Programming With Navigation Services 1.1

November 20, 1998

Technical Publications

Apple Computer, Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Simultaneously published in the United States and Canada.

Contents

Figures, Tables, and Listings

About Navigation Services

Introduction

This chapter provides an overview of Navigation Services, a new suite of file browsing services for the Mac OS. You should read this chapter if you develop Mac OS applications which open or save files. You will find that Navigation Services is useful in new applications and updates of existing applications.

Navigation Services is an application programming interface that allows your application to provide a user interface for navigating, opening, and saving Mac OS file objects. Navigation Services is provided as a Carbon-compliant replacement for and enhancement to the Standard File Package, which was introduced with the original Macintosh System. Prior to Navigation Services, file browsing in the Mac OS was often confusing to users in light of the differences between Standard File Package dialog boxes and the Finder's file interface. Also, users must navigate much larger volumes than those which existed when the Standard File Package was developed. These large data spaces require extended functionality.

Navigation Services provides tools for you to implement a greatly enhanced user experience in the area of document management. One enhancement is the ability for users to select and open multiple files simultaneously. There are buttons that let users easily select mounted storage volumes, choose recently opened files and folders, or build their own list of favorite items. You can take advantage of translation services offered by the Translation Manager or opt for **deferred translation**, which gives your application the ability to save interim changes in a file's native format and avoid the time-consuming task of translation until the user closes the document.

The enhanced functionality of Navigation Services is easy to adopt. Functions are simple and flexible; they are designed to help you avoid writing custom code. Navigation Services also provides automatic support for the Appearance Manager's extended suite of dialog boxes and user controls to ensure a more consistent and comprehensible user interface across applications.

Requirements

Navigation Services 1.1 is built into Mac OS 8.5. Navigation Services 1.0 requires the Navigation shared library, Appearance Manager 1.0.1 or later, and Mac OS 7.5.5, 7.6.1, or 8.1.

IMPORTANT
The Navigation shared library is instantiated on a per-context basis. Fragments with global shared-data sections should never import from per-context code fragments. For more information about code fragments, see *Inside Macintosh: Mac OS Runtime Architectures.* ▲

Some extended features of Navigation Services require other components, as follows:

- QuickTime for viewing and creating previews of graphic documents. If QuickTime is not installed, the preview option is disabled unless you provide your own preview function.
- Systems prior to Mac OS 8.5 require Macintosh Easy Open for document translation. (Mac OS 8.5 has translation services built-in.)
- AppleGuide or Apple Help for online help.

On 680x0 systems, Navigation Services 1.0 requires the CFM-68K Runtime Enabler. However, Navigation Services functions are available to both classic 68K and CFM-68K applications.

The Standard File Package is supported in Mac OS 8.X. You will obtain greater functionality and compatibility, however, by taking advantage of Navigation Services in lieu of the Standard File Package. Using Navigation Services is required for you to make your applications compatible with Mac OS X, which will not support the Standard File Package.

User Interface

Navigation Services provides an improved user interface for opening and saving documents. This user interface includes a host of easy-to-use features for browsing and managing the file system. Navigation Services dialog boxes include

- Open
- Save
- Choose a File
- Choose a Folder
- Choose a Volume
- Choose a File Object
- Create New Folder

Navigation Services provides two types of alert boxes:

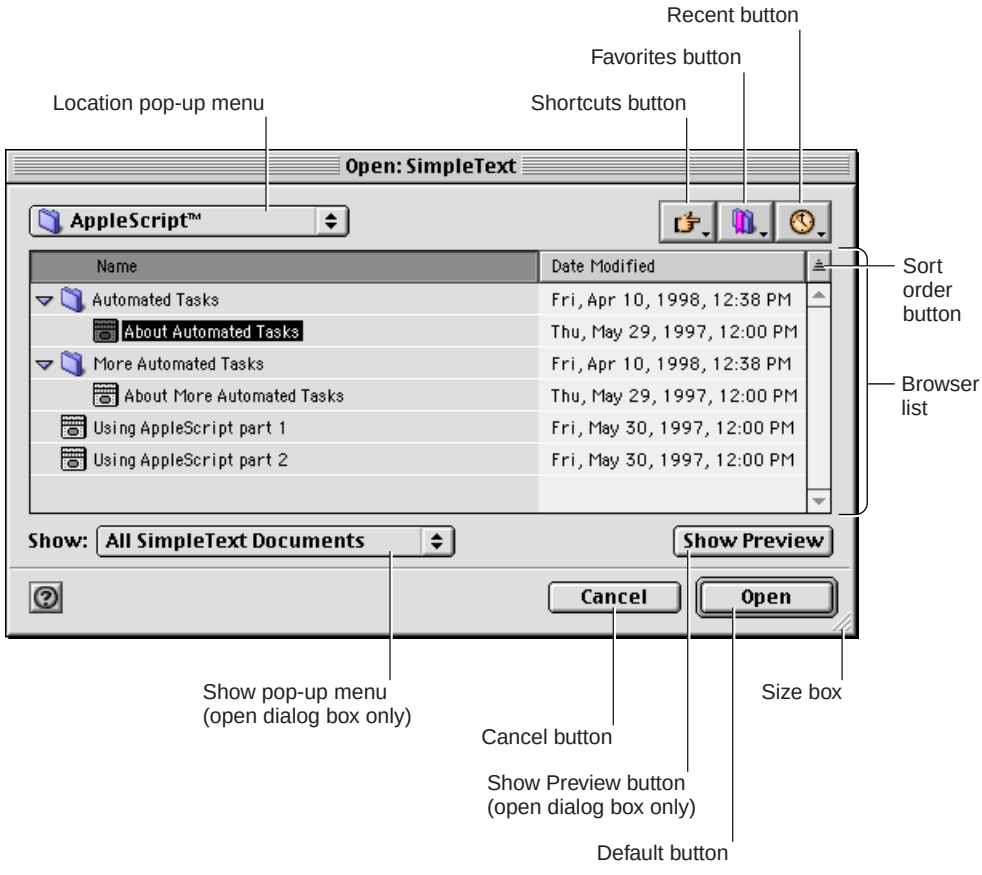
- Save Change
- Discard Changes

All Open, Save, and Choose dialog boxes share some basic user interface elements. These include:

- a browser list
- a Location pop-up menu button
- Shortcuts, Favorites, and Recent bevel buttons
- a default (“action”) button
- a Cancel button
- a sort order button
- an optional size box

Figure 1 shows these elements used in an Open dialog box.

Figure 1 Standard dialog box elements

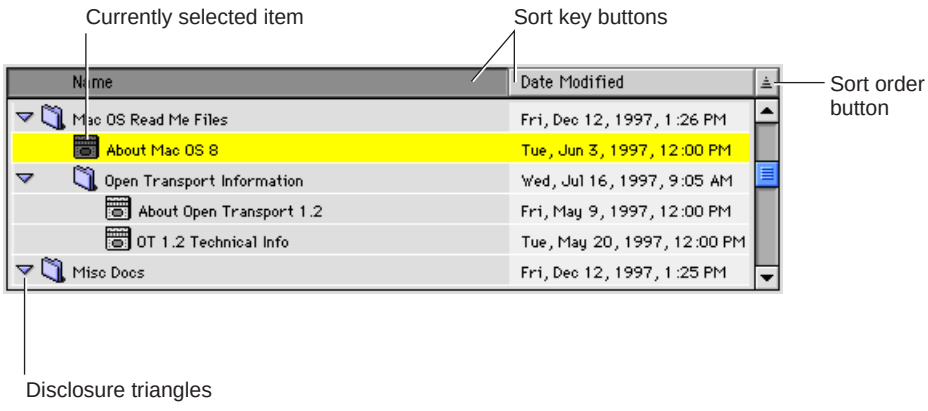


Note
Minor changes were made in the user interface elements of Navigation Services dialog boxes between version 1.0 and version 1.1. Figures in this document have been updated for Navigation Services version 1.1 if appropriate. ◆

Browser List

Navigation Services provides the **browser list** as the primary user access to the file system. The browser list contains a list box, sort buttons, and a scrolling list of files and folders in the directory currently being displayed (the **current location**). The browser list is similar to the Finder's list view in Mac OS 8, as shown in Figure 2.

Figure 2 Browser list



When your application first displays a Navigation Services dialog box, the browser list shows the Desktop location unless you provide a default location. Each type that dialog box is opened afterwards, the browser list defaults (or **rebounds**) to the directory location in use when that particular dialog box was last closed. If a file or folder was selected when the dialog box was last closed, the selection is shown, if possible. If multiple files were selected when an Open dialog box was last closed, the first file in the selection becomes the default selection when the dialog box is next opened.

In the browser list, either the Name or Date field can be used as **sort keys**; the user chooses the sort key by clicking the appropriate bevel button in the list header. The **sort order** (either ascending or descending) can be toggled by clicking the arrow button at the far right of the header. Navigation Services remembers the sort key and sort ordering for each type of dialog box used by each application.

When the user expands the dialog box by using the size box, the browser list expands proportionally. The dates displayed in the browser list provide more information as the browser list expands:

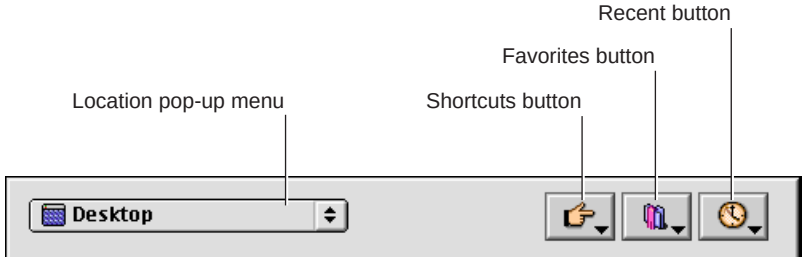
- The smallest size uses the format <MM/DD/YY> (for US systems); for example, 7/16/97.
- If more space is available, the format expands to <DayOfWeek, MMM DD, YY, Time>; for example, Wed, Jul 16, 1997, 9:05 AM.
- The widest format features fully spelled-out names; for example, Wednesday, July 16, 1997, 9:05 AM.

Navigation Services can open multiple files from within the Open dialog box. Users can select multiple files by Shift-clicking within the browser list or using the Select All command. Navigation Services allows multiple selections of files only; folders or volumes are not eligible. If the user tries to extend a selection to include anything but files, Navigation Services treats the attempt as a simple selection; that is, the new object is selected and all previous selections are deselected.

Navigation Options

The Location pop-up menu button and the Shortcut, Favorites, and Recent bevel buttons, shown in Figure 3, provide quick navigation tools.

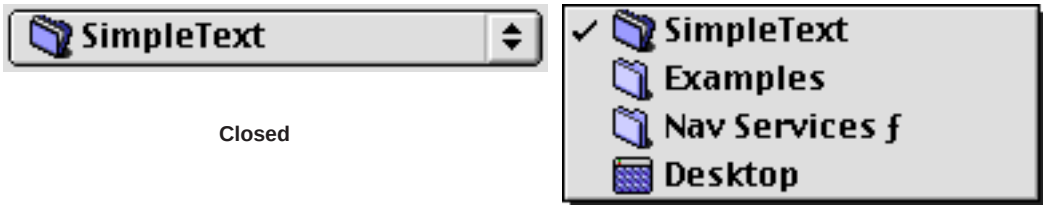
Figure 3 Navigation options



Location Button

The Location pop-up menu displays the current location and provides a familiar way to navigate the file system hierarchy. Figure 4 shows the pop-up menu in its open and closed states.

Figure 4 Location pop-up menu in closed and open states



Closed

Open

Shortcuts Button

The Shortcuts bevel button activates a pop-up menu that allows quick navigation to any mounted volume or directly to the desktop. If ejectable volumes are mounted, an Eject command (one for each volume) appears at the bottom of the menu, as shown in Figure 5.

Figure 5 Shortcuts pop-up menu

Navigation Services 1.1 adds two network connection options to the Shortcuts menu. The Network command displays all available AppleTalk zones in the browser list. The Connect to Server command displays a dialog box that prompts the user to enter a network address for an AppleShare server. The address can be entered as an IP address (“XXX.XXX.XXX.XXX”) or as a domain name (“sample.apple.com”).

Note
Navigation Services 1.1 supports directly opening a new connection to an AppleShare IP server through the Connect to Server command, but does not support direct connections to AppleTalk servers. Your application should not make any assumptions about what type of connections are available, however, since this may change in future versions of Navigation Services. ◆

Favorites Button

The Favorites bevel button activates a pop-up menu of the user’s favorite documents, folders, and volumes, as shown in Figure 6.

Figure 6 Favorites pop-up menu

The Favorites pop-up menu is divided into three sections. The top section contains two commands:

- Add to Favorites allows the user to add the item or items currently selected in the browser list to the Favorites menu. Items may be files, folders, volumes, servers or AppleTalk zones.
- Remove From Favorites opens a dialog box which allows the user to remove an item from the Favorites menu.

The second section contains a list of favorite documents set by the user. The third section contains a list of user-set favorite folders, volumes, and AppleTalk zones. These lists are available to all applications that use Navigation Services.

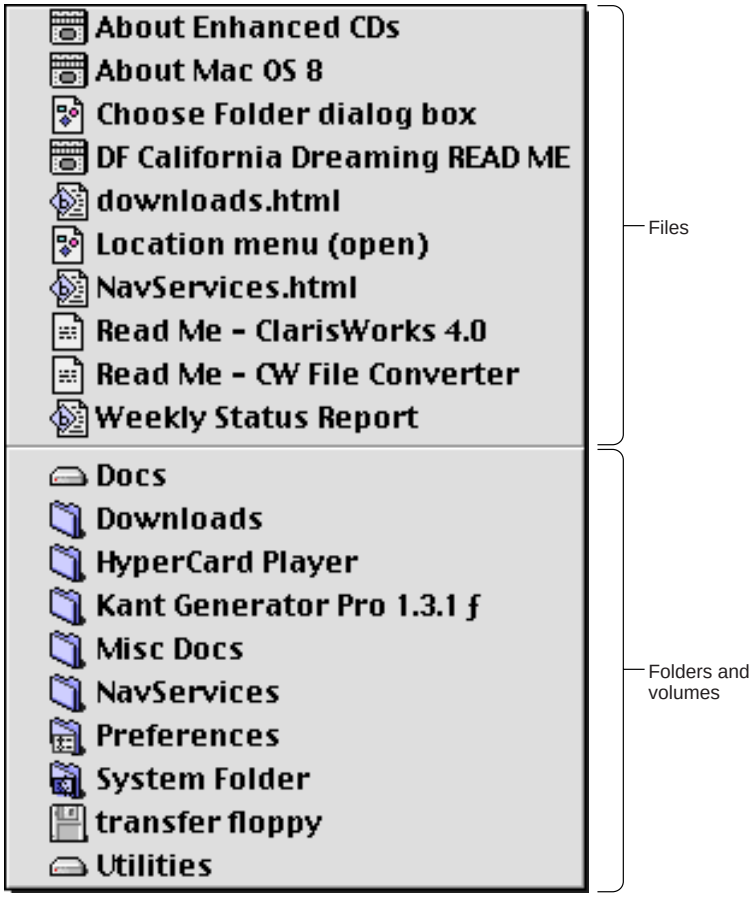
In an Open dialog box, Navigation Services displays favorite files and folders, but in a Save dialog box, only folders are displayed.

In an Open dialog box, the user can add an item to the Favorites menu by using the Add to Favorites command or by dragging the file or folder from the browser list or desktop to the Favorites bevel button. Users can remove items from the list with the Remove From Favorites command.

Recent Button

The Recent bevel button activates a pop-up menu of recently accessed documents, folders, and volumes, as shown in Figure 7.

Figure 7 Recent pop-up menu



The Recent menu is divided into two sections; the first displays documents and the second displays folders and volumes. In an Open dialog box, Navigation Services displays favorite files and folders, but in a Save dialog box, only folders are displayed. The number of items in each section of the Recent menu will not exceed the number set in the Remember Recently Used Items section of the Apple Menu Options dialog box.

Keyboard Equivalents

Navigation Services supports Standard File Package keyboard equivalents, as well as some new ones. Navigation Services keyboard equivalents and the actions they perform are listed below.

- Command-A: Select all (if Edit menu is enabled).
- Command-D or Shift–Command–Up Arrow: Changes current location to desktop.
- Command-N: Creates a new folder.
- Command-O: Opens the selected item.
- Option-Command-O (or press Option while clicking the Open button): Selects the referenced original of an alias item.
- Command-S: Activates Save button if present.
- Up Arrow, Down Arrow: Moves selection up or down in the currently selected scrolling list.
- Shift–Up Arrow and Shift–Down Arrow: Extends the current selection when multiple selection is enabled.
- Command–Up Arrow: Moves to the parent directory of the current location.
- Command–Down Arrow: Opens selected folder or volume.
- Command–Right Arrow: Opens disclosure triangle of selected folder or volume.
- Command–Left Arrow: Closes disclosure triangle of selected folder or volume.
- Option–Left Arrow: Displays previous location in historical sequence; similar to a Web browser's "Back" command.
- Option–Right Arrow: Displays next location in historical sequence; similar to a Web browser's "Forward" command.
- Return key, Enter key: Activates the default button (usually Save or Open).
- Tab: Moves to the next keyboard focus item.
- Escape or Command-period: Cancels and closes the dialog box.
- Home: Moves to the top of the scrolling list.
- End: Moves to the bottom of the scrolling list.

- Page-Up: Scrolls the browser list up one screen.
- Page-Down: Scrolls the browser list down one screen.

Persistence

Persistence is the ability of Navigation Services to store information, such as the last directory location visited and the size and position of dialog boxes. This information is maintained on a per-application basis. For example, this allows the user to set an Open dialog box's position and size differently for a word-processing application than for a spreadsheet, for example.

Navigation Services separates preferences for Open and Save dialog boxes so that each dialog box's preferences are unique for each application. This allows a user to open documents from one folder and save new documents in another folder without any added navigation. Dialog boxes also remember the last document opened and makes this the default selection the next time the dialog box is used.

