
            end;

        keyDown, autoKey: begin
            charCode := BAnd(theEvent.message, charCodeMask);
            if (BAnd(theEvent.modifiers, cmdKey) <> 0) then
                begin
                DoAdjustMenus;
                DoMenuChoice(MenuEvent(theEvent));
                end;
            end;

        updateEvt: begin
            DoUpdate(theEvent);
            end;

        osEvt: begin
            if (BAnd(BSR(theEvent.message, 24), $000000FF) = suspendResumeMessage) then
                begin
                gInBackground := BAnd(theEvent.message, resumeFlag) = 0;
                end;

            HiliteMenu(0);
            end;

        otherwise begin
            end;

        end;
            { of case statement }
    end;
        { of procedure DoEvents }
```

// ⟨⟨⟩ DoMouseDown

```
procedure DoMouseDown({const} var theEvent : EventRecord);
    var
    theWindowPtr : WindowPtr;
    partCode : SInt16;
    ignoredErr : OSErr;

    begin
    partCode := FindWindow(theEvent.where, theWindowPtr);

    case partCode of

        inMenuBar: begin
            DoAdjustMenus;
            DoMenuChoice(MenuSelect(theEvent.where));
            end;

        inContent: begin
            if (theWindowPtr <> FrontWindow) then
                begin
                SelectWindow(theWindowPtr);
                end;
            end;

        inDrag: begin
            DragWindow(theWindowPtr, theEvent.where, qd.screenBits.bounds);
            end;

        inGoAway: begin
            if TrackGoAway(theWindowPtr, theEvent.where) then
```

```
            begin
            ignoredErr := DoCloseCommand(kNavSaveChangesClosingDocument);
            end;
        end;

    otherwise begin
        end;

    end;
        { of case statement }
    end;
        { of procedure DoMouseDown }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoUpdate

procedure DoUpdate({const} var theEvent : EventRecord);
    var
    theWindowPtr : WindowPtr;
    docRecordHdl : DocRecordHandle;
    oldPort : GrafPtr;
    destRect : Rect;

    begin
    theWindowPtr := WindowPtr(theEvent.message);
    docRecordHdl := DocRecordHandle(GetWRefCon(theWindowPtr));

    GetPort(oldPort);
    SetPort(theWindowPtr);

    BeginUpdate(theWindowPtr);

    if (docRecordHdl^^.pictureHdl <> nil) then
        begin
        destRect := docRecordHdl^^.pictureHdl^^.picFrame;
        OffsetRect(destRect, 170, 54);
        HLock(Handle(docRecordHdl^^.pictureHdl));
        DrawPicture(docRecordHdl^^.pictureHdl, destRect);
        HUnlock(Handle(docRecordHdl^^.pictureHdl));
        end
    else if (docRecordHdl^^.editStrucHdl <> nil) then
        begin
        HLock(Handle(docRecordHdl^^.editStrucHdl));
        TEUpdate(gDestRect, docRecordHdl^^.editStrucHdl);
        HUnlock(Handle(docRecordHdl^^.editStrucHdl));
        end;

    if (docRecordHdl^^.windowTouched) then
        begin
        TextSize(48);
        MoveTo(30, 170);
        DrawString('WINDOW TOUCHED');
        TextSize(12);
        end;

    EndUpdate(WindowPtr(theEvent.message));

    SetPort(oldPort);
    end;
        { of procedure DoUpdate }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoMenuChoice

procedure DoMenuChoice(menuChoice : SInt32);
    var
    menuID, menuItem : SInt16;
    itemName : Str255;
    daDriverRefNum : SInt16;
    fileSpec : FSSpec;
    osError : OSErr;
    theRect : Rect;
    theString, numberString : Str255;

    begin
    menuID := HiWord(menuChoice);
    menuItem := LoWord(menuChoice);

    if (menuID = 0) then
        begin
        Exit(DoMenuChoice);
        end;
```

```
      case menuID of

         mApple: begin
            GetMenuItemText(GetMenuHandle(mApple), menuItem, itemName);
            daDriverRefNum := OpenDeskAcc(itemName);
            end;

         mFile: begin
            DoFileMenu(menuItem);
            end;

         mDemonstration: begin
            if (menuItem = iTouchWindow) then
               begin
               DoTouchWindow;
               end
            else if (menuItem = iChooseAFolderDialog) then
               begin
               osError := DoChooseAFolderDialog(fileSpec);
               if ((osError <> noErr) and (osError <> userCanceledErr)) then
                  begin
                  end
               else if (FrontWindow <> nil) then
                  begin
                  SetPort(FrontWindow);
                  TextSize(10);
                  SetRect(theRect, 0, 271, 600, 300);
                  EraseRect(theRect);
                  if (osError <> userCanceledErr) then
                     begin
                     theString := fileSpec.name;
                     theString := theString + '  Volume Reference Number: ';
                     NumToString(SInt32(fileSpec.vRefNum), numberString);
                     theString := theString + numberString;
                     theString := theString + '  Parent Directory ID: ';
                     NumToString(SInt32(fileSpec.parID), numberString);
                     theString := theString + numberString;
                     MoveTo(10, 290);
                     DrawString(theString);
                     end;
                  end;
               end;
            end;

         otherwise begin
            end;

         end;
            { of case statement }

   HiliteMenu(0);
   end;
      { of procedure DoMenuChoice }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoFileMenu

procedure DoFileMenu(menuItem : SInt16);
   var
   osError : OSErr;

   begin
   case menuItem of

      iNew: begin
         osError := DoNewCommand;
         if (osError <> noErr) then
            begin
            DoErrorAlert(osError);
            end;
         end;

      iOpen: begin
         osError := DoOpenCommand;
         if (osError <> noErr) then
            begin
            DoErrorAlert(osError);
            end;
         end;
```