Note
If no location has been stored for a dialog box or if the directory itself is not available (its volume is unmounted, for example), the desktop becomes the default location. ◆

If the user navigates to the parent directory of the default location through the browser list or by using the location pop-up menu, the default location becomes the current selection.

Note
If the user navigates to the parent directory of the current location by using shortcuts and takes an indirect route to the parent directory, the browser list may display a different default selection. ◆

The size, position, sort key, and sort order of dialog boxes are stored for each application. If a dialog box's position has not been previously set or can't be shown, the dialog box is displayed in the center of the main screen — that is, the one with the menu bar. Alert boxes do not store default locations or alert box size information.

Using Navigation Services

This chapter describes how you can use Navigation Services for tasks like opening and saving files and choosing file objects, and how you can implement custom features in Navigation Services dialog boxes. You should read this chapter to learn how to incorporate Navigation Services into your application.

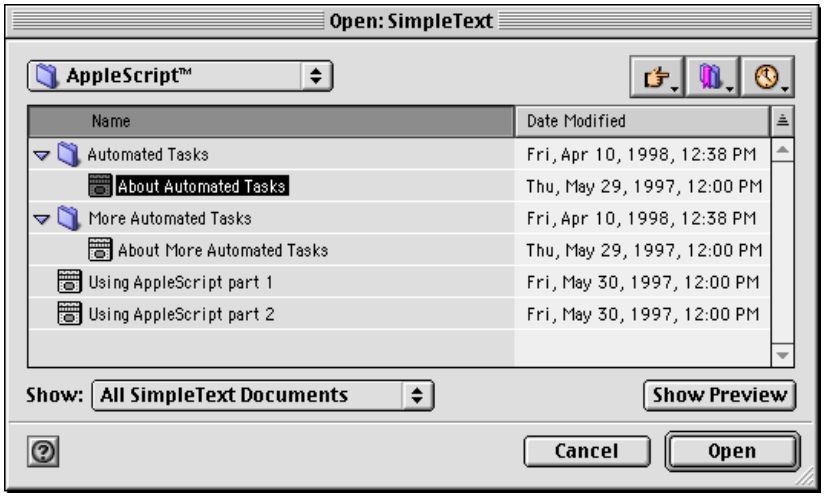
Basic Tasks

Navigation Services is designed to provide a simple, flexible way for applications to open and save files. The primary way to open files is to call the `NavGetFile` function. The primary way to save files is to call the `NavPutFile` function.

Opening Files

The function `NavGetFile` (page 58) displays an Open dialog box that prompts the user to select one or more files to open, as shown in Figure 8.

Figure 8 Open dialog box



For a description of the elements of an Open dialog box, see “User Interface” (page 8).

Note
You are strongly encouraged to make all Navigation Services dialog boxes movable by providing an event-handling function. For more information, see “Handling Events” (page 49). ◆

Providing File Opening Options

The Open dialog box’s Show pop-up menu allows the user to choose the file types displayed by the browser list. The list of available file types is built from information provided by your application when it calls the `NavGetFile` function and by the Translation Manager. If you don’t want the Show pop-up menu button to be displayed, specify the `kNavNoTypePopup` constant in the `dialogOptionFlags` field of a structure of type `NavDialogOptions` (page 85) when you pass this structure in the `dialogOptions` parameter of a file-opening function such as `NavGetFile` (page 58).

Navigation Services uses the Translation Manager to produce a list of file types that your application is capable of translating and displays the list to the user

through an expanded Show pop-up menu. Figure 9 shows an example of a Show pop-up menu with file translation options.

Figure 9 A Show pop-up menu with file translation options

The first section of the Show menu contains an item called “All Readable Documents.” If the user selects this item, the browser list displays all files of the types shown in the second and third sections of the menu.

The second section of the Show menu contains an item called “All <app name> Documents” followed by your application’s “native” file types. **Native file types** are those types you provide in the `typeList` parameter of the file-opening function. This parameter has the same format as an `'open'` resource and may be loaded from a resource or passed in as a structure created on the fly.

Note
Navigation Services will not automatically load an `'open'` resource. Your application must pass a pointer to it in the `typeList` parameter of the file-opening function. ◆

Your application must also contain a `'kind'` resource with entries for each type of file that you want to include in the `typeList` parameter. If you don’t provide an entry for a file type, Navigation Services returns a result of `kNavMissingKindStringErr` (-5699). For more information on `'kind'` resources, see *Inside Macintosh: More Macintosh Toolbox*.

Listing a file type in the `typeList` parameter generally limits you to displaying only those files created by your application. If you specify the

kNavSelectAllReadableItem constant in the dialogOptionFlags field of the NavDialogOptions structure you pass in the dialogOptions parameter, or if the user selects All Readable Items from the Show menu, Navigation Services will ignore the application signature specified in the typeList parameter and display all documents of the specified types. If you choose not to specify file types in the typeList parameter, you can show all files of a particular type in the browser list by using an application-defined filter function. Using a filter function would, for example, allow you to show all text files, regardless of which application created them. For more information on filter functions, see "Filtering File Objects" (page 50).

The third section of the expanded Show pop-up menu contains a list of translatable file types provided by the Translation Manager. Typical names for these entries describe an application document type, such as "MoviePlayer document" in Figure 9. Navigation Services automatically opens and translates file types recognized by the Translation Manager unless you supply the kNavDontAutoTranslate constant in the dialogOptionFlags field of the NavDialogOptions structure you pass in the dialogOptions parameter.

Note
The translatable files section does not appear if you supply the kNavDontAddTranslateItems constant in the dialogOptionFlags field of the NavDialogOptions structure you pass in the dialogOptions parameter. ◆

The last section of the pop-up menu is reserved for the "All Documents" menu item. This item appears if your application supplies the kNavAllFilesInPopup constant in the dialogOptionFlags field of the NavDialogOptions structure you pass in the dialogOptions parameter. This option allows the display of all files, regardless of your application's ability to translate or open them directly. A resource editing application, for example, might take advantage of this option.

Translating Files on Open

If the user selects a file type that is not passed in the typeList parameter, the chosen file must be translated. If the user opens a file called "Doc1" that requires translation, Navigation Services creates a new file called "Doc1 (converted)" with the appropriate file type, in the same directory as the original file. The NavGetFile function automatically performs the translation before returning. However, you can disable this feature by supplying the kNavDontAutoTranslate constant in the dialogOptionFlags field of the structure of type NavDialogOptions (page 85) you pass in the dialogOptions parameter of

the file-opening function. If you disable automatic translation, it is left to your application to call the function `NavTranslateFile` (page 62) as needed.

You can obtain translation information from the structure of type `NavReplyRecord` (page 83) that you passed in the `reply` parameter of the file-opening function. When the file-opening function returns, the `fileTranslation` field of the `NavReplyRecord` structure points to an array of translation records. This array contains one translation record for each file selection identified in the `selection` field of the `NavReplyRecord` structure. If you disable automatic translation, you can use the translation records to provide your own translation.

Note
If a file described in the `selection` field of the `NavReplyRecord` structure does not require translation, its corresponding translation record will be empty. ◆

A Sample File-Opening Function

Listing 1 shows a sample function illustrating one way to call the function `NavGetFile` (page 58). This listing shows how to get default options, modify the options, use an `'open'` resource, and open multiple files.

Note
This listing and other code samples in this document show how to handle an Apple event descriptor of type `'typeFSS'`. Your application should not assume that Navigation Services returns descriptors of any particular type. ◆

Listing 1 A sample file-opening function

```
OSErr MyOpenDocument(const FSSpecPtr defaultLocationfssPtr)
{
    NavDialogOptions    dialogOptions;
    AEDesc              defaultLocation;
    NavEventUPP         eventProc = NewNavEventProc(myEventProc);
    NavObjectFilterUPP  filterProc =
                        NewNavObjectFilterProc(myFilterProc);
    OSErr               anErr = noErr;
```

```
//  Specify default options for dialog box
anErr = NavGetDefaultDialogOptions(&dialogOptions);
if (anErr == noErr)
{
    //  Adjust the options to fit our needs
    //  Set this option
    dialogOptions.dialogOptionFlags |= kNavSelectDefaultLocation;
    //  Clear this one
    dialogOptions.dialogOptionFlags ^= kNavAllowPreviews;

    anErr = AECreateDesc(typeFSS, defaultLocationfssPtr,
                         sizeof(*defaultLocationfssPtr),
                         &defaultLocation );
    if (anErr == noErr)
    {
        // Get 'open' resource. A nil handle being returned is OK,
        // this simply means no automatic file filtering.
        NavTypeListHandle typeList = (NavTypeListHandle)GetResource(
                                    'open', 128);
        NavReplyRecord reply;

        anErr = NavGetFile (&defaultLocation, &reply, &dialogOptions,
                            eventProc, nil, filterProc,
                            typeList, nil);
        if (anErr == noErr && reply.validRecord)
        {
            //  Deal with multiple file selection
            long    count;

            anErr = AECountItems(&(reply.selection), &count);
            if (anErr == noErr)
            {
                longindex;

                for (index = 1; index <= count; index++)
                {
                    AEKeyword   theKeyword;
                    DescType    actualType;
                    Size        actualSize;
                    FSSpec      documentFSSpec;
```

```
                        anErr = AEGetNthPtr(&(reply.selection), index,
                                            typeFSS, &theKeyword,
                                            &actualType,&documentFSSpec,
                                            sizeof(documentFSSpec),
                                            &actualSize);
                        if (anErr == noErr)
                        {
                            anErr = DoOpenFile(&documentFSSpec);
                        }
                    }
                }
                //  Dispose of NavReplyRecord, resources, descriptors
                anErr = NavDisposeReply(&reply);
            }
            if (typeList != NULL)
            {
                ReleaseResource( (Handle)typeList);
            }
            (void) AEDisposeDesc(&defaultLocation);
        }
    }
    DisposeRoutineDescriptor(eventProc);
    DisposeRoutineDescriptor(filterProc);
    return anErr;
}
```

Listing 2 illustrates how your application can open a list of documents by sending itself Apple events.

Listing 2 A sample file-opening function using Apple events

```
static void MyOpenWithEvent(FSSpecPtr defaultLocationfssPtr)
{
    AEDesc              defaultLocation;
    NavReplyRecord      navReply;
    NavDialogOptions    dialogOptions;
    NavTypeListHandle   typeList;
    NavEventUPP         eventProc = NewNavEventProc(myEventProc);
    NavObjectFilterUPP  filterProc =
```

```
                    NewNavObjectFilterProc(myFilterProc);
OSErr               anErr = noErr;

//  Specify default options for dialog box
anErr = NavGetDefaultDialogOptions( &dialogOptions );
if (anErr == noErr)
{
    //  Adjust options to fit our needs
    //  Set this option
    dialogOptions.dialogOptionFlags |= kNavSelectDefaultLocation;
    //  Clear this option
    dialogOptions.dialogOptionFlags ^= kNavAllowPreviews;

    anErr = AECreateDesc(typeFSS, defaultLocationfssPtr,
                         sizeof(*defaultLocationfssPtr),
                         &defaultLocation );
    if (anErr == noErr)
    {
        // Get 'open' resource. A nil handle being returned is OK,
        // this simply means no automatic file filtering.
        typeList = (NavTypeListHandle) GetResource('open', 128);

        anErr = NavGetFile( &defaultLocation, &navReply,
                            &dialogOptions, eventProc,
                            nil, filterProc, typeList, nil);

if (anErr == noErr && navReply.validRecord)
{
    ProcessSerialNumber processSN = {0, kCurrentProcess};
    AEAddressDesc       targetAddress = {typeNull, nil};
    AppleEvent          theEvent = {typeNull, nil};
    AppleEvent          theReply = {typeNull, nil}
    //  This is an event targeted at the current application
    anErr = AECreateDesc(typeProcessSerialNumber, &processSN,
                         sizeof(ProcessSerialNumber),
                         &targetAddress);
    if ( anErr == noErr )
    {
        //  Create an open documents Apple event
        anErr = AECreateAppleEvent(kCoreEventClass, kAEOpenDocuments,
                                   &targetAddress,
```

```
                                        kAutoGenerateReturnID,
                                        kAnyTransactionID, &theEvent);
            (void) AEDisposeDesc(&targetAddress);
        }
        if (anErr == noErr)
        {
            //  Put file list into open document Apple event
            anErr = AEPutParamDesc( &theEvent, keyDirectObject,
                                    &(navReply.selection) );
        }
        if (anErr == noErr)
        {
            //  Send open doc event to our app and dispose of descriptors
            anErr = AESend( &theEvent, &theReply, kAENoReply,
                            kAENormalPriority,
                            kAEDefaultTimeout, nil, nil);
            (void) AEDisposeDesc(&theEvent);
            (void) AEDisposeDesc(&theReply);
        }
        (void) NavDisposeReply(&navReply);
    }
    return;
}
```

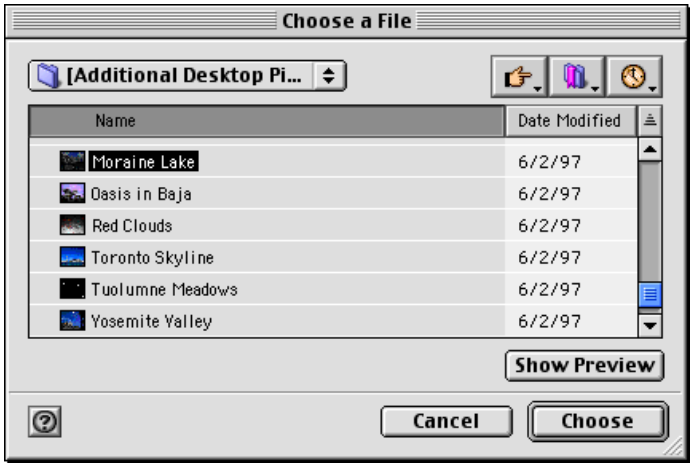
Choosing File Objects

Navigation Services provides functions that prompt the user to select **file objects** (files, folders, or volumes) or create a new folder.

Choosing a File

The function `NavChooseFile` (page 60) displays a dialog box that prompts the user to choose a file for some action other than opening. The file could be a preference file, dictionary, or other specialized file. Figure 10 shows an example of this dialog box.

Figure 10 Choose a File dialog box



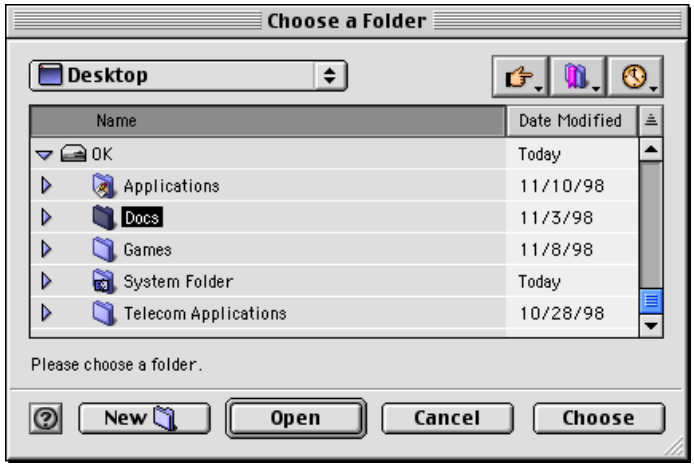
Since this is a simplified function, no built-in translation is available and no Show menu is available. By default, Navigation Services performs no filtering on the files it displays in the browser list. You need to provide file types in the `typeList` parameter or implement an application-defined filter function if you want to affect the file types displayed in the browser list. The `NavChooseFile` function ignores the `signature` field of the `NavTypeList` structure, so all files of the types specified in the `typeList` parameter will be shown, regardless of their application signature. For more information on filtering options, see “Filtering File Objects” (page 41).

You can provide a message or “banner” in the Choose File dialog box by supplying a string in the `message` field of the structure of type `NavDialogOptions` (page 85) that you pass in the `dialogOptions` parameter of the `NavChooseFile` function. This string is displayed below the browser list. If you do not supply a string, no message is displayed.

Choosing a Folder

The function `NavChooseFolder` (page 65) displays a dialog box that prompts the user to choose a folder, as shown in Figure 11. This could be useful when performing installations, for example.

Figure 11 Choose a Folder dialog box



The browser list in a Choose a Folder dialog box displays folders and volumes only.

One possible source of confusion to users who wish to use keyboard equivalents in a list-based file system browser is the purpose of the Enter or Return keys: do they select a folder or do they open it? Navigation Services resolves this issue by mapping the Return and Enter keys to the Open button, which is used to navigate into folders. After navigating to the desired location, the user clicks the Choose button to select the folder.

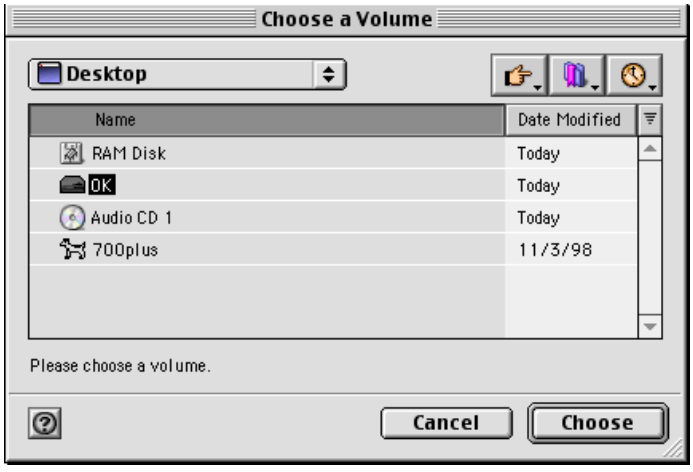
Note
If the user navigates to the desktop, deselects the current selection by shift-clicking and presses the Choose button, `NavChooseFolder` identifies the desktop as the folder chosen. ◆

You can provide a message or "banner" in the Choose a Folder dialog box by supplying a string in the `message` field of the structure of type `NavDialogOptions` (page 85) that you pass in the `dialogOptions` parameter of the `NavChooseFolder` function. This string is displayed below the browser list. In Figure 11, for example, the application supplies the string, "Please select a folder." If you do not supply a string, no message is displayed.

Choosing a Volume

The function `NavChooseVolume` (page 64) displays a dialog box that prompts the user to choose a volume, as shown in Figure 12.

Figure 12 Choose a Volume dialog box



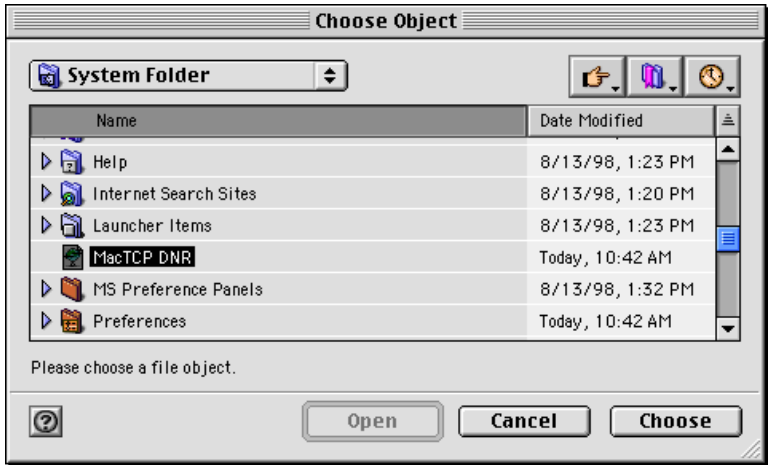
This function is useful when you want the user to select a volume or storage device.

You can provide a message or "banner" in the Choose a Volume dialog box by supplying a string in the `message` field of the structure of type `NavDialogOptions` (page 85) that you pass in the `dialogOptions` parameter of the `NavChooseVolume` function. This string is displayed below the browser list. For example, a disk repair utility might supply a message like "Please choose a volume to repair." If you do not supply a string, no message is displayed.

Choosing a File Object

The function `NavChooseObject` (page 67) displays a dialog box that prompts the user to select a file object, as shown in Figure 13.

Figure 13 Choose Object dialog box



This function is useful when you need the user to select an object that might be one of several different types. If you need to have the user select an object of a specific type, you should use the function appropriate to that type. For example, if you know the object is a file, use the function `NavChooseFile` (page 60).

Note
The `NavChooseObject` function does not allow the user to choose zones or servers. ◆

You can provide a message or "banner" in the Choose Object dialog box by supplying a string in the `message` field of the structure of type `NavDialogOptions` (page 85) that you pass in the `dialogOptions` parameter of the `NavChooseObject` function. This string is displayed below the browser list. In Figure 13, for example, the application supplies the string, "Please choose a file object." If you do not supply a string, no message is displayed.

Creating a New Folder

The function `NavNewFolder` (page 68) displays a dialog box that prompts the user to create a new folder, as shown in Figure 14.

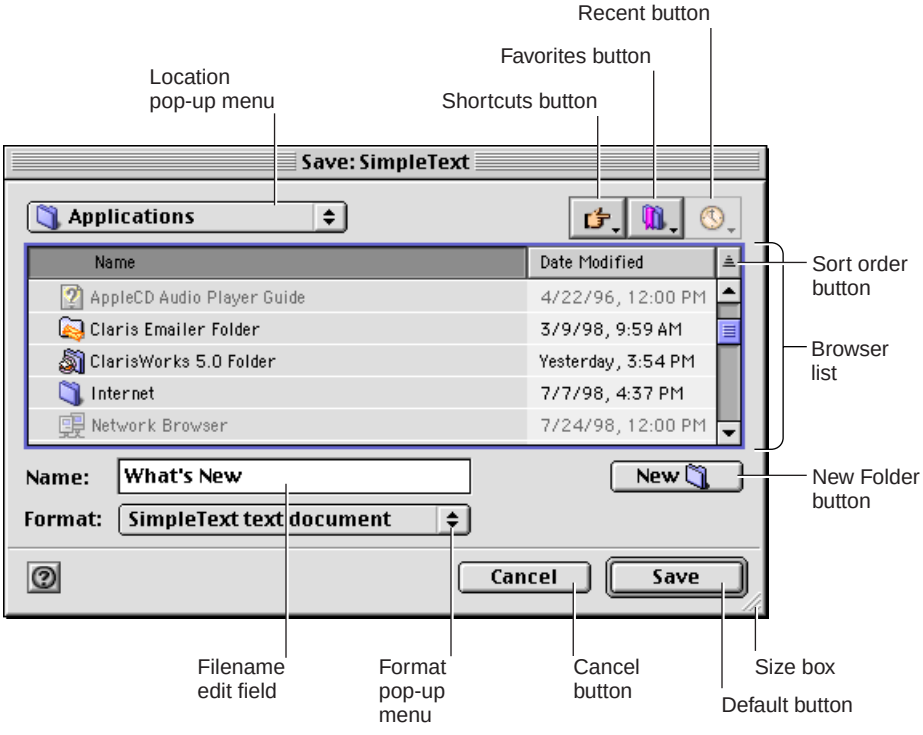
Figure 14 New Folder dialog box

You can provide a message or “banner” in the New Folder dialog box by supplying a string in the `message` field of the structure of type `NavDialogOptions` (page 85) that you pass in the `dialogOptions` parameter of the `NavNewFolder` function. This string is displayed below the browser list. For example, the Sampler installer might provide a message like “Create a folder to install Sampler.” If you do not supply a string, no message is displayed.

Saving Files

The function `NavPutFile` (page 70) displays a Save dialog box, as shown in Figure 15.

Figure 15 Save dialog box



Note
You are strongly encouraged to make all Navigation Services dialog boxes movable by providing updates via an event-handling function. For more information, see "Handling Events" (page 49). ◆

Users can create a new folder for saving a document by using the New Folder button.

When the user selects a folder, the default button title toggles from Save to Open. When the user selects the editable text field (by clicking or keyboard selection), the default button title reverts to Save.

Save dialog boxes display a focus ring to indicate whether the browser list or the editable text field has keyboard focus (that is, the area that receives all

keystrokes.) When no filename is displayed in the editable text field, the Save button is disabled.

IMPORTANT

Always call the function `NavCompleteSave` (page 76) after calling the `NavPutFile` function, even if your application doesn't need automatic translations. Future versions of Navigation Services may provide additional features to your application when it calls the `NavCompleteSave` function. ▲

Providing File Format Options

The function `NavPutFile` (page 70) provides the Format pop-up menu button to allow users to choose how a new document or a copy of a document is to be saved. Figure 16 shows an example of this menu.

Figure 16 Format pop-up menu

Note

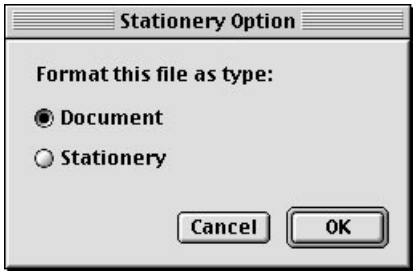
If you specify the `kNavNoTypePopup` constant in the `dialogOptionFlags` field of the structure `NavDialogOptions` (page 85) that you pass in the `dialogOptions` field of the `NavPutFile` function, the Format button does not appear in the Save dialog box. ◆

The first item in the Format pop-up menu is defined by the document type specified by your application in the `fileType` and `fileCreator` parameters of the `NavPutFile` function. The name of the menu item is obtained from the Translation Manager. After setting this item, Navigation Services calls the Translation Manager to determine whether to display subsequent menu items

describing alternative file types. For more information, see “Translating Files on Save” (page 34).

The last item in the menu is the Stationery Option command. This displays the Stationery Option dialog box, shown in Figure 17, which lets the user specify whether a new document or a copy of a document should be saved as a document or as stationery.

Figure 17 Stationery Option dialog box



Note
If you clear the `kNavAllowStationery` constant in the `dialogOptionFlags` field of the structure `NavDialogOptions` (page 85) that you pass in the `dialogOptions` field of the `NavPutFile` function, the Stationery Option menu item does not appear. ◆

Translating Files on Save

Your application supplies its default file type and creator for saved files to the function `NavPutFile` (page 70). 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 structure of type `NavReplyRecord` (page 83). 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`.

IMPORTANT
The function `NavTranslateFile` (page 62) is intended to be used only while opening files. Always call the function `NavCompleteSave` (page 76) to provide automatic translation and complete a save operation. ▲

When saving a document for the first time, your application should wait until the user closes the document before calling the `NavCompleteSave` function. This allows your application to save the file in a native format as the user works with the file. When saving a copy of a document, your application should call the `NavCompleteSave` function immediately after returning from the `NavPutFile` function.

The `NavCompleteSave` function provides any necessary translation. If you wish to turn off automatic translation during a save operation, set the value of the `translationNeeded` field of the `NavReplyRecord` structure to `false` before you call the `NavCompleteSave` function.

Note
You do not need to set the value of the `translationNeeded` field of the `NavReplyRecord` structure to `false` if the user creates a new document that requires translation, but closes it without saving any data. ◆

Once the save is completed, your application must dispose of the `NavReplyRecord` structure by calling the function `NavDisposeReply` (page 57).

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, described in "Translation Option Constants" (page 110), in the `howToTranslate` parameter of the `NavCompleteSave` function.

A Sample File-Saving Function

Listing 3 illustrates how to save files by using the function `NavPutFile` (page 70). The sample listing also shows how to set options and register your event-handling function. Note that this function uses a `DoSafeSave` function to ensure that the save is completed without error before an existing file is deleted.

Listing 3 A sample file-saving function

```
OSErr MySaveDocument(WindowPtr theDocument)
{
    OSErr               anErr = noErr;
    NavReplyRecord      reply;
    NavDialogOptions    dialogOptions;
    OSType              fileTypeToSave = 'TEXT';
    OSType              creatorType = 'xAPP';
    NavEventUPP         eventProc = NewNavEventProc(myEventProc);

    anErr = NavGetDefaultDialogOptions(&dialogOptions);
    if (anErr == noErr)
    {
        //  One way to get the name for the file to be saved.
        GetWTitle(theDocument, dialogOptions.savedFileName);

        anErr = NavPutFile( nil, &reply, &dialogOptions, eventProc, nil,
                            fileTypeToSave, creatorType );
        if (anErr == noErr && reply.validRecord)
        {
            AEKeyword   theKeyword;
            DescType    actualType;
            Size        actualSize;
            FSSpec      documentFSSpec;

            anErr = AEGetNthPtr(&(reply.selection), 1, typeFSS,
                                &theKeyword, &actualType,
                                &documentFSSpec, sizeof(documentFSSpec),
                                &actualSize );
            if (anErr == noErr)
            {
                if (reply.replacing)
                {
                    // Make sure you save a temporary file
                    // so you can check for problems before replacing
                    // an existing file. Once the save is confirmed,
                    // swap the files and delete the original.
                    anErr = DoSafeSave(&documentFSSpec, creatorType,
                                       fileTypeToSave, theDocument);
                }
```

```
                else
                {
                    anErr = WriteNewFile(&documentFSSpec, creatorType,
                                         fileTypeToSave, theDocument);
                }

                if ( anErr == noErr)
                {
                    // Always call NavCompleteSave() to complete
                    anErr = NavCompleteSave(&reply,
                                            kNavTranslateInPlace);
                }
            }
            (void) NavDisposeReply(&reply);
        }
        DisposeRoutineDescriptor(eventProc);
    }
    return anErr;
}
```

Saving Changes

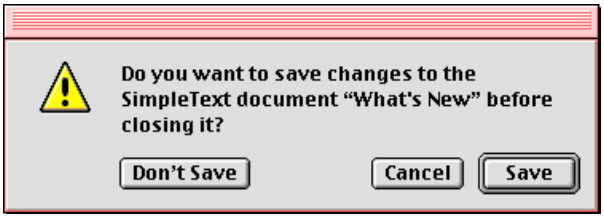
Navigation Services allows you to display a standard alert box for saving changes or quitting an application, and to customize this alert box for other uses.

Note
You are strongly encouraged to make all Navigation Services dialog boxes movable by providing updates via an event-handling function. For more information, see “Handling Events” (page 49). ◆

Displaying a Standard Save Changes Alert Box

To display a standard Save Changes alert box, your application passes its name and the document title to the function `NavAskSaveChanges` (page 72), which displays the alert box shown in Figure 18.

Figure 18 Standard Save Changes alert box

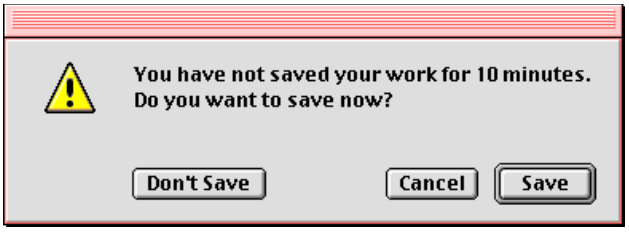


After the user closes the Save Changes alert box, Navigation Services tells your application which button the user clicked by returning one of the `NavAskSaveChangesResult` constants, as described in “Save Changes Action Constants” (page 108).

Customizing the Save Changes Alert Box

You can display a customized Save Changes alert box by using the function `NavCustomAskSaveChanges` (page 73). Figure 19 shows an example of a customized Save Changes alert box.

Figure 19 Custom Save Changes alert box



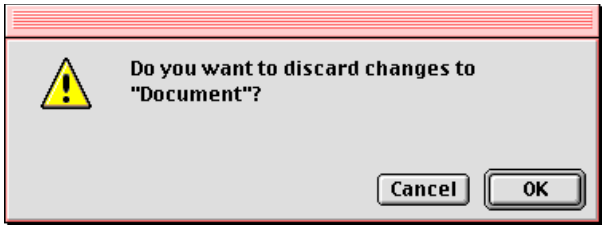
You must provide the message to be displayed in a custom Save Changes alert box.

Displaying a Discard Changes Alert Box

If your application has a Revert to Saved or similar item in its File menu, Navigation Services provides an alert box to handle this situation, as shown in

Figure 20. This alert box is created by calling the function `NavAskDiscardChanges` (page 75).

Figure 20 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 `NavAskDiscardChangesResult` constants, as described in “Discard Changes Action Constants” (page 102).

Advanced Tasks

Setting Custom Features

You can add or change a number of features in Navigation Services dialog boxes.

- You can provide a user prompt or banner, which describes the action performed by the dialog box.
- You can change the action button title.
- You can change the Cancel button title.
- You can preset a dialog box’s location on screen.
- You can change the window title of the dialog box.
- You can provide an application-defined event-handling function (to provide movable and resizable dialog boxes).

- You can provide an application-defined filter function (to determine which file objects are displayed in pop-up menus or the browser list).
- You can provide an application-defined file preview function.

You can set dialog box options by setting values in a structure of type `NavDialogOptions` (page 85).

Setting the Default Location

Navigation Services maintains default location information for dialog boxes. The **default location** is the folder or volume whose contents will be displayed in the browser list when a dialog box is first displayed. You can override the default location and selection of any Navigation Services dialog box by passing a pointer to an Apple event descriptor (`AEDesc`) structure for the new location in the `defaultLocation` parameter of the appropriate function. This `AEDesc` structure will normally be of type `'typeFSS'` and describe a file, folder, or volume.

To select the default location instead of displaying it, supply the `kNavSelectDefaultLocation` constant in the `dialogOptionFlags` field of the structure of type `NavDialogOptions` (page 85) you specify in the `dialogOptions` parameter of a Navigation Services function such as `NavGetFile` (page 58). For example, if you pass an `AEDesc` structure describing the System Folder in the `defaultLocation` parameter of the `NavGetFile` function, Navigation Services displays an Open dialog box with the System Folder as the default location. If you pass the same value to the `NavGetFile` function and supply the `kNavSelectDefaultLocation` constant in the `dialogOptionFlags` field of the `NavDialogOptions` structure, the Open dialog box shows the startup volume as the default location with the System Folder selected.

If you pass `NULL` for the `AEDesc` structure, or attempt to pass an invalid `AEDesc` structure, Navigation Services 1.1 displays the desktop as the default location.

IMPORTANT

If you pass `NULL` for the `AEDesc` structure describing a default location, or attempt to pass an invalid `AEDesc` structure, and specify the `kNavSelectDefaultLocation` constant as described above, Navigation Services 1.0 will crash. ▲

Obtaining Object Descriptions

Several Navigation Services functions return `AEDesc` structures describing objects from the network or the file system. You must not assume that an `AEDesc` structure is of any particular type. If your application needs the structure to be a particular type, you should attempt to coerce the structure using the Apple Event Manager function `AECoerceDesc`. For more information on coercing Apple event descriptors, see *Inside Macintosh: Interapplication Communication*.

When Navigation Services passes you an `AEDesc` structure of type `'typeCString'`, the structure describes a network object by using a Uniform Resource Locator (URL). Network objects can be AppleTalk zones or AppleShare servers. For example, an AppleTalk zone called “Building 1 - 3rd floor” would be represented by a URL of `'at://Building 1 - 3rd floor'`. An AppleShare server called “Mac Software” in the same zone would be represented by a URL of `'afp:/at/Mac Software:Building 1 - 3rd floor'`.

If Navigation Services passes you an `AEDesc` structure of descriptor type `'typeFSS'` describing a directory, the directory’s file specification contains an empty `name` field and its `parID` field contains the directory ID. If an `AEDesc` structure of type `'typeFSS'` describes a file, its file specification’s `name` field contains the filename and its `parID` field contains the directory ID of the file’s parent directory. This means you can use the `name` field to determine whether an object is a file or a folder.

If you need to determine the ID of a directory’s parent directory, use the File Manager function `PBGetCatInfo`, described in *Inside Macintosh: Files*.

Filtering File Objects

The process of choosing which files, folders, and volumes to display in the browser list or the pop-up menus is known as **object filtering**. If your application needs simple, straightforward object filtering, pass a pointer to a structure of type `NavTypeList` (page 91) to the appropriate 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 can’t be displayed; for example, your function can filter out non-HFS objects. You can use both a `NavTypeList` structure and a filter function if you wish, but keep in mind that your filter function is directly affected by the `NavTypeList` structure. For example, if the `NavTypeList` structure contains only `'TEXT'` and `'PICT'` types, only files of those types are passed into your filter function. Also, your filter function can filter out file types that are

defined in your `NavTypeList` structure. Make sure you don't accidentally exclude items you wish to display.