```
        iClose: begin
            osError := DoCloseCommand(kNavSaveChangesClosingDocument);
            if ((osError <> noErr) and (osError <> kNavAskSaveChangesCancel)) then
                begin
                DoErrorAlert(osError);
                end;
            end;

        iSave: begin
            osError := DoSaveCommand;
            if (osError <> noErr) then
                begin
                DoErrorAlert(osError);
                end;
            end;

        iSaveAs: begin
            osError := DoSaveAsCommand;
            if (osError <> noErr) then
                begin
                DoErrorAlert(osError);
                end;
            end;

        iRevert: begin
            osError := DoRevertCommand;
            if (osError <> noErr) then
                begin
                DoErrorAlert(osError);
                end;
            end;

        iQuit: begin
            osError := DoQuitCommand(kNavSaveChangesQuittingApplication);
            if ((osError <> noErr) and (osError <> kNavAskSaveChangesCancel)) then
                begin
                DoErrorAlert(osError);
                end;

            if (osError <> kNavAskSaveChangesCancel) then
                begin
                osError := NavUnload;
                gDone := true;
                end;
            end;

        otherwise begin
            end;

        end;
            { of case statement }
    end;
        { of procedure DoFileMenu }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoAdjustMenus

procedure DoAdjustMenus;
    var
    menuHdl : MenuHandle;
    theWindowPtr : WindowPtr;
    docRecordHdl : DocRecordHandle;

    begin
    theWindowPtr := FrontWindow;
    docRecordHdl := DocRecordHandle(GetWRefCon(theWindowPtr));

    menuHdl := GetMenuHandle(mFile);

    if (gCurrentNumberOfWindows > 0) then
        begin
        menuHdl := GetMenuHandle(mFile);
        EnableItem(menuHdl, iClose);
        if (docRecordHdl^^.windowTouched) then
            begin
            EnableItem(menuHdl, iSave);
            EnableItem(menuHdl, iRevert);
            end
        else begin
            DisableItem(menuHdl, iSave);
            DisableItem(menuHdl, iRevert);
```

```
               end;
         EnableItem(menuHdl, iSaveAs);

         menuHdl := GetMenuHandle(mDemonstration);
         if not docRecordHdl^^.windowTouched then
            begin
            EnableItem(menuHdl, iTouchWindow);
            end
         else begin
            DisableItem(menuHdl, iTouchWindow);
            end;
         end
      else begin
         menuHdl := GetMenuHandle(mFile);
         DisableItem(menuHdl, iClose);
         DisableItem(menuHdl, iSave);
         DisableItem(menuHdl, iSaveAs);
         DisableItem(menuHdl, iRevert);
         menuHdl := GetMenuHandle(mDemonstration);
         DisableItem(menuHdl, iTouchWindow);
         end;

   DrawMenuBar;
   end;
      { of procedure DoAdjustMenus }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoErrorAlert

procedure DoErrorAlert(errorCode : SInt16);
   var
   paramRec : AlertStdAlertParamRec;
   errorString, theString : Str255;
   itemHit : SInt16;
   ignoredErr : OSErr;

   begin
   paramRec.movable := true;
   paramRec.helpButton := false;
   paramRec.filterProc := nil;
   paramRec.defaultText := StringPtr(kAlertDefaultOKText);
   paramRec.cancelText := nil;
   paramRec.otherText := nil;
   paramRec.defaultButton := kAlertStdAlertOKButton;
   paramRec.cancelButton := 0;
   paramRec.position := kWindowDefaultPosition;

   if (errorCode = eInstallHandler) then
      begin
      GetIndString(errorString, rErrorStrings, 1);
      end
   else if (errorCode = eMaxWindows) then
      begin
      GetIndString(errorString, rErrorStrings, 2);
      end
   else if (errorCode = eFileIsOpen) then
      begin
      GetIndString(errorString, rErrorStrings, 3);
      end
   else if (errorCode = eNoNavServices) then
      begin
      GetIndString(errorString, rErrorStrings, 4);
      end
   else begin
      GetIndString(errorString, rErrorStrings, 5);
      NumToString(SInt32(errorCode), theString);
      errorString := errorString + theString;
      end;

   if ((errorCode <> memFullErr) and (errorCode <> eNoNavServices)) then
      begin
      ignoredErr := StandardAlert(kAlertCautionAlert, @errorString, nil, @paramRec, itemHit);
      end
   else begin
      ignoredErr := StandardAlert(kAlertStopAlert, @errorString, nil, @paramRec, itemHit);
      ExitToShell;
      end;
   end;
      { of procedure DoErrorAlert }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoTouchWindow
```

```
procedure DoTouchWindow;
    var
    theWindowPtr : WindowPtr;
    docRecordHdl : DocRecordHandle;

    begin
    theWindowPtr := FrontWindow;
    docRecordHdl := DocRecordHandle(GetWRefCon(theWindowPtr));

    SetPort(theWindowPtr);

    TextSize(48);
    MoveTo(30, 170);
    DrawString('WINDOW TOUCHED');
    TextSize(12);

    docRecordHdl^^.windowTouched := true;
    end;
        { of procedure DoTouchWindow }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoOpenAppEvent

function DoOpenAppEvent(var appEvent, reply : AppleEvent; handlerRefcon : UInt32) : OSErr;
    var
    osError : OSErr;

    begin
    osError := DoHasGotRequiredParams(appEvent);
    if (osError = noErr) then
        begin
        osError := DoNewCommand;
        end;

    DoOpenAppEvent := osError;
    end;
        { of function DoOpenAppEvent }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoReopenAppEvent

function DoReopenAppEvent(var appEvent, reply : AppleEvent; handlerRefcon : UInt32) : OSErr;
    var
    osError : OSErr;

    begin
    osError := DoHasGotRequiredParams(appEvent);
    if (osError = noErr) then
        begin
        if (FrontWindow = nil) then
            begin
            osError := DoNewCommand;
            end;
        end;