Navigation Services tells you which dialog box control was used for each call to your filter function, so you can implement different criteria for each control. You might choose to limit the Desktop button to displaying specific volumes, for example, or to restrict navigation through the Location pop-up menu. The default location and selections can also be filtered. For more information, see "Filtering File Objects" (page 50).

The function `NavGetFile` (page 58) displays a Show pop-up menu that lists your application's native types as well as translatable file types. If the user chooses a translatable file type, Navigation Services ignores your `NavTypeList` structure and responds only to your filter function. For more information on the Show pop-up menu, see "Providing File Opening Options" (page 19).

The function `NavPutFile` (page 70) displays a Format pop-up menu that displays the save format options, the application's native types, and the file types that can translated. This pop-up menu selection does not affect filtering of the browser list; it determines the file format used to save the final document.

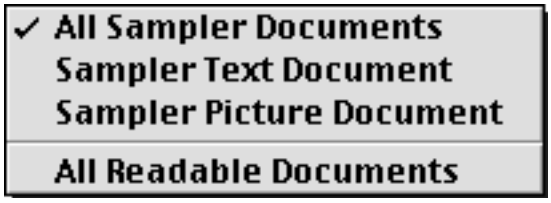
Object Filtering Scenarios

This section gives some examples to help explain how filtering works in the Show pop-up menu. For purposes of illustration, assume the following:

- The application Sampler can open files of type '`TEXT`' and '`PICT`'.
- Sampler passes to the `NavGetFile` function a structure of type `NavTypeList` (page 91) that contains these two file types as well as Sampler's application signature.
- Sampler implements a kind string for each of these native file types.
- Sampler specifies the `kNavDontAddTranslateItems` constant in the `dialogOptions` field in the structure of type `NavDialogOptions` (page 85) that it passes in the `dialogOptions` parameter of the `NavGetFile` function.

The Show pop-up menu contains the items shown in Figure 21. Note that the menu does not contain a translatable file section.

Figure 21 A Show pop-up menu without a translatable file section



The user can select the All Readable Documents command to display all of Sampler's native file types at once.

If Sampler specifies the `kNavNoTypePopup` constant in the `dialogOptions` field, no Show pop-up menu appears and Sampler's `NavTypeList` structure and filter function determine any filtering. If Sampler passes `NULL` to the `NavGetFile` function in place of a reference to the `NavTypeList` structure, the Show pop-up menu won't appear (regardless of the dialog options) and Sampler's application-defined filter function is the only determining filter. If Sampler doesn't provide a filter function, all files are displayed.

Note
Under Navigation Services 1.1, if your application passes a `NavTypeList` structure to the `NavGetFile` function and specifies the `kNavNoTypePopup` constant, Navigation Services will display all files of the types described in the `NavTypeList` structure, even if they were created by a different application. ◆

In the next example, assume the following:

- The application Portal can open files of type '`TEXT`', '`PICT`', and '`MooV`'.
- Portal has a structure of type `NavTypeList` (page 91) containing these three file types as well as its application signature.
- Portal provides kind strings for each of these native file types.
- Portal supplies the `kNavAllFilesInPopup` constant in the `dialogOptions` field of the `NavDialogOptions` structure. This adds the All Documents item at the bottom of the menu.
- Portal does not supply the `kNavDontAddTranslateItems` constant in the `dialogOptions` field of the `NavDialogOptions` structure.

In this case, the Show pop-up menu appears as shown in Figure 22.

Figure 22 A Show pop-up menu with a translatable files section

The third section of the Show menu shows file types that the Translation Manager can translate into one of Portal's three native file types.

Under Navigation Services 1.1, if the user chooses the All Readable Documents menu item, Navigation Services displays all native and translatable file types, regardless of which application created them. If the user chooses the All Documents menu item, the browser list shows all file types, regardless of whether Portal has identified them as translatable or not.

Refreshing the Browser List

If your application needs to refresh the list of file objects in the browser before exiting a Navigation Services function such as `NavGetFile` (page 58), follow these steps:

1. Supply the `kNavCtlGetLocation` constant in the `selector` parameter of the function `NavCustomControl` (page 81) to obtain the current location.
2. Pass the current location in the `parms` parameter of `NavCustomControl` and supply the `kNavCtlSetLocation` constant in the `selector` parameter of `NavCustomControl`.

Getting and setting the current location causes Navigation Services to rebuild the browser list. For more information on these constants, see "Custom Control Setting Constants" (page 96).

Providing Document Previews

Navigation Services provides a preview area in all dialog boxes that open files. This area can be toggled on or off by the user. If the preview area is visible, Navigation Services will automatically display a preview of any file that contains a valid QuickTime component of type `'pnot'`. You can request automatic preview display by setting the `kNavAllowPreviews` constant in the `dialogOptionFlags` field of the structure of type `NavDialogOptions` (page 85) that you pass in the `dialogOptions` parameter of the file-opening function. You can provide your own previews and add custom controls to the preview area, as well. For more information on preview functions, see "Drawing Custom Previews" (page 52). For more information on `'pnot'` resources, see *Inside Macintosh: QuickTime Components.*

Customizing Type Pop-up Menus

The Show pop-up menu displayed in Open dialog boxes and the Format pop-up menu displayed in Save dialog boxed are known collectively as **type pop-up menus**. If your application needs to add its own menu items to one of the type pop-up menus, use a structure of type `NavMenuItemSpec` (page 87) to describe each menu item to add. This allows you to add specific document types to be opened or saved, or different ways of saving a file (with or without line breaks, as HTML, and so forth). To set your menu items, add a handle to one or more `NavMenuItemSpec` structures to the `popupExtension` field in the structure of type `NavDialogOptions` (page 85) that you pass in the `dialogOptions` parameter of the appropriate function. If you provide a `NavMenuItemSpec` structure, you must also provide an event-handling function and an object filtering function. Navigation Services will not handle your custom menu items, so if you do not provide these application-defined functions and attempt to use a `NavMenuItemSpec` structure, Navigation Services functions return a result of `paramErr` (-50). For more information, see "Creating Application-Defined Functions" (page 48).

You do not have to provide a value in the `menuItemName` field of the `NavMenuItemSpec` structure, but Navigation Services uses this value, if it is available, as a search key. If you choose not to provide a value for this field, make sure to set it to an empty string.

To handle and examine a selected pop-up menu item, respond to the `kNavCBPopupMenuSelect` message constant, described in "Custom Control Setting Constants" (page 96), when Navigation Services calls your application's event-handling function. Navigation Services provides information about the

selected menu item in a structure of type NavCBRec (page 89) passed in the callBackParms parameter of your event-handling function. The param field of the NavCBRec structure points to a structure of type NavMenuItemSpec (page 87) describing the menu item. Your application can respond to a particular menu item by comparing the type and creator fields, for example.

It is possible to set the Show or Format pop-up menus so that they display only custom items.

- Define your custom menu items by using structures of type NavMenuItemSpec (page 87).
- Specify a handle to the NavMenuItemSpec structures in the popupExtension field of the structure of type NavDialogOptions (page 85) that you pass in the dialogOptions parameter.
- Pass NULL in the typeList parameter. If you pass any file types in the typeList parameter, Navigation Services will place its own items in the pop-up menu.
- Set your filter function to display only your custom items in the pop-up menu.
- Ensure that your event-handling function takes care of any selection made from the pop-up menu.

If your application tries to extend the type pop-up menu and does not provide an event-handling function, Navigation Services functions return a result of paramErr (-50). If you add menu items that require filtering, you must implement a filter function. For more information, see “Filtering File Objects” (page 50).

Adding Custom Controls

The Navigation Services programming interface handles most common situations that demand interface customization when using the Standard File Package. If you look through all the features and find that you still need to provide custom controls in a Navigation Services dialog box, perform the following steps:

1. Implement an event-handling function to communicate with Navigation Services while Open or Save dialog boxes are open. For more information, see “Handling Events” (page 49).
2. Respond to the kNavCBCustomize constant, described in “Event Message Constants” (page 103), which your application can obtain from the param

field of the structure of type `NavCBRec` (page 89) pointed to in the `callBackParms` parameter of your event-handling function. The `customRect` field of the `NavCBRec` structure defines a rectangle in the local coordinates of the window; the top-left coordinates define the anchor point for the **customization rectangle**, which is the area Navigation Services provides for your application to add custom dialog box items. Your application responds by setting the values which complete the dimensions of the customization rectangle you require in the `customRect` field of the `NavCBRec` structure. After your application responds and exits from the event-handling function, Navigation Services inspects the `customRect` field to determine if the requested dimensions result in a dialog window that can fit on the screen. If the resulting window dimensions are too large, then Navigation Services responds by setting the rectangle to the largest size that can be accommodated and notifying your application with the `kNavCBCustomize` constant again. Your application can continue to negotiate with Navigation Services by examining the `customRect` field and requesting a different size until Navigation Services provides an acceptable rectangle value.
The minimum dimensions for the customization area are 400 pixels wide by 40 pixels high on a 600 x 400 pixel screen. If you are designing for a minimum screen size of 640 x 480 or larger, you can assume a larger minimum customization area.

3. After a customization rectangle has been established, your application must check for the `kNavCBStart` constant in the `param` field of the `NavCBRec` structure. This constant indicates that Navigation Services is opening the dialog box. After you obtain this constant, you can add your interface elements to the customization rectangle. The simplest way to do this is to provide a `'DITL'` resource (in local coordinates, relative to the anchor point of the customization rectangle) and pass the `kNavCtlAddControlList` constant in the `selector` parameter of the function `NavCustomControl` (page 81). Listing 4 illustrates one way to do this.

Listing 4 Adding a custom 'DITL' resource

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

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

You can also draw a single control by calling the Control Manager function `NewControl` and passing the `kNavCtlAddControl` constant, described in "Custom Control Setting Constants" (page 96), in the `selector` parameter of the function `NavCustomControl` (page 81). Listing 5 illustrates this approach.

Listing 5 Adding a single custom control

```
gCustomControl = NewControl (callBackParms->window, &itemRect,
                "\pcheckbox", false, 1, 0, 1, checkBoxProc, NULL);
theErr = NavCustomControl (callBackParms->context, kNavCtlAddControl,
                           gCustomControl);
```

If you call `NewControl`, you must track the custom control yourself.

4. Navigation Services supplies the `kNavCBTerminate` constant in the `param` field of the `NavCBRec` structure after the user closes the dialog box. Make sure you check for this constant, which is your signal to dispose of your control or resource.

Creating Application-Defined Functions

You can implement application-defined functions to intercept and handle events, provide custom view filtering, and draw custom previews.

IMPORTANT

During the execution of any application-defined function, do not assume that your application is at the top of the resource chain or that the current heap zone belongs to your application. Use the Resource Manager function `UseResFile` if you need to set the resource chain. Use the Memory Manager function `SetZone` if you need to set the current heap zone. For more information on the `UseResFile` function, see *Inside Macintosh: More Macintosh Toolbox*. For more information on the `SetZone` function, see *Inside Macintosh: Memory.* ▲

Handling Events

To respond to events generated by the user and Navigation Services, you can create an event-handling function, described in this document as `MyEventProc` (page 77). You register your event-handling function by passing a Universal Procedure Pointer (UPP) in the `eventProc` parameter of a Navigation Services function such as `NavGetFile` (page 58). You obtain this UPP by calling the macro `NewNavEventProc` and passing a pointer to your event-handling function. If an event occurs while the Open dialog box is displayed, for example, the `NavGetFile` function will call your event-handling function. In the `callBackParms` parameter of your event-handling function, `NavGetFile` supplies a structure of type `NavCBRec` (page 89). This structure contains the information your application needs to respond to the event. For instance, your application can obtain the event record describing the event to be handled from the pointer in the `event` field of the `NavCBRec` structure.

Navigation Services will pass update events that apply only to your windows. When calling your event-handling function, Navigation Services passes mouse-down events only when they occur in the preview or customization areas.

You are strongly encouraged to provide at least a simple function to handle update events. If you do this, Navigation Services dialog boxes automatically become movable and resizable. Listing 6 shows an example of such a function.

Listing 6 A sample event-handling function

```
pascal void myEventProc(NavEventCallbackMessage callBackSelector,
                        NavCBRecPtr callBackParms,
                        NavCallBackUserData callBackUD)
{
    WindowPtr window =
                    (WindowPtr)callBackParms->eventData.event->message;
    switch (callBackSelector)
    {
        case kNavCBEvent:
            switch (callBackParms->eventData.event->what)
            {
                case updateEvt:
                    MyHandleUpdateEvent(window,
                        (EventRecord*)callBackParms->eventData.event);
```

```
                    break;
            }
            break;
    }
}
```

In your event-handling function, you can also call the function `NavCustomControl` (page 81) to control various aspects of dialog boxes. For example, the following line shows how you can determine whether the preview area is currently showing:

```
NavCustomControl(context, kNavCtlIsPreviewShowing, &isShowing);
```

If you extend the type pop-up menus with custom menu items, Navigation Services expects your event-handling function to respond to these items. For more information, see “Customizing Type Pop-up Menus” (page 45).

Filtering File Objects

Navigation Services notifies you before displaying items in the following areas:

- Location pop-up menu
- browser list
- Favorites menu
- Recent menu
- Shortcuts menu

You can take advantage of this notification process by creating a filter function, described in this document as `MyFilterProc` (page 79), to determine which items are displayed. Register your filter function by passing a Universal Procedure Pointer (UPP) in the `filterProc` parameter of a Navigation Services function such as `NavGetFile` (page 58). You obtain this UPP by calling the macro `NewNavObjectFilterProc` and passing a pointer to your filter function. When calling your filter function, Navigation Services provides detailed information on HFS files and folders via a structure of type `NavFileOrFolderInfo` (page 87). Your filter function specifies which file objects to display to the user by returning `true` for each object you wish to display and `false` for each object you do not wish to display. If your filter function does not recognize an object, it should return `true` and allow the object to be displayed.

IMPORTANT

Your filter function must not assume that the data passed to it in the `info` parameter is a valid reference to a file specification (that is, a structure of type `FSSpec`. This data is not valid when Navigation Services 1.1 passes a non-HFS file object to your filter function. If you intend to use the data passed in the `info` parameter, you must first check the Apple event descriptor (`AEDesc`) structure passed in the `theItem` parameter. If the `AEDesc` structure cannot be coerced into an `FSSpec`, the object is non-HFS and the data in the `info` parameter is not a reference to a file specification. ▲

Listing 7 illustrates a sample filter function that allows only text files to be displayed.

Listing 7 A sample filter function

```
pascal Boolean myFilterProc(AEDesc* theItem, void* info,
                            NavCallBackUserData callBackUD,
                            NavFilterModes filterMode)
{
    OSErr theErr = noErr;
    Boolean display = true;
    NavFileOrFolderInfo* theInfo = (NavFileOrFolderInfo*)info;

    if (theItem->descriptorType == typeFSS)
        if (!theInfo->isFolder)
            if (theInfo->fileAndFolder.fileInfo.finderInfo.fdType
                != 'TEXT')
                display = false;
    return display;
}
```

IMPORTANT

Navigation Services expects your filter function to return `true` if an object is to be displayed. This is the opposite of what Standard File expects from file filter functions. ▲

For more information on object filtering options, see “Filtering File Objects” (page 41).

Drawing Custom Previews

By default, Navigation Services draws a preview in the Open dialog box when a file selected in the browser list contains a valid `'pnot'` component. To override how previews are drawn and handled, you can create a preview-drawing function, as described in `MyPreviewProc` (page 78). You register your preview-drawing function by passing a Universal Procedure Pointer (UPP) in the `previewProc` parameter of a Navigation Services function such as `NavGetFile` (page 58). You obtain this UPP by calling the macro `NewNavPreviewProc` and passing a pointer to your preview-drawing function. When the user selects a file, Navigation Services calls your preview-drawing function. Before you attempt to create a custom preview, your application should determine whether previews are enabled by specifying the `kNavCtlIsPreviewShowing` constant in the `NavCustomControlMessages` parameter of the function `NavCustomControl` (page 81).

Your preview-drawing function obtains information from a structure of type `NavCBRec` (page 89) specified in the `callBackParms` parameter of your event-handling function. The `NavCBRec` structure contains the following information:

- The `eventData` field contains a structure of type `NavEventData` (page 90) describing the file to be previewed. This structure provides an Apple event descriptor list (`AEDescList`) that you must be able to coerce into a file specification (`FSSpec`). If you cannot coerce the `AEDescList` into a valid `FSSpec`, then the object is not a file and you should not attempt to create a preview.
- The `previewRect` field describes the **preview area**, which is the section of the dialog box reserved for preview drawing.
- The `window` field identifies the window to draw in.

Once you have determined that previews are enabled, your preview-drawing function should draw the custom preview in the specified area and return a function result of `true`. If you don't want to draw a preview for a given file, be sure to return a function result of `false`.

Reference for Navigation Services

This chapter describes the application programming interface to Navigation Services. You should consult this chapter for specific information on how to use Navigation Services functions, constants, and data types in your application.

Functions for Navigation Services

Navigation Services supplies functions to perform the following types of tasks:

- "Identifying Navigation Services Availability" (page 53)
- "Setting up Navigation Services" (page 55)
- "Choosing Files, Folders and Volumes" (page 57)
- "Saving Files" (page 70)
- "Handling Events and Customizing Dialog Boxes" (page 77)

Identifying Navigation Services Availability

You should always use these functions to test for Navigation Services availability and features. Navigation Services is a shared library, so calling the `Gestalt` function does not reveal whether it is running.

- `NavServicesAvailable` (page 54) determines whether the Navigation Services library is available on the user's system.
- `NavLibraryVersion` (page 54) reports the currently installed version of the Navigation Services library.

NavServicesAvailable

Determines whether the Navigation Services library is available on the user's system.

```
Boolean NavServicesAvailable (void);
```

function result A Boolean value. This function returns `true` if Navigation Services is available, `false` if not.

DISCUSSION

Use this function before attempting to use Navigation Services.