    DoReopenAppEvent := osError;
    end;
        { of function DoReopenAppEvent }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoOpenDocsEvent

function DoOpenDocsEvent(var appEvent, reply : AppleEvent; handlerRefcon : UInt32) : OSErr;
    var
    fileSpec : FSSpec;
    docList : AEDescList;
    osError, ignoredErr : OSErr;
    index, numberOfItems : SInt32;
    actualSize : Size;
    keyWord : AEKeyword;
    returnedType : DescType;
    fileInfo : FInfo;

    begin
    osError := AEGetParamDesc(appEvent, keyDirectObject, typeAEList, docList);

    if (osError = noErr) then
        begin
        osError := DoHasGotRequiredParams(appEvent);
        if (osError = noErr) then
            begin
            ignoredErr := AECountItems(docList, numberOfItems);
```

```
                if (osError = noErr) then
                    begin
                    for index := 1 to numberOfItems do
                        begin
                        osError := AEGetNthPtr(docList, index, typeFSS, keyWord, returnedType,
                                        @fileSpec, sizeof(fileSpec), actualSize);
                            if (osError = noErr) then
                                begin
                                osError := FSpGetFInfo(fileSpec, fileInfo);
                                if (osError = noErr) then
                                    begin
                                    osError := DoOpenFile(fileSpec, fileInfo.fdType);
                                    if (osError <> noErr) then
                                        begin
                                        DoErrorAlert(osError);
                                        end;
                                    end;
                                end
                            else begin
                                DoErrorAlert(osError);
                                end;
                            end;
                        end;
                    end
                else begin
                    DoErrorAlert(osError);
                    end;

                ignoredErr := AEDisposeDesc(docList);
                end
            else begin
                DoErrorAlert(osError);
                end;

            DoOpenDocsEvent := osError;
            end;
                { of function DoOpenDocsEvent }
```

// ◊◊◊ DoQuitAppEvent

```
function DoQuitAppEvent(var appEvent, reply : AppleEvent; handlerRefcon : UInt32) : OSErr;
    var
    osError : OSErr;
    ignoredErr : OSErr;

    begin
    osError := DoHasGotRequiredParams(appEvent);
    if (osError = noErr) then
        begin
        while (FrontWindow <> nil) do
            begin
            osError := DoCloseCommand(kNavSaveChangesQuittingApplication);
            if ((osError <> noErr) and (osError <> kNavAskSaveChangesCancel)) then
                begin
                DoErrorAlert(osError);
                end;

            if (osError = kNavAskSaveChangesCancel) then
                begin
                Exit(DoQuitAppEvent);
                end;
            end;
        end;

    ignoredErr := NavUnload;
    gDone := true;

    DoQuitAppEvent := osError;
    end;
        { of function DoQuitAppEvent }
```

// ◊◊ DoHasGotRequiredParams

```
function DoHasGotRequiredParams(var appEvent : AppleEvent) : OSErr;
    var
    returnedType : DescType;
    actualSize : Size;
    osError : OSErr;

    begin
```

```
   osError := AEGetAttributePtr(appEvent, keyMissedKeywordAttr, typeWildCard, returnedType,
                                nil, 0, actualSize);
   if (osError = errAEDescNotFound) then
      begin
      DoHasGotRequiredParams := noErr;
      end
   else if (osError = noErr) then
      begin
      DoHasGotRequiredParams := errAEParamMissed;
      end;
   end;
      { of function DoHasGotRequiredParams }

end.
   { of unit Files1 }
```

```
{ ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ }

{ ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊
  NewOpenCloseSave.p
 ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ }

unit NewOpenCloseSave;


interface

uses

   { Other project files. }
   Files2;

function DoNewCommand : OSErr;
function DoOpenCommand : OSErr;
function DoCloseCommand(action : NavAskSaveChangesAction) : OSErr;
function DoSaveCommand : OSErr;
function DoSaveAsCommand : OSErr;
function DoRevertCommand : OSErr;
function DoQuitCommand(action : NavAskSaveChangesAction) : OSErr;
function DoNewDocWindow(showWindow : boolean; documentType : OSType) : OSErr;
function DoOpenFile(fileSpec : FSSpec; documentType : OSType) : OSErr;
function DoCloseFile(theWindowPtr : WindowPtr; docRecordHdl : DocRecordHandle;
            action : NavAskSaveChangesAction) : OSErr;
function DoWriteFile(theWindowPtr : WindowPtr; newFile : boolean) : OSErr;
function DoReadTextFile(theWindowPtr : WindowPtr) : OSErr;
function DoReadPictFile(theWindowPtr : WindowPtr) : OSErr;
function DoWriteTextData(theWindowPtr : WindowPtr; tempFileRefNum : SInt16) : OSErr;
function DoWritePictData(theWindowPtr : WindowPtr; tempFileRefNum : SInt16) : OSErr;
function DoCopyResources(theWindowPtr : WindowPtr) : OSErr;
function DoCopyAResource(resourceType : ResType;
            resourceID, sourceFileRefNum, destFileRefNum : SInt16) : OSErr;
procedure NavEventFunction(callBackSelector : NavEventCallbackMessage;
            callBackParms : NavCBRecPtr; callBackUD : NavCallBackUserData);


implementation

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoNewCommand

function DoNewCommand : OSErr;
   var
   documentType : OSType;

   begin
   documentType := 'TEXT';
   DoNewCommand := DoNewDocWindow(true, documentType);
   end;
      { of function DoNewCommand }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoOpenCommand

function DoOpenCommand : OSErr;
   var
   dialogOptions : NavDialogOptions;
   fileTypeListHdl : NavTypeListHandle;
   navEventFunctionUPP : NavEventUPP;
   osError : OSErr;
   theNavReply : NavReplyRecord;
   resultDesc : AEDesc;
   defaultLocation : AEDescPtr;
```

```
      index, count : SInt32;
      fileSpec : FSSpec;
      fileInfo : FInfo;
      documentType : OSType;
      ignoredErr : OSErr;
      theKeyword : AEKeyword;

   begin
   fileTypeListHdl := nil;
   osError := noErr;

   ignoredErr := NavGetDefaultDialogOptions(dialogOptions);