SPECIAL CONSIDERATIONS

There is a known problem with Navigation Services 1.0 that occurs if you call `NavServicesAvailable` more than once without the Appearance Manager being installed. Make sure that you check for the presence of the Appearance Manager before calling `NavServicesAvailable`.

NavLibraryVersion

Reports the currently installed version of the Navigation Services shared library.

```
UInt32 NavLibraryVersion (void);
```

function result An unsigned 32-bit integer. This value represents the version number (in binary-coded decimal) of Navigation Services installed on the user's system.

DISCUSSION

If you need to use Navigation Services features that are present only in version 1.1, use this function to determine which version of Navigation Services is installed.

Setting up Navigation Services

These functions allow you to configure Navigation Services features.

- `NavLoad` (page 55) pre-loads the Navigation Services library.
- `NavUnload` (page 55) unloads the Navigation Services library.
- `NavGetDefaultDialogOptions` (page 56) determines the default attributes or behavior for dialog boxes.
- `NavDisposeReply` (page 57) releases the memory used for the `NavReplyRecord` structure after your application has finished using the structure.

NavLoad

Pre-loads the Navigation Services shared library.

```
OSErr NavLoad (void);
```

function result See “Result Codes for Navigation Services” (page 110) for a list of result codes.

DISCUSSION

Use this function to pre-load the Navigation Services library. Pre-loading increases the memory used by your application, but it provides the best performance when using Navigation Services functions. If you don’t use the `NavLoad` function, the Navigation Services shared library is not loaded until your application calls one of the Navigation Services functions and unloads after the call completes. If you use the `NavLoad` function, you must call the function `NavUnload` (page 55) if you want to release reserved memory prior to quitting.

NavUnload

Unloads the Navigation Services shared library.

```
OSErr NavUnload (void);
```

function result See “Result Codes for Navigation Services” (page 110) for a list of result codes.

DISCUSSION

This function allows your application to unload the Navigation Services library and release the memory reserved for it. If you use the function `NavLoad` (page 55) to load the Navigation Services library, you must call the `NavUnload` function if you want to release reserved memory prior to quitting.

NavGetDefaultDialogOptions

Determines the default attributes or behavior for dialog boxes.

```
OSErr NavGetDefaultDialogOptions (NavDialogOptions *dialogOptions);
```

`dialogOptions` A pointer to a structure of type `NavDialogOptions` (page 85). On return, Navigation Services fills out the structure with default option values that your application can change as needed.

function result See “Result Codes for Navigation Services” (page 110) for a list of result codes.

DISCUSSION

This function gives you a simple way to initialize a structure of type `NavDialogOptions` (page 85) and set the default dialog box options before calling one of the dialog box display functions. After you create the `NavDialogOptions` structure, you can supply it with the `NavDialogOptions` constants, described in “Configuration Option Constants” (page 93), to change the configuration options.

NavDisposeReply

Releases the memory allocated for a `NavReplyRecord` structure after your application has finished using the structure.

```
OSErr NavDisposeReply(NavReplyRecord *reply);
```

`reply`	A pointer to a structure of type `NavReplyRecord` (page 83) that your application has created.
function result	See "Result Codes for Navigation Services" (page 110) for a list of result codes.

DISCUSSION

If your application calls a Navigation Services function that uses a structure of type `NavReplyRecord` (page 83), you must use the `NavDisposeReply` function afterward to release the memory allotted for the `NavReplyRecord` structure.

Choosing Files, Folders and Volumes

Navigation Services provides functions to display dialog boxes that prompt the user to select and open various types of file objects.

- `NavGetFile` (page 58) displays an Open dialog box and prompts the user to select a file or files to be opened.
- `NavChooseFile` (page 60) displays a simple dialog box that prompts the user to select a file.
- `NavTranslateFile` (page 62) provides a means for files opened through Navigation Services to be read from different file formats.
- `NavChooseVolume` (page 64) displays a dialog box that prompts the user to choose a volume.
- `NavChooseFolder` (page 65) displays a dialog box that prompts the user to choose a folder or volume.
- `NavChooseObject` (page 67) displays a dialog box that prompts the user to choose a file, folder or volume.

- `NavNewFolder` (page 68) displays a dialog box that prompts the user to create a new folder.

NavGetFile

Displays an Open dialog box and prompts the user to select a file or files to be opened.

```
OSErr NavGetFile    (AEDesc *defaultLocation,
                     NavReplyRecord *reply,
                     NavDialogOptions *dialogOptions,
                     NavEventUPP eventProc,
                     NavPreviewUPP previewProc,
                     NavObjectFilterUPP filterProc,
                     NavTypeListHandle typeList,
                     void *callBackUD);
```

`defaultLocation` A pointer to an Apple event descriptor structure (`AEDesc`). Before calling `NavGetFile`, you can set up a structure of `AEDesc` type `'typeFSS'` to specify a default location to be viewed. If you pass `NULL` in this parameter, Navigation Services defaults to the last location visited during a call to the `NavGetFile` function. If the file system specification in the `AEDesc` structure does not describe a directory or volume, Navigation Services uses the desktop as the default location.

`reply` A pointer to a structure of type `NavReplyRecord` (page 83). Upon return, Navigation Services uses this structure to provide data to your application about the results of your `NavGetFile` call.

`dialogOptions` A pointer to a structure of type `NavDialogOptions` (page 85). Before calling `NavGetFile`, set up this structure to specify dialog box settings. If you pass `NULL` in this parameter, Navigation Services uses the defaults for all options. See "Configuration Option Constants" (page 93) for a description of the default settings.

`eventProc` A Universal Procedure Pointer (UPP) of type `NavEventProcPtr` (page 92) that points to your

	application-defined event-handling function. Obtain this UPP by calling the function `NewNavEventProc`. Implementing an event-handling function allows your application to update windows after the user moves or resizes the dialog box. If you pass `NULL` in this parameter, the Open dialog box is not movable or resizable. For more information, see "Handling Events" (page 49) .
`previewProc`	A Universal Procedure Pointer (UPP) of type `NavPreviewProcPtr` (page 92) that points to your application-defined preview function. Obtain this UPP by calling the function `NewNavPreviewProc`. A preview function allows your application to draw previews or to override Navigation Services previews. For more information, see "Drawing Custom Previews" (page 52).
`filterProc`	A Universal Procedure Pointer (UPP) of type `NavFilterProcPtr` (page 93) that points to your application-defined filter function. Obtain this UPP by calling the function `NewNavObjectFilterProc`. An application-defined filter function determines if a volume, directory, or file should be displayed in the browser list or pop-up menus. For more information, see "Filtering File Objects" (page 50).
`typeList`	A handle to a structure of type `NavTypeList` (page 91). Before calling, set up this structure to declare file types that your application can open. For more information, see "Providing File Format Options" (page 33).
`callBackUD`	A pointer to a value set by your application. When the `NavGetFile` function calls your event-handling function, the `callBackUD` value is passed back to your application.
function result	See "Result Codes for Navigation Services" (page 110) for a list of result codes.

DISCUSSION

After your application calls the `NavGetFile` function to display an Open dialog box and the user selects one or more files and clicks the Open button, `NavGetFile` closes the dialog box and returns references to the files to be opened in the `NavReplyRecord` structure. Your application should check the `validRecord` field of the `NavReplyRecord` structure; if this field is set to `true`, your application

should open the files specified in the `selection` field of the `NavReplyRecord` structure.

SPECIAL CONSIDERATIONS

Always dispose of the `NavReplyRecord` structure after completing the file opening operation by calling the function `NavDisposeReply` (page 57). Failure to use the `NavDisposeReply` function causes memory used for the `NavReplyRecord` structure to remain allocated.

If you use the Show pop-up menu in an Open dialog box, your application must provide adequate kind strings to describe its native file types. For more information, see "Providing File Format Options" (page 33). For more information on kind strings, see *Inside Macintosh: More Macintosh Toolbox*.

SEE ALSO

For more information, see "Opening Files" (page 18).

For a sample code listing, see "A Sample File-Saving Function" (page 35).

NavChooseFile

Creates a simple dialog box that prompts the user to select a file.

```
OSErr NavChooseFile (AEDesc *defaultLocation,
                     NavReplyRecord *reply,
                     NavDialogOptions *dialogOptions,
                     NavEventUPP eventProc,
                     NavPreviewUPP previewProc,
                     NavObjectFilterUPP filterProc,
                     NavTypeListHandle typeList,
                     void *callBackUD);
```

`defaultLocation` A pointer to an Apple event descriptor structure (`AEDesc`). Before calling `NavChooseFile`, you can set up a structure of `AEDesc` type `'typeFSS'` to specify a default location to be viewed. If you pass `NULL` in this parameter, Navigation Services displays the last location visited during a call to

	the `NavChooseFile` function. If the file system specification in the `AEDesc` structure does not describe a directory or volume, Navigation Services uses the desktop as the default location.
`reply`	A pointer to a structure of type `NavReplyRecord` (page 83). Upon return, Navigation Services uses this structure to provide data to your application about the results of your `NavChooseFile` call.
`dialogOptions`	A pointer to a structure of type `NavDialogOptions` (page 85). Before calling `NavChooseFile`, you can set up this structure to specify dialog box settings. If you pass `NULL` in this parameter, Navigation Services uses the defaults for all options. See “Configuration Option Constants” (page 93) for a description of the default settings.
`eventProc`	A Universal Procedure Pointer (UPP) of type `NavEventProcPtr` (page 92) that points to your application-defined event-handling function. Obtain this UPP by calling the function `NewNavEventProc`. Implementing an event-handling function allows your application to update windows after the user moves or resizes the dialog box. If you pass `NULL` in this parameter, the Choose a File dialog box is not movable or resizable. For more information, see “Handling Events” (page 49) .
`previewProc`	A Universal Procedure Pointer (UPP) of type `NavPreviewProcPtr` (page 92) that points to your application-defined preview function. Obtain this UPP by calling the function `NewNavPreviewProc`. A preview function allows your application to draw previews or to override Navigation Services previews. For more information, see “Drawing Custom Previews” (page 52).
`filterProc`	A Universal Procedure Pointer (UPP) of type `NavFilterProcPtr` (page 93) that points to your application-defined filter function. Obtain this UPP by calling the function `NewNavObjectFilterProc`. An application-defined filter function determines if a volume, directory, or file should be displayed in the browser list or pop-up menus. For more information, see “Filtering File Objects” (page 50).

`typeList`	A handle to a structure of type `NavTypeList` (page 91). Before calling `NavChooseFile`, you can set up this structure to declare file types that your application can open. For more information, see "Providing File Format Options" (page 33).
`callBackUD`	A pointer to a value set by your application. When the `NavChooseFile` function calls your event-handling function, the `callBackUD` value is passed back to your application.
function result	See "Result Codes for Navigation Services" (page 110) for a list of result codes.

DISCUSSION

This function allows the user to choose a single file, such as a preferences file, for an action other than opening. The `NavChooseFile` function is similar to the `NavGetFile` function (page 58), but is limited to selecting a single file.

SPECIAL CONSIDERATIONS

The dialog box displayed by the `NavChooseFile` function does not display a Show menu. If you wish to control the files displayed by the browser list or the pop-up menus, you must specify a list of file types in the `typeList` parameter or specify a filter function in the `filterProc` parameter. If you specify a list of file types in the `typeList` parameter, the `NavChooseFile` function ignores the `signature` field of the `NavTypeList` structure. This means that all files of the types specified in the list of file types will be displayed, regardless of their application signature.

NavTranslateFile

Provides a means for files opened through Navigation Services to be read from different file formats.

```
OSErr NavTranslateFile (NavReplyRecord *translateInfo,
                        NavTranslateOptions howToTranslate);
```

`translateInfo`	A pointer to a structure of type `NavReplyRecord` (page 83). Upon return, Navigation Services uses this structure to provide translation information about the selected files.
`howToTranslate`	A value of type `NavTranslationOptions`. Pass one of these constants to tell Navigation Services how to perform the translation: either in-place or to a copy. For a description of the constants, see "Translation Option Constants" (page 110).
function result	See "Result Codes for Navigation Services" (page 110) for a list of result codes.

DISCUSSION

Under automatic file translation, Navigation Services calls the `NavTranslateFile` function, if necessary, before returning from a file-opening function.

Your application can perform its own translation using the `NavReplyRecord` structure you specified in the `translateInfo` parameter. The `NavReplyRecord` structure contains a list of descriptors for the file or files to be opened. The `NavReplyRecord` structure also contains a corresponding list of translation specification records that can be passed to the Translation Manager. To determine if your application has to translate a file, your application can examine the `NavReplyRecord` structure to see if Navigation Services set the `translationNeeded` field to `true`. (The `translationNeeded` field of the `NavReplyRecord` structure is also set to `true` after returning from a `NavGetFile` call during which automatic translation was performed.) If you want to turn off automatic file translation, set the option `kNavDontAutoTranslate` in the `dialogOptionFlags` field of the structure of type `NavDialogOptions` (page 85) that you pass in the `dialogOptions` parameter of the file-opening function.

If your application uses the `NavTranslateFile` function after opening a file without automatic translation, Navigation Services prompts the user to select a location in which to save the translated file if the source location cannot accept a new file (as occurs when the volume is locked or there is insufficient space). The same prompt may occur when automatic translation is enabled in an Open dialog box.

SEE ALSO

"Translating Files on Open" (page 21).

NavChooseVolume

Displays a dialog box that prompts the user to choose a volume.

```
OSErr NavChooseVolume (AEDesc *defaultSelection,
                       NavReplyRecord *reply,
                       NavDialogOptions *dialogOptions,
                       NavEventUPP eventProc,
                       NavFilterUPP filterProc,
                       void *callBackUD);
```

`defaultSelection` A pointer to an Apple event descriptor structure (`AEDesc`). Before calling `NavChooseVolume`, you can set up a structure of `AEDesc` type `'typeFSS'` to specify a default location to be viewed. If you pass `NULL` in this parameter, Navigation Services displays the last location visited during a call to the `NavChooseVolume` function. If the file system specification in the `AEDesc` structure does not describe a directory or volume, Navigation Services uses the desktop as the default location.

`reply` A pointer to a structure of type `NavReplyRecord` (page 83). Upon return, Navigation Services uses this structure to provide data to your application about the results of your `NavChooseVolume` call.

`dialogOptions` A pointer to a structure of type `NavDialogOptions` (page 85). Before calling, set up this structure to specify dialog box settings. If you pass `NULL` in this parameter, Navigation Services uses the defaults for all options. See "Configuration Option Constants" (page 93) for a description of the default settings.

`eventProc` A Universal Procedure Pointer (UPP) of type `NavEventProcPtr` (page 92) that points to your application-defined event-handling function. Obtain this UPP by calling the function `NewNavEventProc`. Implementing an event-handling function allows your application to update windows after the user moves or resizes the dialog box. If you pass `NULL` in this parameter, the Choose a Volume dialog box is not movable or resizable. For more information, see "Handling Events" (page 49) .

`filterProc`	A Universal Procedure Pointer (UPP) of type `NavFilterProcPtr` (page 93) that points to your application-defined filter function. Obtain this UPP by calling the function `NewNavObjectFilterProc`. An application-defined filter function determines if a volume, directory, or file should be displayed in the browser list or pop-up menus. For more information, see "Filtering File Objects" (page 50).
`callBackUD`	A pointer to a value set by your application. When the `NavChooseVolume` function calls your event-handling function, the `callBackUD` value is passed back to your application.
function result	See "Result Codes for Navigation Services" (page 110) for a list of result codes.

DISCUSSION

This function provides a way for your application to prompt the user to select a volume. This might be useful for a disk repair utility, for example.

NavChooseFolder

Displays a dialog box that prompts the user to choose a folder or volume.

```
OSErr NavChooseFolder(AEDesc *defaultLocation,
                      NavReplyRecord *reply,
                      NavDialogOptions *dialogOptions,
                      NavEventUPP eventProc,
                      NavObjectFilterUPP filterProc,
                      void *callBackUD);
```

`defaultLocation`	A pointer to an Apple event descriptor structure (`AEDesc`). Before calling `NavChooseFolder`, you can set up a structure of `AEDesc` type `'typeFSS'` to specify a default location to be viewed. If you pass `NULL` in this parameter, Navigation Services displays the last location visited during a call to the `NavChooseFolder` function. If the file system specification in the `AEDesc` structure does not describe a

	directory or volume, Navigation Services uses the desktop as the default location.
`reply`	A pointer to a structure of type `NavReplyRecord` (page 83). Upon return, Navigation Services uses this structure to provide data to your application about the results of your `NavChooseFolder` call.
`dialogOptions`	A pointer to a structure of type `NavDialogOptions` (page 85). Before calling `NavChooseFolder`, set up this structure to specify dialog box settings. If you pass `NULL` in this parameter, Navigation Services uses the defaults for all options. See "Configuration Option Constants" (page 93) for a description of the default settings.
`eventProc`	A Universal Procedure Pointer (UPP) of type `NavEventProcPtr` (page 92) that points to your application-defined event-handling function. Obtain this UPP by calling the function `NewNavEventProc`. Implementing an event-handling function allows your application to update windows after the user moves or resizes the dialog box. If you pass `NULL` in this parameter, the dialog box is not movable or resizable. For more information, see "Handling Events" (page 49) .
`filterProc`	A Universal Procedure Pointer (UPP) of type `NavFilterProcPtr` (page 93) that points to your application-defined filter function. Obtain this UPP by calling the function `NewNavObjectFilterProc`. An application-defined filter function determines if a volume, directory, or file should be displayed in the browser list or pop-up menus. For more information, see "Filtering File Objects" (page 50).
`callBackUD`	A pointer to a value set by your application. When the `NavChooseFolder` function calls your event-handling function, the `callBackUD` value is passed back to your application.
function result	See "Result Codes for Navigation Services" (page 110) for a list of result codes.

DISCUSSION

This function provides a way for your application to prompt the user to select a folder or volume. This might be useful if you need to install application files, for example.

NavChooseObject

Displays a dialog box that prompts the user to choose a file, folder, or volume.

```
OSErr NavChooseObject(AEDesc *defaultLocation,
                      NavReplyRecord *reply,
                      NavDialogOptions *dialogOptions,
                      NavEventUPP eventProc,
                      NavObjectFilterUPP filterProc,
                      void *callBackUD);
```

`defaultLocation` A pointer to an Apple event descriptor structure (`AEDesc`). Before calling `NavChooseObject`, you can set up a structure of `AEDesc` type `'typeFSS'` to specify a default location to be viewed. If you pass `NULL` in this parameter, Navigation Services displays the last location visited during a call to the `NavChooseObject` function. If the file system specification in the `AEDesc` structure does not describe a directory or volume, Navigation Services uses the desktop as the default location.

`reply` A pointer to a structure of type `NavReplyRecord` (page 83). Upon return, Navigation Services uses this structure to provide data to your application about the results of your `NavChooseObject` call.

`dialogOptions` A pointer to a structure of type `NavDialogOptions` (page 85). Before calling `NavChooseObject`, set up this structure to specify dialog box settings. If you do not provide this structure, Navigation Services uses the defaults for all options. See “Configuration Option Constants” (page 93) for a description of the default settings.

`eventProc` A Universal Procedure Pointer (UPP) of type `NavEventProcPtr` (page 92) that points to your application-defined event-handling function. Obtain this

	UPP by calling the function `NewNavEventProc`. Implementing an event-handling function allows your application to update windows after the user moves or resizes the dialog box. If you pass `NULL` in this parameter, the dialog box is not movable or resizable. For more information, see "Handling Events" (page 49) .
`filterProc`	A Universal Procedure Pointer (UPP) of type `NavFilterProcPtr` (page 93) that points to your application-defined filter function. Obtain this UPP by calling the function `NewNavObjectFilterProc`. An application-defined filter function determines if a volume, directory, or file should be displayed in the browser list or pop-up menus. For more information, see "Filtering File Objects" (page 50).
`callBackUD`	A pointer to a value set by your application. When the `NavChooseObject` function calls your event-handling function, the `callBackUD` value is passed back to your application.
function result	See "Result Codes for Navigation Services" (page 110) for a list of result codes.

DISCUSSION

This function is useful when you need to display a dialog box that prompts the user to choose a file object that might be a file, folder, or volume. If you want the user to choose a specific type of file object, you should use the function designed for that type of object; to select a file, for example, use the function `NavChooseFile` (page 60).

NavNewFolder

Displays a dialog box that prompts the user to create a new folder.

```
OSErr NavNewFolder  (AEDesc *defaultLocation,
                     NavReplyRecord *reply,
                     NavDialogOptions *dialogOptions,
                     NavEventUPP eventProc,
                     void *callBackUD);
```

`defaultLocation`	A pointer to an Apple event descriptor structure (`AEDesc`). Before calling `NavNewFolder`, you can set up a structure of `AEDesc` type `'typeFSS'` to specify a default location to be viewed. If you pass `NULL` in this parameter, Navigation Services displays the last location visited during a call to the `NavNewFolder` function. If the file system specification in the `AEDesc` structure does not describe a directory or volume, Navigation Services uses the desktop as the default location.
`reply`	A pointer to a structure of type `NavReplyRecord` (page 83). Upon return, Navigation Services uses this structure to provide data to your application about the results of the `NavNewFolder` function call.
`dialogOptions`	A pointer to a structure of type `NavDialogOptions` (page 85). Before calling `NavNewFolder`, set up this structure to specify dialog box settings. If you pass `NULL` in this parameter, Navigation Services uses the defaults for all options. See "Configuration Option Constants" (page 93) for a description of the default settings.
`eventProc`	A Universal Procedure Pointer (UPP) of type `NavEventProcPtr` (page 92) that points to your application-defined event-handling function. Obtain this UPP by calling the function `NewNavEventProc`. Implementing an event-handling function allows your application to update windows after the user moves or resizes the dialog box. If you pass `NULL` in this parameter, the dialog box is not movable or resizable. For more information, see "Handling Events" (page 49) .
`callBackUD`	A pointer to a value set by your application. When the `NavNewFolder` function calls your event-handling function, the `callBackUD` value is passed back to your application.
function result	See "Result Codes for Navigation Services" (page 110) for a list of result codes.

DISCUSSION

This function provides a way for your application to prompt the user to create a new folder. This might be useful for creating a project folder, for example.

Saving Files

Navigation Services provides functions to save documents and to query the user about unsaved changes.

- `NavPutFile` (page 70) displays a Save dialog box.
- `NavAskSaveChanges` (page 72) displays a Save Changes alert box.
- `NavCustomAskSaveChanges` (page 73) displays a Save Changes alert box with a custom alert message.
- `NavAskDiscardChanges` (page 75) displays an alert box that asks the user whether to discard changes to a particular document.
- `NavCompleteSave` (page 76) completes a save operation and performs any needed translation on the file.

NavPutFile

Displays a Save dialog box.

```
OSErr NavPutFile    (AEDesc *defaultLocation,
                     NavReplyRecord *reply,
                     NavDialogOptions *dialogOptions,
                     NavEventUPP eventProc,
                     OSType fileType,
                     OSType fileCreator,
                     void *callBackUD);
```

`defaultLocation` A pointer to an Apple event descriptor structure (`AEDesc`). Before calling `NavPutFile`, you can set up a structure of `AEDesc` type `'typeFSS'` to specify a default location to be viewed. If you pass `NULL` in this parameter, Navigation Services displays the last location visited during a call to the `NavPutFile` function. If the file system specification in the `AEDesc` structure does not describe a directory or volume, Navigation Services uses the desktop as the default location.

`reply` A pointer to a structure of type `NavReplyRecord` (page 83). Upon return, Navigation Services uses this structure to

provide data to your application about the results of your `NavPutFile` call.

`dialogOptions` A pointer to a structure of type `NavDialogOptions` (page 85). Before calling `NavPutFile`, you can set up this structure to specify dialog box settings. If you pass `NULL` in this parameter, Navigation Services uses the defaults for all options. See "Configuration Option Constants" (page 93) for a description of the default settings.

`eventProc` A Universal Procedure Pointer (UPP) of type `NavEventProcPtr` (page 92) that points to your application-defined event-handling function. Obtain this UPP by calling the function `NewNavEventProc`. Implementing an event-handling function allows your application to update windows after the user moves or resizes the dialog box. If you pass `NULL` in this parameter, the Save dialog box is not movable or resizable. For more information, see "Handling Events" (page 49) .

`fileType` A four-character code. Pass a file type code for the document to be saved.

`fileCreator` A four-character code. Pass a file creator code for the document to be saved.

`callBackUD` A pointer to a value set by your application. When the `NavPutFile` function calls your event-handling function, the `callBackUD` value is passed back to your application.

function result See "Result Codes for Navigation Services" (page 110) for a list of result codes.

NOTE

If you specify the `kNavDontResolveAliases` constant as a dialog box option, as described in "Configuration Option Constants" (page 93), before calling the `NavPutFile` function, Navigation Services returns a `paramErr (-50)`. ◆

DISCUSSION

After your application calls the `NavPutFile` function to display a Save dialog box and the user selects a location, enters a filename, and clicks OK, `NavPutFile` closes the dialog box and returns references to the file to be saved in the `NavReplyRecord` structure. Your application should check the `validRecord` field of

the `NavReplyRecord` structure; if this field is set to `true`, your application should save the file and call the function `NavCompleteSave` (page 76).

If you specify the Format pop-up menu in a dialog box displayed by the `NavPutFile` function, your application must provide adequate kind strings to describe the file types available. If the user uses the Format menu to save a file to a format other than the file's native format, Navigation Services translates the file automatically. If you wish to turn off automatic translation, set to `false` the value of the `translationNeeded` field of the `NavReplyRecord` structure you pass in the `reply` parameter. If you turn off automatic translation, your application is responsible for any required translation.

SEE ALSO

For a sample code listing, see "A Sample File-Saving Function" (page 35).

For more information on translation, see "Translating Files on Save" (page 34).

For more information on kind strings, see *Inside Macintosh: More Macintosh Toolbox*.

NavAskSaveChanges

Displays a Save Changes alert box.

```
OSErr NavAskSaveChanges (NavDialogOptions *dialogOptions,
                         NavAskSaveChangesAction action,
                         NavAskSaveChangesResult *reply,
                         NavEventUPP eventProc,
                         void *callBackUD);
```

`dialogOptions` A pointer to a structure of type `NavDialogOptions` (page 85). Before calling `NavAskSaveChanges`, set up this structure to specify dialog box settings. In this case, the `clientName` and `savedFileName` fields are the only two you must supply with values.

`action` A value of type `NavAskSaveChangesAction`. Pass a constant describing the user action that prompted the Save Changes

	alert box. For a description of the constants, see “Save Changes Request Constants” (page 109).
`reply`	A pointer to a structure of type `NavAskSaveChangesResult`. On return, the value describes the user’s response to the Save Changes alert box. For a description of the constants used to represent possible responses, see “Save Changes Action Constants” (page 108).
`eventProc`	A Universal Procedure Pointer (UPP) of type `NavEventProcPtr` (page 92) that points to your application-defined event-handling function. Obtain this UPP by calling the function `NewNavEventProc`. Implementing an event-handling function allows your application to update windows after the user moves or resizes the dialog box. If you pass `NULL` in this parameter, the Save Changes alert box is not movable. For more information, see “Handling Events” (page 49) .
`callBackUD`	A pointer to a value set by your application. When the `NavAskSaveChanges` function calls your event-handling function, the `callBackUD` value is passed back to your application.
function result	See “Result Codes for Navigation Services” (page 110) for a list of result codes.

DISCUSSION

This function is useful when your application needs to display an alert when the user attempts to close a document or an application with unsaved changes.

NavCustomAskSaveChanges

Displays a Save Changes alert box with a custom alert message.

```
OSErr NavCustomAskSaveChanges (NavDialogOptions *dialogOptions,
                               NavAskSaveChangesResult *reply,
                               NavEventUPP eventProc,
                               void *callBackUD);
```

`dialogOptions`	A pointer to a structure of type `NavDialogOptions` (page 85). Before calling `NavCustomAskSaveChanges`, set up this structure to specify dialog box settings. In this case, the `message` field is the only one you must supply with a value.
`reply`	A pointer to a structure of type `NavAskSaveChangesResult`. On return, the value describes the user's response to the Save Changes alert box. For a description of the constants used to represent possible responses, see "Save Changes Action Constants" (page 108).
`eventProc`	A Universal Procedure Pointer (UPP) of type `NavEventProcPtr` (page 92) that points to your application-defined event-handling function. Obtain this UPP by calling the function `NewNavEventProc`. Implementing an event-handling function allows your application to update windows after the user moves or resizes the dialog box. If you pass `NULL` in this parameter, the Save Changes alert box is not movable. For more information, see "Handling Events" (page 49) .
`callBackUD`	A pointer to a value set by your application. When the `NavCustomAskSaveChanges` function calls your event-handling function, the `callBackUD` value is passed back to your application.
function result	See "Result Codes for Navigation Services" (page 110) for a list of result codes.

DISCUSSION

This function is similar to the function `NavAskSaveChanges`, except you provide a custom alert message. This function is useful when you need to post a Save Changes alert box at times other than quitting or closing a file. Your application can display this alert box if a specified time interval has passed since the user last saved changes, for example.

NavAskDiscardChanges

Displays an alert box that asks the user whether to discard changes to a particular document.

```
OSErr NavAskDiscardChanges (NavDialogOptions *dialogOptions,
                            NavAskDiscardChangesResult *reply,
                            NavEventUPP eventProc,
                            void *callBackUD);
```

`dialogOptions`	A pointer to a structure of type `NavDialogOptions` (page 85). Before calling `NavAskDiscardChanges`, set up this structure to specify dialog box settings. In this case, the `savedFileName` field is the only one you must supply with a value.
`reply`	A pointer to a structure of type `NavAskDiscardChanges`. On return, the value describes the user's response to the Discard Changes alert box. For a description of the constants used to represent possible responses, see "Discard Changes Action Constants" (page 102).
`eventProc`	A Universal Procedure Pointer (UPP) of type `NavEventProcPtr` (page 92) that points to your application-defined event-handling function. Obtain this UPP by calling the function `NewNavEventProc`. Implementing an event-handling function allows your application to update windows after the user moves or resizes the dialog box. If you pass `NULL` in this parameter, the Discard Changes alert box is not movable. For more information, see "Handling Events" (page 49) .
`callBackUD`	A pointer to a value set by your application. When the `NavGetFile` function calls your event-handling function, the `callBackUD` value is passed back to your application.
function result	See "Result Codes for Navigation Services" (page 110) for a list of result codes.

DISCUSSION

If your application provides a Revert to Saved command, you can use the `NavAskDiscardChanges` function to display a confirmation alert box when a user selects Revert to Saved for a document with unsaved changes. Navigation

Services uses the string you supply in the `savedFileName` field of the `NavDialogOptions` structure you passed in the `dialogOptions` parameter to display the alert message, “Discard changes to <savedFilename>?”.

NavCompleteSave

Completes a save operation and performs any needed translation on the file.

```
OSErr NavCompleteSave (NavReplyRecord *translateInfo,
                       NavTranslateOptions howToTranslate);
```

`translateInfo` A pointer to a structure of type `NavReplyRecord` (page 83). Upon return, Navigation Services uses this structure to provide data to your application about the results of your `NavCompleteSave` call.

`howToTranslate` A pointer to a structure of type `NavTranslationOptions`. Pass one of two values to specify how to perform any needed translation. For a description of the constants you can use to represent these values, see “Translation Option Constants” (page 110). Translating in-place causes the source file to be replaced by the translation. Translating to a copy results in a file name followed by the string “(converted)” to avoid unwanted replacement. If you call the `NavCompleteSave` function in response to a Save a Copy command, you should pass the `kNavTranslateInPlace` constant in this parameter.

function result See “Result Codes for Navigation Services” (page 110) for a list of result codes. Since this function performs any needed translation, it may return a translation error.

DISCUSSION

You should always call `NavCompleteSave` to complete any file saving operation performed with the `NavPutFile` function. `NavCompleteSave` performs any needed translation, so you do not have to use the function `NavTranslateFile` (page 62) when saving. If you wish to turn off automatic translation, set to `false` the value of the `translationNeeded` field of the `NavReplyRecord` structure you pass in the `reply` parameter of the `NavPutFile` function. If you turn off automatic

translation, your application is responsible for any required translation. For more information, see “Saving Files” (page 31).

SEE ALSO

“Translating Files on Save” (page 34)

Handling Events and Customizing Dialog Boxes

You can use the following functions to handle events, draw custom previews, filter file objects, and add custom controls to Navigation Services dialog boxes.

- `MyEventProc` (page 77) describes an application-defined event-handling function.
- `MyPreviewProc` (page 78) describes an application-defined event-handling function.
- `MyFilterProc` (page 79) describes an application-defined event-handling function.
- `NavCustomControl` (page 81) allows your application to control various settings in Navigation Services dialog boxes.

MyEventProc

Handles events such as window updating and resizing.

This is how your application would define its own event-handling function if you chose to call it `MyEventProc`:

```
pascal void MyEventProc (NavEventCallbackMessage callBackSelector,
                         NavCBRecPtr callBackParms,
                         void *callBackUD );
```

`callBackSelector` One of the values specified by the `NavEventCallbackMessage` data type indicating which type of event your function must respond to. For a description of the constants that represent these values, see “Event Message Constants” (page 103).

`callBackParms`	A pointer to a structure of type `NavCBRec` (page 89). Your application uses the data supplied in this structure to begin processing the event.
`callBackUD`	A pointer to a value set by your application when it calls a Navigation Services function such as `NavGetFile`. When Navigation Services calls your event-handling function, the `callBackUD` value is passed back to your application in this parameter.

DISCUSSION

Register your event-handling function by passing a Universal Procedure Pointer (UPP) in the `eventProc` parameter of a Navigation Services function such as `NavGetFile` (page 58). You obtain this UPP by calling the macro `NewNavEventProc` and passing a pointer to your event-handling function. If you determine that the event is appropriate for your event-handling function, you can call other functions to handle custom control drawing, object filtering, or custom previews. For more information, see "Creating Application-Defined Functions" (page 48).

SPECIAL CONSIDERATIONS

When events involve controls, your event-handling function must respond to events only for your application-defined controls. To determine which control is affected by an event, pass the `kNavCtlGetFirstControlID` constant, described in "Custom Control Setting Constants" (page 96), in the `selector` parameter of the function `NavCustomControl` (page 81).

MyPreviewProc

Displays custom file previews. This is how your application would define its own preview function if you chose to call it `MyPreviewProc`:

```
pascal Boolean MyPreviewProc    (
                                NavCBRecPtr callBackParms,
                                void *callBackUD );
```

`callBackParms`	A pointer to a structure of type `NavCBRec` (page 89). Navigation Services uses this structure to provide data needed for your function to draw the preview.
`callBackUD`	A pointer to a value set by your application when it calls a Navigation Services function such as `NavGetFile`. When Navigation Services calls your preview function, the `callBackUD` value is passed back to your application in this parameter.
function result	A Boolean value. Your application returns `true` if your preview function successfully draws the custom file preview. If your preview function returns `false`, Navigation Services displays the preview if the file contains a valid `'pnot'` resource. If your preview function returns `false` and a `'pnot'` resource is not available, Navigation Services displays a blank preview area. For more information, see "Drawing Custom Previews" (page 52).

DISCUSSION

Register your preview function by passing the resulting Universal Procedure Pointer (UPP) in the `previewProc` parameter of a Navigation Services function, such as `NavGetFile` (page 58). You obtain this UPP by calling the function `NewNavPreviewProc` and passing a pointer to your preview-drawing function. When the user selects a file, Navigation Services calls your preview-drawing function. Your preview function, in turn, calls the function `NavCustomControl` (page 81) to determine if the preview area is visible and, if so, what its dimensions are. For more information, see "Drawing Custom Previews" (page 52).