   GetIndString(dialogOptions.clientName, rMiscStrings, sApplicationName);
   fileTypeListHdl := NavTypeListHandle(GetResource('open', rOpenResource));

   navEventFunctionUPP := NavEventUPP(NewRoutineDescriptor(@NavEventFunction,
                                      uppNavEventProcInfo, GetCurrentISA));

   defaultLocation^.descriptorType := FourCharCode(0);
   defaultLocation^.dataHandle := nil;
   osError := NavGetFile(defaultLocation, theNavReply, dialogOptions, navEventFunctionUPP, nil,
                     nil,fileTypeListHdl, nil);
   DisposeRoutineDescriptor(navEventFunctionUPP);

   if (theNavReply.validRecord and (osError = noErr)) then
      begin
      if (osError = noErr) then
         begin
         osError := AECountItems(theNavReply.selection, count);

         for index := 1 to count do
            begin
            resultDesc.dataHandle := nil;

            theKeyword := FourCharCode(0);
            osError := AEGetNthDesc(theNavReply.selection, index, typeFSS, theKeyword,
                                       resultDesc);
            if (osError = noErr) then
               begin
               BlockMoveData(resultDesc.dataHandle^, @fileSpec, sizeof(FSSpec));

               osError := FSpGetFInfo(fileSpec, fileInfo);
               if (osError = noErr) then
                  begin
                  documentType := fileInfo.fdType;

                  osError := DoOpenFile(fileSpec,documentType);
                  end;

               ignoredErr := AEDisposeDesc(resultDesc);
               end;
            end;
         end;

      ignoredErr := NavDisposeReply(theNavReply);
      end;

   if (fileTypeListHdl <> nil) then
      begin
      ReleaseResource(Handle(fileTypeListHdl));
      end;

   if (osError = userCanceledErr) then
      begin
      osError := noErr;
      end;

   DoOpenCommand := osError;
   end;
      { of function DoOpenCommand }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoCloseCommand

function DoCloseCommand(action : NavAskSaveChangesAction) : OSErr;
   var
   theWindowPtr : WindowPtr;
   windowKind : SInt16;
   docRecordHdl : DocRecordHandle;
   osError : OSErr;
```

```
      begin
      osError := noErr;
      theWindowPtr := FrontWindow;
      windowKind := WindowPeek(theWindowPtr)^.windowKind;

      case windowKind of

         kApplicationWindowKind: begin
            docRecordHdl := DocRecordHandle(GetWRefCon(theWindowPtr));
            osError := DoCloseFile(theWindowPtr, docRecordHdl, action);

            if (osError = kNavAskSaveChangesCancel) then
               begin
               DoCloseCommand := osError;
               Exit(DoCloseCommand);
               end
            else if (osError = noErr) then
               begin
               DisposeWindow(theWindowPtr);
               gCurrentNumberOfWindows := gCurrentNumberOfWindows - 1;
               end;
            end;

         kDialogWindowKind: begin
            { Hide or close modeless dialog, as required. }
            end;

         otherwise begin
            end;

         end;
            { of case statement }

      DoCloseCommand := osError;
      end;
         { of function DoCloseCommand }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoSaveCommand

function DoSaveCommand : OSErr;
   var
   theWindowPtr : WindowPtr;
   docRecordHdl : DocRecordHandle;
   osError : OSErr;

   begin
   osError := noErr;
   theWindowPtr := FrontWindow;
   docRecordHdl := DocRecordHandle(GetWRefCon(theWindowPtr));

   if (docRecordHdl^^.fileRefNum <> 0) then
      begin
      osError := DoWriteFile(theWindowPtr, false);

      SetPort(theWindowPtr);
      EraseRect(theWindowPtr^.portRect);
      InvalRect(theWindowPtr^.portRect);
      end
   else begin
      osError := DoSaveAsCommand;
      end;

   DoSaveCommand := osError;
   end;
      { of function DoSaveCommand }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoSaveAsCommand

function DoSaveAsCommand : OSErr;
   var
   dialogOptions : NavDialogOptions;
   theWindowPtr : WindowPtr;
   docRecordHdl : DocRecordHandle;
   navEventFunctionUPP : NavEventUPP;
   fileType : OSType;
   osError : OSErr;
   theNavReply : NavReplyRecord;
   resultDesc : AEDesc;
   defaultLocation : AEDescPtr;
```

```
theKeyword : AEKeyword;
fileSpec : FSSpec;
fileRefNum : SInt16;
ignoredErr : OSErr;
userData : NavCallBackUserData;


begin
osError := noErr;

ignoredErr := NavGetDefaultDialogOptions(dialogOptions);

theWindowPtr := FrontWindow;
docRecordHdl := DocRecordHandle(GetWRefCon(theWindowPtr));
GetWTitle(theWindowPtr, dialogOptions.savedFileName);
GetIndString(dialogOptions.clientName, rMiscStrings, sApplicationName);

navEventFunctionUPP := NavEventUPP(NewRoutineDescriptor(@NavEventFunction,
                                     uppNavEventProcInfo, GetCurrentISA));

if (docRecordHdl^^.editStrucHdl <> nil) then
   begin
   fileType := 'TEXT';
   end
else if (docRecordHdl^^.pictureHdl <> nil) then
   begin
   fileType := 'PICT';
   end;

defaultLocation^.descriptorType := FourCharCode(0);
defaultLocation^.dataHandle := nil;
userData := nil;
osError := NavPutFile(defaultLocation, theNavReply, dialogOptions, navEventFunctionUPP,
                          fileType, 'kkkG', userData);