MyFilterProc

Determines whether file objects should be displayed in the browser list and navigation menus. This is how your application would define its own filter function if you chose to call it `MyFilterProc`:

```
pascal Boolean MyFilterProc (AEDesc* theItem,
                             void* info,
                             void *callBackUD,
                             NavFilterModes filterMode);
```

`theItem` A pointer to an Apple event descriptor structure (`AEDesc`). Navigation Services uses this structure to provide information about the object being passed to your filter function.

IMPORTANT
Always check the Apple event descriptor type before deciding if an object needs to be filtered. Never assume that objects are type `'typeFSS'`, because the browser or pop-up menus may contain objects of other types. Make sure that your function only returns `false` if it recognizes the object. For more information, see "Obtaining Object Descriptions" (page 41). ▲

`info` A pointer to a structure of type `NavFileOrFolderInfo` (page 87). Navigation Services uses this structure to provide file or folder information about the item being passed to your filter function. This information is not valid for objects which are not HFS file objects.

`callBackUD` A pointer to a value set by your application when it calls a Navigation Services function such as `NavGetFile`. When Navigation Services calls your filter function, the `callBackUD` value is passed back to your application in this parameter.

`filterMode` A value representing which list of objects is currently being filtered. For a description of the constants used to represent these values, see "Object Filtering Constants" (page 107).

function result A Boolean value. If your application returns `true`, Navigation Services displays the object. If your application returns `false`, Navigation Services will not display the object.

DISCUSSION

Register your filter function by passing a Universal Procedure Pointer (UPP) in the `filterProc` parameter of a function such as `NavGetFile` (page 58). You obtain this UPP by calling the macro `NewNavObjectFilterProc` and passing a pointer to your filter function. Navigation Services calls your filter function to determine whether a file object should be displayed in the browser list or the pop-up menus.

If you use a filter function in conjunction with built-in translation, you should provide a list of file types in the `typeList` parameter of a file-opening function such as `NavGetFile` to inform Navigation Services which document types your application can open. If you provide a list of file types in the `typeList` parameter, take care to ensure that your filter function does not accidentally filter out any document type described in the list. For more information, see "Filtering File Objects" (page 50).

SPECIAL CONSIDERATIONS

If your filter function returns a result of `true`, Navigation Services displays the object. This is the opposite of Standard File filter functions.

NavCustomControl

Allows your application to control various settings in Navigation Services dialog boxes.

```
OSErr NavCustomControl (NavContext context,
                        NavCustomControlMessages selector,
                        void *parms );
```

`context` A value set by your application to provide context. When Navigation Services processes your `NavCustomControl` call and, in turn, calls your event-handling function, you can obtain this value from the `context` field of the structure of type `NavCBRec` (page 89) specified in the `callBackParms` parameter of your event-handling function.

`selector`	A value of type `NavCustomControlMessage`. Pass one or more of the constants representing the possible values used to control various aspects of the active dialog box. For a description of these constants, see "Custom Control Setting Constants" (page 96).
`parms`	A pointer to a configuration value. Some of the control setting constants passed in the `selector` parameter require that you provide an additional configuration value. For a description of which constants require configuration values, see "Custom Control Setting Constants" (page 96).
function result	See "Result Codes for Navigation Services" (page 110) for a list of result codes.

DISCUSSION

If you have provided an event-handling function and an event occurs in a Navigation Services dialog box, Navigation Services calls your event-handling function and specifies one of the constants described in "Event Message Constants" (page 103) in the `param` field of the structure of type `NavCBRec` (page 89). Navigation Services specifies this structure in the `callBackParms` parameter of your event-handling function. When Navigation Services supplies the `kNavCBStart` constant in the `param` field, your application can call the `NavCustomControl` function and pass one of the constants described in "Custom Control Setting Constants" (page 96) to control various aspects of the active Navigation Services dialog box. For example, your application can tell Navigation Services to sort the browser list by date by calling the `NavCustomControl` function and passing the `kNavCtlSortBy` constant in the `selector` parameter and a pointer to the `kNavSortDateField` configuration constant in the `parms` parameter. (Some of the `NavCustomControlMessage` constants do not require a corresponding configuration constant.)

SPECIAL CONSIDERATIONS

Navigation Services does not accept calls to the `NavCustomControl` function until an appropriate dialog box is fully initialized and displayed. Always check for the `kNavCBStart` constant, described in "Event Message Constants" (page 103), in the `param` field of the `NavCBRec` structure before calling the `NavCustomControl` function.

Note that your application can call the `NavCustomControl` function from within its event-handling function or its preview-drawing function.

SEE ALSO

"Adding Custom Controls" (page 46).

"Handling Events" (page 49).

Data Types for Navigation Services

NavReplyRecord

When your application calls a Navigation Services function that creates a dialog box, you pass a pointer to a `NavReplyRecord` structure. Navigation Services, in turn, uses this structure to provide your application with information about the user's interactions with the dialog box. When your application is through using the structure, remember to dispose of it by calling the function `NavDisposeReply` (page 57).

```
struct NavReplyRecord {
    UInt16      version;
    Boolean     validRecord;
    Boolean     replacing;
    Boolean     isStationery;
    Boolean     translationNeeded;
    AEDescList  selection;
    ScriptCode  keyScript;
    FileTranslationSpec **fileTranslation;
};
```

Field descriptions

`version` Identifies the version of this structure.

`validRecord` A Boolean value of `true` if the user closes a dialog box by pressing Return or Enter, or by clicking 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`	A Boolean value of `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 whether the file about to be saved should be saved as a stationery document.
`translationNeeded`	A Boolean value indicating whether translation was or will be needed for files selected in Open and Save dialog boxes.
`selection`	An Apple event descriptor list (`AEDescList`) created from 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` (page 57). 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 HFS file object 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. For more information, see "Obtaining Object Descriptions" (page 41). For more information on Apple event structures and functions, see *Inside Macintosh: Interapplication Communication*. For more information on `FSSpec` structures, see *Inside Macintosh: Files*.
`keyScript`	The keyboard script system used for the filename. For information on script codes, see *Inside Macintosh: Text*.
`fileTranslation`	A handle to a Translation Manager structure of type `FileTranslationSpec`. 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 (page 93). 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 your 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. For more information, see "Translating Files on Open" (page 21) and "Translating Files on Save" (page 34). For information on the FileTranslationSpec structure, see *Inside Macintosh: More Macintosh Toolbox*.

NavDialogOptions

The NavDialogOptions structure contains dialog box configuration settings you can pass to several Navigation Services functions.

```
struct NavDialogOptions {
    UInt16 version;
    NavDialogOptionFlags dialogOptionFlags;
    Point   location;
    Str255  clientName;
    Str255  windowTitle;
    Str255  actionButtonLabel;
    Str255  cancelButtonLabel;
    Str255  savedFileName;
    Str255  message;
    UInt32  preferenceKey;
    Handle  popupExtension;
};
```

Field descriptions

`version` Identifies the version of this structure. Be sure to specify the kNavDialogOptionsVersion constant in this field.

`dialogOptionsFlags` One of several constants defined by the NavDialogOptionFlags data type as described in "Configuration Option Constants" (page 93).

`location`	The upper-left location of the dialog box (in global coordinates). If you set the `dialogOptionFlags` field to `NULL` or set this field to (-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. For more information, see “Persistence” (page 17).
`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 button. If you do not specify a label, the button will use the default label (Open or Save, for example.)
`cancelButtonLabel`	An alternative button title for the Cancel button in dialog boxes.
`savedFileName`	The default filename 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 expands 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` (page 58) 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` (page 87) 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).

For more information, see “Customizing Type Pop-up Menus” (page 45).

NavMenuItemSpec

Your application uses the `NavMenuItemSpec` structure to define additional items in an Open dialog box’s Show pop-up menu or a Save dialog box’s Format pop-up menu. For a description of how to add menu items, see “Customizing Type Pop-up Menus” (page 45). For information about file creators and file types, see *Inside Macintosh: Macintosh Toolbox Essentials*.

```
struct NavMenuItemSpec {
    UInt16  version;
    OSType  menuCreator;
    OSType  menuType;
    Str255  menuItemName;
};
```

Field descriptions

`version` Identifies the version of this structure. Be sure to specify the `kNavMenuItemSpecVersion` constant in this field.

`menuCreator` A value set by your application. This is a unique tag that Navigation Services passes back to your application to identify the application type of the selected menu item.

`menuType` A value set by your application. This is a unique tag that Navigation Services passes back to your application to identify the file type of the selected menu item.

`menuItemName` The item name that appears in the pop-up menu.

NavFileOrFolderInfo

The `NavFileOrFolderInfo` structure contains file or folder information for use by your application-defined filter function. For more information, see “Filtering File Objects” (page 50). Your filter function can determine whether the currently selected object is a file by checking the `isFolder` field of the `NavFileOrFolderInfo`

structure for the value `false`. After making this determination, you can obtain more information about the object from the `fileAndFolder` structure.

Note
The information in this structure is valid only for HFS file objects. ◆

```
struct NavFileOrFolderInfo {
    UInt16  version;
    Boolean isFolder;
    Boolean visible;
    UInt32  creationDate;
    UInt32  modificationdate;
    union {
        struct {
            Boolean     locked;         /* file is locked */
            Boolean     resourceOpen;   /* resource fork is open */
            Boolean     dataOpen;       /* data fork is open */
            Boolean     reserved1;
            UInt32      dataSize;       /* size of data fork */
            UInt32      resourceSize;   /* size of resource fork */
            FInfo       finderInfo;
            FXInfo      finderXInfo;
        }           fileInfo;
        struct {
            Boolean     shareable;      /* volume can be shared */
            Boolean     sharePoint;     /* volume is being shared */
            Boolean     mounted;        /* volume is mounted */
            Boolean     readable;       /* volume is readable */
            Boolean     writeable;      /* volume is writeable */
            Boolean     reserved2;
            UInt32      numberOfFiles;  /* number of files in folder */
            DInfo       finderDInfo;
            DXInfo      finderDXInfo;
            char        reserved3[214]; /*
        }           folderInfo
    }           fileAndFolder;
};

typedef struct NavFileOrFolderInfo  NavFileOrFolderInfo;
```

Field descriptions

`version`	Identifies the version of this structure.
`isFolder`	A Boolean value. If this value is set to `true`, the object being described is a folder or volume; otherwise, the value is set to `false`. An alias to a folder or volume returns `true`. Check for the `kIsAlias` constant in the `FInfo` field to determine whether an object is an alias.
`visible`	A Boolean value. If this value is set to `true`, the object being described is visible in the browser list; otherwise, the value is set to `false`.
`creationDate`	The creation date of the object being described.
`modificationDate`	The modification date of the object being described.
`fileAndFolder`	A union of the `fileInfo` and `folderInfo` fields.

NavCBRec

The `NavCBRec` structure provides customization information for your application-defined functions.

```
struct NavCBRec {
    UInt16 version;
    NavContext context;
    WindowPtr window;
    Rect customRect;
    Rect previewRect;
    NavEventDataInfo eventData;
};
```

Field descriptions

`version`	Identifies the version of this structure.
`context`	A pointer to a value set by your application. Your application passes this value in the `context` parameter of the function `NavCustomControl` (page 81). When Navigation Services calls your event-handling function, your application can obtain this value from the `context` field.

`window`	A pointer to the Navigation Services dialog box that generated the call to your application-defined function.
`customRect`	A local coordinate rectangle describing the customization area available to your application. This determines how much room your application has to install custom controls. For more information on using this field, see “Adding Custom Controls” (page 46).
`previewRect`	A local coordinate rectangle describing the preview area available to your application’s preview function. The minimum size is 145 pixels wide by 118 pixels high.
`eventData`	A structure of type `NavEventData` (page 90).

NavEventData

The `NavEventData` structure contains a structure of type `NavEventDataInfo` (page 91); in Navigation Services 1.1, the `NavEventData` structure also contains a field describing the dialog box item last clicked by the user.

```
struct NavEventData {
    NavEventDataInfo eventDataParms;
    SInt16 itemHit;                     /* Nav Svcs 1.1 ONLY */
};

typedef struct NavEventData     NavEventData
```

Field descriptions

`eventDataParms`	A structure of type `NavEventDataInfo` (page 91).
`itemHit`	A signed integer value. On return, this value represents the item number of the dialog box item last clicked by the user. If the user clicks something other than a valid Navigation Services-generated control item, this value will be -1. Use the function `NavLibraryVersion` (page 54) to ensure Navigation Services 1.1 is installed before you attempt to use this data.

NavEventDataInfo

The `NavEventDataInfo` structure provides event-handling data to your application.

```
union NavEventDataInfo {
    EventRecord *event;
    void *param;
};

typedef struct NavEventDataInfo     NavEventDataInfo;
```

Field descriptions

`event` A pointer to the `EventRecord` structure describing an event to be handled by your event-handling function.

`param` A pointer to additional event data. In most cases, this data consists of an Apple event descriptor list (`AEDescList`) for the file or files affected by the event described in the `event` field. For example, if the event consists of the user making a selection in the browser list, the `AEDescList` specifies the file or files selected.

NavTypeList

Your application uses the `NavTypeList` structure to define a list of file types that your application is capable of opening. Your application passes a pointer to this list to Navigation Services functions that display Open or Save dialog boxes. You may create this list dynamically or reference a Translation Manager `'open'` resource.

For a description of how to use the `NavTypeList` structure, see “Providing File Opening Options” (page 19). For more information on the `'open'` resource and the Translation Manager, see the “Translation Manager” chapter in *Inside Macintosh: More Macintosh Toolbox*.

```
struct NavTypeList {
    OSType  componentSignature;
    short   reserved;
```

```
    short   osTypeCount;
    OSType  osType[1];
};
```

Field descriptions

`componentSignature`	Your application signature.
`osTypeCount`	Number indicating how many file types are defined.
`osType`	A list of file types your application can open.

NavEventProcPtr

If you create an event-handling function, it must accept certain input parameters, as defined by the `NavEventProcPtr` data type. For more information on implementing an event-handling function, see `MyEventProc` (page 77). For a sample code listing, see "A sample event-handling function" (page 49).

```
typedef pascal void (*NavEventProcPtr)
                    ( NavEventCallbackMessage callBackSelector,
                    NavCBRecPtr callBackParms,
                    void *callBackUD);
```

NavPreviewProcPtr

If you create a preview function, it must accept certain input parameters, as defined by the `NavPreviewProcPtr` data type. For more information on implementing a preview function, see `MyPreviewProc` (page 78).

```
typedef pascal Boolean  (*NavPreviewProcPtr)
                        ( NavCBRecPtr callBackParms,
                        void *callBackUD );
```

NavFilterProcPtr

If you create a filter function, it must accept certain input parameters, as defined by the `NavFilterProcPtr` data type. For more information on implementing a filter function, see `MyFilterProc` (page 79). For a sample code listing, see "A sample filter function" (page 51).

```
typedef pascal Boolean (*NavObjectFilterProcPtr)
                        (AEDesc *theItem,
                        void *info,
                        void *callBackUD,
                        NavFilterModes filterMode);
```

Constants for Navigation Services

Configuration Option Constants

In the `dialogOptionFlags` field of the structure of type `NavDialogOptions` (page 85), you can specify values for Navigation Services dialog box configuration options. Use the constants defined by the `NavDialogOptionFlags` data type to specify these values.

```
enum NavDialogOptionFlags {
    kNavDefaultNavDlogOption    = 0x000000E4,/* use default options */
    kNavNoTypePopup             = 0x00000001,/* don't show type pop-up*/
    kNavDontAutoTranslate       = 0x00000002,/* don't auto-translate */
    kNavDontAddTranslateItems   = 0x00000004,/* don't add translation
                                                choices */
    kNavAllFilesInPopup         = 0x00000010,/* add All Files
                                                menu item */
    kNavAllowStationery         = 0x00000020,/* allow stationery files */
    kNavAllowPreviews           = 0x00000040,/* allow previews */
    kNavAllowMultipleFiles      = 0x00000080,/* allow mult. selection*/
    kNavAllowInvisibleFiles     = 0x00000100,/* show invisible objects*/
    kNavDontResolveAliases      = 0x00000200,/* don't resolve aliases */
    kNavSelectDefaultLocation   = 0x00000400,/* make default location
```

```
                                                    the browser selection */
    kNavSelectAllReadableItem   = 0x00000800/* make All Readable Items
                                                    default selection */
};
```

Constant descriptions

`kNavDefaultNavDlogOptions`
Tells Navigation Services to use default configuration options. These default options are

- no custom control titles
- no banner or prompt message
- automatic resolution of aliases
- support for file previews
- no display of invisible file objects
- support for multiple file selection
- support for stationery

`kNavNoTypePopup` Tells Navigation Services not to display the Show pop-up menu in the Open dialog box or the Format pop-up menu in the Save dialog box.

`kNavDontAutoTranslate`
Tells Navigation Services that your application will handle file translation. Normally a file chosen in an Open dialog box that requires translation is automatically translated. Navigation Services informs your application that a file needs translating by setting the `translationNeeded` field of the structure `NavReplyRecord` (page 83) to `true`. A translation specification array, specified in the `fileTranslation` field of the `NavReplyRecord` structure, contains the associated translation specification records. When you set the `kNavDontAutoTranslate` flag, your application is responsible for translation, either by calling the function `NavTranslateFile` (page 62) or by performing the translation itself. For more information, see "Translating Files on Open" (page 21).

`kNavDontAddTranslateItems`
Tells Navigation Services not to display file translation

options in the Show pop-up menu. For more information, see “Translating Files on Open” (page 21).

`kNavAllFilesInPopup`

Tells Navigation Services to add a pop-up menu item called All Documents, so the user can see a display of all files in the current directory.

`kNavAllowStationery`

Tells Navigation Services to display a Stationery Option command in the Format pop-up menu of Save dialog boxes, so users can choose to save a file as a document or as stationery. This is a default option. For more information, see “Providing File Format Options” (page 33).

`kNavAllowPreviews` Tells Navigation Services to provide previews, when available, of selected files. This is a default option. See also “Drawing Custom Previews” (page 52).

`kNavAllowMultipleFiles`

Tells Navigation Services to allow users to select and open multiple files in the browser list by shift-clicking or using the Select All command. If you don’t specify this constant, users can select multiple files for drag-and-drop operations, but the default button (normally titled Open) is disabled when multiple items are selected.