DisposeRoutineDescriptor(navEventFunctionUPP);

if (theNavReply.validRecord and (osError = noErr)) then
   begin
   resultDesc.dataHandle := nil;
   theKeyword := FourCharCode(0);

   osError := AEGetNthDesc(theNavReply.selection, 1, typeFSS, theKeyword, resultDesc);
   if (osError = noErr) then
      begin
      BlockMoveData(resultDesc.dataHandle^, @fileSpec, sizeof(FSSpec));

      if not theNavReply.replacing then
         begin
         osError := FSpCreate(fileSpec, 'kkkG', fileType, theNavReply.keyScript);
         if (osError <> noErr) then
            begin
            ignoredErr := AEDisposeDesc(resultDesc);
            ignoredErr := NavDisposeReply(theNavReply);
            DoSaveAsCommand := osError;
            Exit(DoSaveAsCommand);
            end;
         end;

      docRecordHdl^^.fileFSSpec := fileSpec;

      if (docRecordHdl^^.fileRefNum <> 0) then
         begin
         osError := FSClose(docRecordHdl^^.fileRefNum);
         docRecordHdl^^.fileRefNum := 0;
         end;

      if (osError = noErr) then
         begin
         osError := FSpOpenDF(docRecordHdl^^.fileFSSpec, fsRdWrPerm, fileRefNum);
         end;

      if (osError = noErr) then
         begin
         docRecordHdl^^.fileRefNum := fileRefNum;
         SetWTitle(theWindowPtr,fileSpec.name);

         osError := DoWriteFile(theWindowPtr, not theNavReply.replacing);
         end;

      ignoredErr := AEDisposeDesc(resultDesc);
```

```
        ignoredErr := NavCompleteSave(theNavReply, kNavTranslateInPlace);
        end;

      ignoredErr := NavDisposeReply(theNavReply);
      end;

   SetPort(theWindowPtr);
   EraseRect(theWindowPtr^.portRect);
   InvalRect(theWindowPtr^.portRect);

   if (osError = userCanceledErr) then
      begin
      osError := noErr;
      end;

   DoSaveAsCommand := osError;
   end;
      { of function DoSaveAsCommand }
```

// ◊◊◊ DoRevertCommand

```
function DoRevertCommand : OSErr;
   var
   navEventFunctionUPP : NavEventUPP;
   theWindowPtr : WindowPtr;
   docRecordHdl : DocRecordHandle;
   fileName : Str255;
   dialogOptions : NavDialogOptions;
   reply : NavAskSaveChangesResult;
   osError : OSErr;

   begin
   osError := noErr;
   navEventFunctionUPP := NavEventUPP(NewRoutineDescriptor(@NavEventFunction,
                                  uppNavEventProcInfo, GetCurrentISA));

   theWindowPtr := FrontWindow;
   docRecordHdl := DocRecordHandle(GetWRefCon(theWindowPtr));

   SetPort(theWindowPtr);

   GetWTitle(theWindowPtr, fileName);
   BlockMoveData(@fileName[1], @dialogOptions.savedFileName, UInt8(fileName[0]));

   osError := NavAskDiscardChanges(dialogOptions, reply, navEventFunctionUPP, nil);

   if (reply = kNavAskDiscardChanges) then
      begin
      EraseRect(theWindowPtr^.portRect);

      if (docRecordHdl^^.editStrucHdl <> nil) then
         begin
         osError := DoReadTextFile(theWindowPtr);
         end
      else if (docRecordHdl^^.pictureHdl <> nil) then
         begin
         KillPicture(docRecordHdl^^.pictureHdl);
         docRecordHdl^^.pictureHdl := nil;
         osError := DoReadPictFile(theWindowPtr);
         end;

      docRecordHdl^^.windowTouched := false;

      InvalRect(theWindowPtr^.portRect);
      end;

   DoRevertCommand := osError;
   end;
      { of function DoRevertCommand }
```

// ◊◊ DoQuitCommand

```
function DoQuitCommand(action : NavAskSaveChangesAction) : OSErr;
   var
   osError : OSErr;

   begin
   osError := noErr;

   while (FrontWindow <> nil) do
```

```
      begin
      osError := DoCloseCommand(action);
      if (osError <> noErr) then
         begin
         DoQuitCommand := osError;
         Exit(DoQuitCommand);
         end;
      end;

   DoQuitCommand := osError;
   end;
      { of function DoQuitCommand }
```

// ◊◊ DoNewDocWindow

```
function DoNewDocWindow(showWindow : boolean; documentType : OSType) : OSErr;
   var
   docRecordHdl : DocRecordHandle;

   begin
   if (gCurrentNumberOfWindows = kMaxWindows) then
      begin
      DoNewDocWindow := eMaxWindows;
      Exit(DoNewDocWindow);
      end;

   gWindowPtr := GetNewCWindow(rNewWindow, nil, WindowPtr(-1));
   if (gWindowPtr = nil) then
      begin
      DoNewDocWindow := MemError;
      Exit(DoNewDocWindow);
      end;

   SetPort(gWindowPtr);

   docRecordHdl := DocRecordHandle(NewHandle(sizeof(DocRecord)));
   if (docRecordHdl = nil) then
      begin
      DisposeWindow(gWindowPtr);
      DoNewDocWindow := MemError;
      Exit(DoNewDocWindow);
      end;

   SetWRefCon(gWindowPtr, SInt32(docRecordHdl));

   docRecordHdl^^.editStrucHdl := nil;
   docRecordHdl^^.pictureHdl := nil;
   docRecordHdl^^.fileRefNum := 0;
   docRecordHdl^^.windowTouched := false;

   if (documentType = 'TEXT') then
      begin
      gDestRect := gWindowPtr^.portRect;
      InsetRect(gDestRect, 6, 6);
      gViewRect := gDestRect;

      MoveHHi(Handle(docRecordHdl));
      HLock(Handle(docRecordHdl));

      docRecordHdl^^.editStrucHdl := TENew(gDestRect, gViewRect);
      if (docRecordHdl = nil) then
         begin
         DisposeWindow(gWindowPtr);
         DisposeHandle(Handle(docRecordHdl));
         DoNewDocWindow := MemError;
         Exit(DoNewDocWindow);
         end;

      HUnlock(Handle(docRecordHdl));
      end;

   if showWindow then
      begin
      ShowWindow(gWindowPtr);
      end;

   gCurrentNumberOfWindows := gCurrentNumberOfWindows + 1;

   DoNewDocWindow := noErr;
   end;
```

```
        { of function DoNewDocWindow }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoOpenFile

function DoOpenFile(fileSpec : FSSpec; documentType : OSType) : OSErr;
    var
    osError : OSErr;
    fileRefNum : SInt16;
    docRecordHdl : DocRecordHandle;

    begin
    osError := DoNewDocWindow(false, documentType);
    if (osError <> noErr) then
        begin
        DoOpenFile := osError;
        Exit(DoOpenFile);
        end;

    SetWTitle(gWindowPtr, fileSpec.name);

    osError := FSpOpenDF(fileSpec, fsRdWrPerm, fileRefNum);
    if (osError <> noErr) then
        begin
        DisposeWindow(gWindowPtr);
        gCurrentNumberOfWindows := gCurrentNumberOfWindows - 1;
        DoOpenFile := osError;
        Exit(DoOpenFile);
        end;

    docRecordHdl := DocRecordHandle(GetWRefCon(gWindowPtr));
    docRecordHdl^^.fileRefNum := fileRefNum;
    docRecordHdl^^.fileFSSpec := fileSpec;

    if (documentType = 'TEXT') then
        begin
        osError := DoReadTextFile(gWindowPtr);
        if (osError <> noErr) then
            begin
            DoOpenFile := osError;
            Exit(DoOpenFile);
            end;
        end
    else if (documentType = 'PICT') then
        begin
        osError := DoReadPictFile(gWindowPtr);
        if (osError <> noErr) then
            begin
            DoOpenFile := osError;
            Exit(DoOpenFile);
            end;
        end;

    ShowWindow(gWindowPtr);

    DoOpenFile := osError;
    end;
        { of function DoOpenFile }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoCloseFile

function DoCloseFile(theWindowPtr : WindowPtr; docRecordHdl : DocRecordHandle;
            action : NavAskSaveChangesAction) : OSErr;
    var
    reply : NavAskSaveChangesResult;
    navEventFunctionUPP : NavEventUPP;
    dialogOptions : NavDialogOptions;
    fileName : Str255;
    osError : OSErr;

    begin
    reply := 0;

    if (docRecordHdl^^.windowTouched) then
        begin
        GetWTitle(theWindowPtr, fileName);
        BlockMoveData(@fileName[0], @dialogOptions.savedFileName, UInt8(fileName[0]) + 1);

        navEventFunctionUPP := NavEventUPP(NewRoutineDescriptor(@NavEventFunction,
                                    uppNavEventProcInfo, GetCurrentISA));
```

```
      osError := NavAskSaveChanges(dialogOptions, action, reply, navEventFunctionUPP, nil);

      DisposeRoutineDescriptor(navEventFunctionUPP);

    if (reply = kNavAskSaveChangesCancel) then
       begin
       DoCloseFile := OSErr(reply);
       Exit(DoCloseFile);
       end
    else if (reply = kNavAskSaveChangesSave) then
       begin
       osError := DoSaveCommand;
       if (osError <> noErr) then
          begin
          DoCloseFile := osError;
          Exit(DoCloseFile);
          end;
       end;
    end;

  if (docRecordHdl^^.fileRefNum <> 0) then
     begin
     osError := FSClose(docRecordHdl^^.fileRefNum);
     if (osError <> noErr) then
        begin
        osError := FlushVol(nil, docRecordHdl^^.fileFSSpec.vRefNum);
        docRecordHdl^^.fileRefNum := 0;
        end;
     end;

  if (docRecordHdl^^.editStrucHdl <> nil) then
     begin
     TEDispose(docRecordHdl^^.editStrucHdl);
     end;

  if (docRecordHdl^^.pictureHdl <> nil) then
     begin
     KillPicture(docRecordHdl^^.pictureHdl);
     end;

  DisposeHandle(Handle(docRecordHdl));

  DoCloseFile := osError;
  end;
     { of function DoCloseFile }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoWriteFile

function DoWriteFile(theWindowPtr : WindowPtr; newFile : boolean) : OSErr;

  var
  docRecordHdl : DocRecordHandle;
  osError : OSErr;
  fileRefNum : SInt32;

  begin
  docRecordHdl := DocRecordHandle(GetWRefCon(theWindowPtr));
  fileRefNum := docRecordHdl^^.fileRefNum;

  if (docRecordHdl^^.editStrucHdl <> nil) then
     begin
     osError := DoWriteTextData(theWindowPtr,fileRefNum);
     end
  else if (docRecordHdl^^.pictureHdl <> nil) then
     begin
     osError := DoWritePictData(theWindowPtr,fileRefNum);
     end;

  if (osError = noErr) then
     begin
     if newFile then
        begin
        osError := DoCopyResources(theWindowPtr);
        end;
     end;

  DoWriteFile := osError;
  end;
     { of function DoWriteFile }
```

// ◊◊ DoReadTextFile

```
function DoReadTextFile(theWindowPtr : WindowPtr) : OSErr;
  var
  docRecordHdl : DocRecordHandle;
  fileRefNum : SInt16;
  textEditHdl : TEHandle;
  numberOfBytes : SInt32;
  textBuffer : Handle;
  osError : OSErr;

  begin
  docRecordHdl := DocRecordHandle(GetWRefCon(theWindowPtr));
  fileRefNum := docRecordHdl^^.fileRefNum;

  textEditHdl := docRecordHdl^^.editStrucHdl;
  textEditHdl^^.txSize := 10;
  textEditHdl^^.lineHeight := 15;

  osError := SetFPos(fileRefNum, fsFromStart, 0);
  osError := GetEOF(fileRefNum, numberOfBytes);

  if (numberOfBytes > 32767) then
     begin
     numberOfBytes := 32767;
     end;

  textBuffer := NewHandle(Size(numberOfBytes));
  if (textBuffer = nil) then
     begin
     DoReadTextFile := MemError;
     Exit(DoReadTextFile);
     end;

  osError := FSRead(fileRefNum, numberOfBytes, textBuffer^);
  if ((osError = noErr) or (osError = eofErr)) then
     begin
     MoveHHi(textBuffer);
     HLockHi(textBuffer);
     TESetText(textBuffer^, numberOfBytes, docRecordHdl^^.editStrucHdl);
     HUnlock(textBuffer);
     DisposeHandle(textBuffer);
     end
  else begin
     DoReadTextFile := osError;
     Exit(DoReadTextFile);
     end;

  DoReadTextFile := noErr;
  end;
     { of function DoReadTextFile }
```

// ◊◊ DoReadPictFile

```
function DoReadPictFile(theWindowPtr : WindowPtr) : OSErr;
  var
  docRecordHdl : DocRecordHandle;
  fileRefNum : SInt16;
  numberOfBytes : SInt32;
  osError : OSErr;

  begin
  docRecordHdl := DocRecordHandle(GetWRefCon(theWindowPtr));
  fileRefNum := docRecordHdl^^.fileRefNum;

  osError := GetEOF(fileRefNum, numberOfBytes);
  osError := SetFPos(fileRefNum, fsFromStart, 512);
  numberOfBytes := numberOfBytes - 512;

  docRecordHdl^^.pictureHdl := PicHandle(NewHandle(numberOfBytes));
  if (docRecordHdl^^.pictureHdl = nil) then
     begin
     DoReadPictFile := MemError;
     Exit(DoReadPictFile);
     end;

  osError := FSRead(fileRefNum, numberOfBytes, docRecordHdl^^.pictureHdl^);
  if ((osError = noErr) or (osError = eofErr)) then
     begin
     DoReadPictFile := noErr;
```

```
            end
        else begin
            DoReadPictFile := osError;
            end;
        end;
            { of function DoReadPictFile }
```

// ◊◊ DoWriteTextData

```
function DoWriteTextData(theWindowPtr : WindowPtr; tempFileRefNum : SInt16) : OSErr;
    var
    docRecordHdl : DocRecordHandle;
    textEditHdl : TEHandle;
    editText : Handle;
    numberOfBytes : SInt32;
    volRefNum : SInt16;
    osError : OSErr;

    begin
    docRecordHdl := DocRecordHandle(GetWRefCon(theWindowPtr));
    textEditHdl := docRecordHdl^^.editStrucHdl;
    editText := textEditHdl^^.hText;
    numberOfBytes := textEditHdl^^.teLength;

    osError := SetFPos(tempFileRefNum, fsFromStart, 0);
    if (osError = noErr) then
        begin
        osError := FSWrite(tempFileRefNum, numberOfBytes, editText^);
        end;

    if (osError = noErr) then
        begin
        osError := SetEOF(tempFileRefNum, numberOfBytes);
        end;

    if (osError = noErr) then
        begin
        osError := GetVRefNum(tempFileRefNum, volRefNum);
        end;

    if (osError = noErr) then
        begin
        osError := FlushVol(nil, volRefNum);
        end;

    if (osError = noErr) then
        begin
        docRecordHdl^^.windowTouched := false;
        end;

    DoWriteTextData := osError;
    end;
        { of function DoWriteTextData }
```

// ◊◊◊ DoWritePictData

```
function DoWritePictData(theWindowPtr : WindowPtr; tempFileRefNum : SInt16) : OSErr;
    var
    docRecordHdl : DocRecordHandle;
    pictureHdl : PicHandle;
    numberOfBytes, dummyData : SInt32;
    volRefNum : SInt16;
    osError : OSErr;

    begin
    docRecordHdl := DocRecordHandle(GetWRefCon(theWindowPtr));
    pictureHdl := docRecordHdl^^.pictureHdl;

    numberOfBytes := 512;
    dummyData := 0;

    osError := SetFPos(tempFileRefNum, fsFromStart, 0);

    if (osError = noErr) then
        begin
        osError := FSWrite(tempFileRefNum, numberOfBytes, @dummyData);
        end;

    numberOfBytes := GetHandleSize(Handle(docRecordHdl^^.pictureHdl));
```

```
   if (osError = noErr) then
      begin
      HLock(Handle(docRecordHdl^^.pictureHdl));
      osError := FSWrite(tempFileRefNum, numberOfBytes, docRecordHdl^^.pictureHdl^);
      HUnlock(Handle(docRecordHdl^^.pictureHdl));
      end;

   if (osError = noErr) then
      begin
      osError := SetEOF(tempFileRefNum, 512 + numberOfBytes);
      end;

   if (osError = noErr) then
      begin
      osError := GetVRefNum(tempFileRefNum, volRefNum);
      end;

   if (osError = noErr) then
      begin
      osError := FlushVol(nil, volRefNum);
      end;

   if (osError = noErr) then
      begin
      docRecordHdl^^.windowTouched := false;
      end;

   DoWritePictData := osError;
   end;
      { of function DoWritePictData }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoCopyAppNameResource

function DoCopyResources(theWindowPtr : WindowPtr) : OSErr;
   var
   docRecordHdl : DocRecordHandle;
   fileType : OSType;
   osError : OSErr;
   fileRefNum : SInt16;
   ignoredErr : OSErr;

   begin
   docRecordHdl := DocRecordHandle(GetWRefCon(theWindowPtr));

   if (docRecordHdl^^.editStrucHdl <> nil) then
      begin
      fileType := 'TEXT';
      end
   else if (docRecordHdl^^.pictureHdl <> nil) then
      begin
      fileType := 'PICT';
      end;

   FSpCreateResFile(docRecordHdl^^.fileFSSpec, 'kkkG', fileType, smSystemScript);

   osError := ResError;
   if (osError = noErr) then
      begin
      fileRefNum := FSpOpenResFile(docRecordHdl^^.fileFSSpec, fsRdWrPerm);
      end;

   if (fileRefNum > 0) then
      begin
      osError := DoCopyAResource('STR ', -16396, gAppResFileRefNum, fileRefNum);

      if (fileType = 'PICT') then
         begin
         ignoredErr := DoCopyAResource('pnot', 128, gAppResFileRefNum, fileRefNum);
         ignoredErr := DoCopyAResource('PICT', 128, gAppResFileRefNum, fileRefNum);
         end;
      end
   else begin
      osError := ResError;
      end;

   if (osError = noErr) then
      begin
      CloseResFile(fileRefNum);
      end;
```

```
        DoCopyResources := ResError;
        end;
           { of function DoCopyResources }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoCopyAResource

function DoCopyAResource(resourceType : ResType;
                    resourceID, sourceFileRefNum, destFileRefNum : SInt16) : OSErr;
        var
        sourceResourceHdl : Handle;
        sourceResourceName : Str255;
        ignoredType : ResType;
        ignoredID : SInt16;

        begin
        UseResFile(sourceFileRefNum);

        sourceResourceHdl := GetResource(resourceType, resourceID);

        if (sourceResourceHdl <> nil) then
           begin
           GetResInfo(sourceResourceHdl, ignoredID, ignoredType, sourceResourceName);
           DetachResource(sourceResourceHdl);
           UseResFile(destFileRefNum);
           AddResource(sourceResourceHdl, resourceType, resourceID, sourceResourceName);
           if (ResError = noErr) then
              begin
              UpdateResFile(destFileRefNum);
              end;
           end;

        ReleaseResource(sourceResourceHdl);

        DoCopyAResource := ResError;
        end;
           { of function DoCopyAResource }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ NavEventFunction

procedure NavEventFunction(callBackSelector : NavEventCallbackMessage;
           callBackParms : NavCBRecPtr; callBackUD : NavCallBackUserData);

        var
        theWindowPtr : WindowPtr;

        begin
        if (callBackParms^.eventData.eventDataParms.event <> nil) then
           begin
           theWindowPtr := WindowPtr(callBackParms^.eventData.eventDataParms.event^.message);
           end;

        if (callBackSelector = kNavCBEvent) then
           begin
           if (callBackParms^.eventData.eventDataParms.event^.what = updateEvt) then
              begin
              if (WindowPeek(theWindowPtr)^.windowKind <> kDialogWindowKind) then
                 begin
                 DoUpdate(callBackParms^.eventData.eventDataParms.event^);
                 end;
              end;
           end;
        end;
           { of procedure NavEventFunction }

end.
    { of unit NewOpenCloseSave }

{ ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ }

{ ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊
  ChooseAFolderDialog.p
 ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ }

unit ChooseAFolderDialog;


interface
```

```
//
....................................................................................................................................................
............................................ includes

uses

   Files2, Navigation;


function DoChooseAFolderDialog(var theFileSpec : FSSpec) : OSErr;


implementation

uses

   NewOpenCloseSave;

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoChooseAFolderDialog

function DoChooseAFolderDialog(var theFileSpec : FSSpec) : OSErr;
   var
   dialogOptions : NavDialogOptions;
   navEventFunctionUPP : NavEventUPP;
   osError : OSErr;
   theNavReply : NavReplyRecord;
   fileSpec : FSSpec;
   resultDesc : AEDesc;
   defaultLocation : AEDescPtr;
   ignoredErr : OSErr;

   begin
   osError := NavGetDefaultDialogOptions(dialogOptions);
   GetIndString(dialogOptions.message, rMiscStrings, sChooseAFolder);

   navEventFunctionUPP := NavEventUPP(NewRoutineDescriptor(@NavEventFunction,
                                        uppNavEventProcInfo, GetCurrentISA));

   defaultLocation^.descriptorType := FourCharCode(UInt32(0));
   defaultLocation^.dataHandle := nil;
   osError := NavChooseFolder(defaultLocation, theNavReply, dialogOptions, navEventFunctionUPP,
                                  nil, nil);

   DisposeRoutineDescriptor(navEventFunctionUPP);

   if (theNavReply.validRecord and (osError = noErr)) then
      begin
      osError := AECoerceDesc(theNavReply.selection, typeFSS, resultDesc);
      if (osError = noErr) then
         begin
         BlockMoveData(resultDesc.dataHandle^, @fileSpec, sizeof(FSSpec));

         ignoredErr := FSMakeFSSpec(fileSpec.vRefNum, fileSpec.parID ,fileSpec.name,
                                        theFileSpec);
         end;

      ignoredErr := AEDisposeDesc(resultDesc);
      ignoredErr := NavDisposeReply(theNavReply);
      end;

   DoChooseAFolderDialog := osError;
   end;
      { of function DoChooseAFolderDialog }

end.
   { of unit ChooseAFolderDialog }

{ ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ }
```

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

Files2Program.p

main program block

The call to NavServicesCanRun (68K & PowerPC) and check that the Navigation Services library is installed by comparing the address of a routine in the library with the constant kUnresolvedCFragSymbolAddress (PowerPC only) 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 routine 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.

Files2.p

Files2.p is simply the basic "engine" which supports the demonstration. There is little in this file which has not featured in previous demonstration programs.

type definitions

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

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.

constants

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.

Global Variables

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

DoInstallAEHandlers

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.

DoMenuChoice

If the Choose a Folder item in the Demonstration menu is chosen, the application-defined routine 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.p

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 defaults.

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. Setting all fields of the first parameter to 0/nil 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 routine 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.

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 routine 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 routine DoWriteFile is called, otherwise the application-defined routine 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.

NavGetDefaultDialogOptions initialises the specified NavDialogOptions 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. Setting the fields of the first parameter to 0/nil 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 routine 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 routine 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 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 routine, 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 routine 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 record is disposed of. If it is a picture document, the Picture structure is disposed of. Finally, the document record is disposed of.

DoWriteFile

DoWriteFile is called by DoSaveCommand and DoSaveAsCommand. In conjunction with two supporting application-defined routines, it writes the document to disk. Note that, unlike the routine 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 routine is called to write the document's data to its file.

If the file is a newly created file, the application-defined routine 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 lines retrieve the file reference number from the document structure.

The next 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 record to indicate that the document data on disk equates to the document data in memory.

DoWritePictData

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 record 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 routine 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 eventDataParms field, which itself 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

DoChooseAFolderDialog 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 routine 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.