`kNavAllowInvisibleFiles`

Tells Navigation Services to show invisible file objects in the browser list.

`kNavDontResolveAliases`

Tells Navigation Services not to resolve any alias selected by the user. If the user selects an alias with this option set, the file system specification returned by Navigation Services designates the alias file instead of its referenced original. If you specify this constant before calling the function `NavPutFile` (page 70), Navigation Services returns a result code of `paramErr` (-50).

`kNavSelectDefaultLocation`

Tells Navigation Services to select the default location in the browser list. By default, Navigation Services will open the browser list with the default location displayed, not selected. For example, if you define the System Folder as the default location and specify the

`kNavSelectDefaultLocation` constant, the System Folder appears as the current selection in the browser list. Without this constant, the browser list displays the contents of the System Folder.

`kNavSelectAllReadableItem`

Tells Navigation Services to show All Readable Documents as the default selection in the Show pop-up menu when the Open dialog box is first displayed. If you do not specify this constant, Navigation Services shows the All <AppName> Documents menu item as the default selection in the Show pop-up menu when the Open dialog box is first displayed.

Custom Control Setting Constants

The `NavCustomControlMessage` data type defines constants that your application can pass in the `selector` parameter of the `NavCustomControl` function (page 81) to control various aspects of the active dialog box.

```
enum {
    kNavCtlShowDesktop          = 0,/*  show desktop, parms = nil */
    kNavCtlSortBy               = 1,/*  set sort key field,
                                        parms->NavSortKeyField */
    kNavCtlSortOrder            = 2,/*  set sort order,
                                        parms->NavSortOrder */
    kNavCtlScrollHome           = 3,/*  scroll list home,parms = nil */
    kNavCtlScrollEnd            = 4,/*  scroll list end,parms = nil */
    kNavCtlPageUp               = 5,/*  page list up,parms = nil */
    kNavCtlPageDown             = 6,/*  page list down,parms = nil */
    kNavCtlGetLocation          = 7,/*  get current location,
                                        parms<-AEDesc */
    kNavCtlSetLocation          = 8,/*  set current location,
                                        parms->AEDesc */
    kNavCtlGetSelection         = 9,/*  get current selection,
                                        parms<-AEDescList */
    kNavCtlSetSelection         = 10,/* set current selection,
                                        parms->AEDescList*/
    kNavCtlShowSelection        = 11,/* make selection visible,
                                        parms = nil */
    kNavCtlOpenSelection        = 12,/* open view of selection,
```

```
                                        parms = nil */
    kNavCtlEjectVolume          = 13,/* eject volume,parms->vRefNum */
    kNavCtlNewFolder            = 14,/* create a new folder,
                                        parms->StringPtr */
    kNavCtlCancel               = 15,/* cancel dialog,parms = nil */
    kNavCtlAccept               = 16,/* accept default,parms = nil */
    kNavCtlIsPreviewShowing     = 17,/* get preview status,
                                        parms<-Boolean */
    kNavCtlAddControl           = 18,/* add 1 control,
                                        parms->ControlHandle */
    kNavCtlAddControlList       = 19,/* add control list to dialog,
                                        parms->Handle (DITL rsrc) */
    kNavCtlGetFirstControlID    = 20,/* get 1st cntrl ID,parms<-UInt16 */
    kNavCtlSelectCustomType     = 21,/* select a custom menu item,
                                        parms->NavMenuItemSpec */
    kNavCtlSelectAllType        = 22,/* select "All" menu item,
                                        parms->SInt16 */
    kNavCtlGetEditFileName      = 23,/* get save filename,
                                        parms<-StringPtr */
    kNavCtlSetEditFileName      = 24,/* set save filename,
                                        parms->StringPtr */
    kNavCtlSelectEditFileName   = 25 /* select save filename,
                                       parms->ControlEditTextSelectionRec
                                       (v 1.1 only) */
};

typedef SInt32  NavCustomControlMessage;
```

Constant descriptions

`kNavCtlShowDesktop` Tells Navigation Services to change the browser list location to the desktop.

`kNavCtlSortBy` Alerts Navigation Services that your application is setting a sort key in the browser list. In addition to the `kNavCtlSortBy` constant, your application passes one of the `NavSortKeyField` constants in the `parms` parameter of the function `NavCustomControl` (page 81). For a description of the `NavSortKeyField` constants, see “File Sorting Constants” (page 106).

`kNavCtlSortOrder` Alerts Navigation Services that your application is setting sort order, either ascending or descending, in the browser

list. In addition to passing the `kNavCtlSortOrder` constant, your application must pass one of the `NavSortOrder` constants in the `parms` parameter of the `NavCustomControl` function. For a description of the `NavSortOrder` constants, see "Sort Order Constants" (page 109).

`kNavCtlScrollHome` Tells Navigation Services to scroll the browser to the top of the file list.

`kNavCtlScrollEnd` Tells Navigation Services to scroll the browser to the bottom of the file list.

`kNavCtlPageUp` Tells Navigation Services to scroll the browser up one page length as a result of the user clicking the scroll bar above the scroll box.

`kNavCtlPageDown` Tells Navigation Services to scroll the browser down one page length as a result of the user clicking the scroll bar below the scroll box.

`kNavCtlGetLocation` Tells Navigation Services to return the current location. When you specify this constant, the `parms` parameter of the `NavCustomControl` function returns a pointer to an `AEDesc` structure describing the current location. For more information on parsing `AEDesc` structures, see "Obtaining Object Descriptions" (page 41).

`kNavCtlSetLocation` Tells Navigation Services that your application wishes to set the location being viewed in the browser list. In addition to specifying the `kNavCtlSetLocation` constant, your application passes a pointer to an `AEDesc` structure describing the new location in the `parms` parameter of the `NavCustomControl` function.

`kNavCtlGetSelection`

Tells Navigation Services to return the selected item or items in the browser. When you specify this constant, the `parms` parameter of the `NavCustomControl` function returns a pointer to an `AEDescList` structure describing the selected item(s). For more information on parsing `AEDescList` structures, see "Obtaining Object Descriptions" (page 41). If the user deselects the current selection, the `AEDescList` returned by Navigation Services contains an empty reference. You can account for this case by using the function `AECountItems` and checking for a zero count.

`kNavCtlSetSelection`
: Tells Navigation Services to change the browser list selection. In addition to specifying the `kNavCtlSetSelection` constant, your application must pass a pointer to an `AEDescList` structure describing the selection in the `parms` parameter of the `NavCustomControl` function. If you want to deselect the current selection without making a new selection, pass `NULL` for the pointer.

`kNavCtlShowSelection`
: Tells Navigation Services to make the current selection visible in the browser list if the selection has been scrolled out of sight by the user.

`kNavCtlOpenSelection`
: Tells Navigation Services to open the current selection.

`kNavCtlEjectVolume`
: Tells Navigation Services to eject a volume. In addition to specifying this constant, you must pass a pointer to the volume reference number (`vRefNum`) of the volume to be ejected in the `parms` parameter of the `NavCustomControl` function.

`kNavCtlNewFolder`
: Tells Navigation Services to create a new folder in the current location. In addition to specifying the `kNavCtlNewFolder` constant, your application passes a string representing the name of the new folder in the `parms` parameter of the `NavCustomControl` function.

`kNavCtlCancel`
: Tells Navigation Services to dismiss the Open or Save dialog box as if the user had pressed the Cancel button.

`kNavCtlAccept`
: Tells Navigation Services to close the Open or Save dialog box as if the user had pressed the Open or Save button. Navigation Services does not act on this constant if there is no current selection.

`kNavCtlIsPreviewShowing`
: Asks Navigation Services if the preview area is currently available. If you specify this constant, Navigation Services returns a pointer to a Boolean value of `true` in the `parms` parameter of the `NavCustomControl` function if the preview area is available.

`kNavCtlAddControl`
: Tells Navigation Services to add one application-defined control to Open or Save dialog boxes. In addition to sending this message, your application passes a control

handle in the `parms` parameter of the `NavCustomControl` function. Design the control in local coordinates.

IMPORTANT

To avoid any unnecessary flickering or redrawing, ensure the control is initially invisible before specifying this constant. You may set the control to visible after Navigation Services supplies the `kNavCBStart` constant, described in "Event Message Constants" (page 103), in the `param` field of the structure of type `NavCBRec` (page 89). If the user resizes the dialog box, your application must move the control because it is not maintained by Navigation Services. If you use the `kNavCtlAddControlList` constant (described next) and you supply a `'DITL'` resource, you avoid the need to move the control yourself. ▲

`kNavCtlAddControlList`

Tells Navigation Services to add a list of application-defined dialog box items to Open or Save dialog boxes. In addition to specifying this constant, your application must pass a handle to a dialog box item list or `'DITL'` resource in the `parms` parameter of the `NavCustomControl` function. Design the `'DITL'` resource in local coordinates. Navigation Services will add the custom items relative to the upper left corner of the customization area. If the user resizes the dialog box, your custom items are moved automatically.

`kNavCtlGetFirstControlID`

Asks Navigation Services to help you identify the first custom control in the dialog box, in order to determine which custom control item was selected by the user. The `parms` parameter of the `NavCustomControl` function returns a pointer to a 16-bit integer that indicates the item number of the first custom control. In your event-handling function, use the Dialog Manager function `FindDialogItem` to find out which item was selected. The `FindDialogItem` function returns 0 for the first item, 1 for the second and so on. To get the proper item number, add 1 to the `FindDialogItem` function result. The Open or Save dialog box's standard controls precede yours, so use the formula (itemHit - yourFirstItem + 1) to determine which of your items was

selected. Your application should not depend on any hardcoded value for the number of items, since this value is likely to change in the future.

IMPORTANT

Take care to test the result from `FindDialogItem` to ensure that it describes a control that you defined. Your application must not respond to any controls that do not belong to it. ▲

`kNavCtlSelectCustomType`

Tells Navigation Services to set one of your custom menu items in the Show pop-up menu or the Format pop-up menu as the default selection. This is useful if you want to override the default pop-up menu selection. In addition to specifying this constant, pass a pointer to a structure of type `NavMenuItemSpec` (page 87) in the `parms` parameter of the `NavCustomControl` function. This structure describes the item you wish to have selected. For more information on providing custom menu items, see "Customizing Type Pop-up Menus" (page 45).

`kNavCtlSelectAllType`

Tells Navigation Services to override the default menu item in the Type pop-up menu. By specifying one of the `NavPopupMenuItem` constants, described in "Menu Item Selection Constants" (page 106), in the `parms` parameter of the function `NavCustomControl` (page 81), you can set the default item to All <AppName> Documents, All Readable Documents or All Documents.

`kNavCtlGetEditFileName`

Tells Navigation Services to return the name of the file to be saved by the function `NavPutFile` (page 70). This would be useful if you wanted to automatically add an extension to the filename, for example. When you send this message, the `parms` parameter of the `NavCustomControl` function returns a `StringPtr` to a Pascal string containing the filename.

`kNavCtlSetEditFileName`

Tells Navigation Services that your application wishes to set the name of the file to be saved by the function `NavPutFile` (page 70). Your application normally specifies

the `KNavCtlSetEditFileName` constant after modifying the filename obtained by specifying the `kNavCtlGetEditFileName` constant. In addition to specifying the `kNavCtlSetEditFileName` constant, your application must pass a `StringPtr` to a Pascal string containing the filename in the `parms` parameter of the `NavCustomControl` function.

`kNavCtlSelectEditFileName`

(Navigation Services 1.1 **ONLY**)
Tells Navigation Services to display the name of the file to be saved by the function `NavPutFile` (page 70) with some or all of the filename string highlighted for selection. In addition to specifying the `kNavCtlSelectEditFileName` constant, your application passes a Control Manager structure of type `ControlEditTextSelectionRec` in the `parms` parameter of the `NavCustomControl` function in order to specify which part of the filename string to highlight. For more information on the `ControlEditTextSelectionRec` structure, see *Mac OS 8 Control Manager Reference*.

Discard Changes Action Constants

In the `reply` parameter of the `NavAskDiscardChanges` function (page 74), your application receives a value represented by one of the constants defined by the `NavAskDiscardChangesResult` data type.

```
enum {
    kNavAskDiscardChanges       = 1,
    kNavAskDiscardChangesCancel = 2
};
typedef UInt32  NavAskDiscardChangesResult;
```

Constant descriptions

`kNavAskDiscardChanges`

User clicked the Okay button.

`kNavAskDiscardChangesCancel`

User clicked the Cancel button.

Event Message Constants

Your application can provide an event-handling function to update application windows and handle other events related to your Open and Save dialog boxes. Navigation Services informs you of pertinent events by supplying event message constants to your event-handling function via the `param` field of the structure of type `NavCBRec` (page 89). These constants are defined in the `NavEventCallbackMessages` data type. For more information, see "Handling Events" (page 49).

```
enum {
    kNavCBEvent             = 0,    /*  event has occurred */
    kNavCBCustomize         = 1,    /*  signal to negotiate
                                        customization space */
    kNavCBStart             = 2,    /*  nav dialog box starting up */
    kNavCBTerminate         = 3,    /*  nav dialog box closing down */
    kNavCBAdjustRect        = 4,    /*  nav dialog box being resized */
    kNavCBNewLocation       = 5,    /*  user chose new location*/
    kNavCBShowDesktop       = 6,    /*  user navigated to the desktop */
    kNavCBSelectEntry       = 7,    /*  user made selection in browser */
    kNavCBPopupMenuSelect   = 8,    /*  user made popup menu selection */
    kNavCBAccept            = 9,    /*  user accepted nav dialog box */
    kNavCBCancel            = 10,   /*  user cancelled nav dialog box */
    kNavCBAdjustPreview     = 11    /*  preview toggled or resized */
}
typedef UInt32 NavEventCallbackMessages;
```

Constant descriptions

`kNavCBEvent` Tells your application that an event has occurred (including an idle event), which provides an opportunity for your application to track controls, update other windows, and so forth. Your application can obtain the event record describing this event from the `event` field of the structure of type `NavCBRec` (page 89). The `kNavCBEvent` constant is the only message that needs to be processed by most applications that do not customize Open and Save dialog boxes.

`kNavCBCustomize` Tells your application to supply a dialog box customization request. The `customRect` field of the `NavCBRec` structure defines a rectangle in the local coordinates of the dialog box; the top-left coordinates define the anchor point for a

customization rectangle. If you want to customize the dialog box, your application responds to the `kNavCBCustomize` message by setting a value in the `customRect` field that completes the dimensions of the customization rectangle. After your application responds, Navigation Services inspects the `customRect` field to determine if the requested dimensions result in a dialog box that can fit on the screen. If the dimensions are too large, then Navigation Services responds by setting the rectangle to the largest size that the screen can accommodate. Your application can continue to "negotiate" by examining the `customRect` field and requesting a different size until Navigation Services provides an acceptable rectangle value, at which time you should create your custom control or item list. The minimum size for the customization area is 400 pixels wide by 40 pixels high. For more information, see "Adding Custom Controls" (page 46).

NOTE

Don't add new dialog box items until your application receives the `kNavCBStart` event message constant. ◆

`kNavCBStart` Tells your application that a dialog box is ready to be displayed. After receiving the `kNavCBCustomize` event message constant, your event-handling function should wait for the `kNavCBStart` event message constant to ensure that your application can safely add dialog items. No additional data is provided to your application with this constant. For more information, see "Adding Custom Controls" (page 46).

`kNavCBTerminate` Tells your application that the dialog box is about to be closed, which means you must remove any user-interface items that were created in response to the `kNavCBStart` message. No additional data is provided to your application with this constant.

`kNavCBAdjustRect` Tells your application that the dialog box has been resized and the customization rectangle has been accordingly resized. Use the `customRect` field from the structure of type `NavCBRec` (page 89) to determine the new customization rectangle size. Your application does not need to offset the

controls; Navigation Services moves them automatically. Your application is responsible for any redrawing of the controls or handling events beyond moving the controls, however. For more information, see “Adding Custom Controls” (page 46).

`kNavCBNewLocation` Tells your application that a new location is being viewed in the dialog box. The `param` field of the `NavCBRec` structure contains a pointer to an `AEDesc` structure of type `'typeFSS'` describing the new location. This pointer is valid only during the execution of your event-handling function. For more information on how to parse `AEDesc` structures, see “Obtaining Object Descriptions” (page 41).

`kNavCBShowDesktop` Tells your application that the Open or Save dialog box is showing the desktop view, consisting of the composite of all desktop folders from all mounted volumes. The `param` field of the `NavCBRec` structure contains a pointer to an `AEDescList` structure identifying the desktop location. This pointer is valid only during the execution of your event-handling function. For more information on how to parse `AEDesc` structures, see “Obtaining Object Descriptions” (page 41).

`kNavCBSelectEntry` Tells your application that an entry in the browser list has been selected or deselected by the user. The `param` field of the `NavCBRec` structure contains a pointer to an `AEDescList` record of type `'typeFSS'` identifying the current selection. If the user deselects the current selection, the `AEDescList` record contains an empty reference. This pointer is valid only during the execution of your event-handling function. For more information on how to parse `AEDesc` structures, see “Obtaining Object Descriptions” (page 41).

`kNavCBPopupMenuSelect`
Tells your application that a selection was made from the Open dialog box’s Show pop-up menu or Save dialog box’s Format pop-up menu. The `param` field of the `NavCBRec` structure contains a pointer to a structure of type `NavMenuItemSpec` (page 87) describing the pop-up menu item selected. This data is valid only during the execution of your event-handling function.

`kNavCBAccept` Tells your application that the user has pressed the Accept button.

`kNavCBCancel` Tells your application that the user has pressed the Cancel button.

`kNavCBAdjustPreview`
Tells your application that the user has toggled the preview area on or off. The `param` field of the `NavCBRec` structure contains a pointer to a Boolean value of `true` if the preview area is toggled on and `false` if toggled off. This information is useful if your application creates custom controls in the preview area. For more information, see "Drawing Custom Previews" (page 52).

File Sorting Constants

Your application can determine the sort key for displayed files by passing the `kNavCtlSortBy` constant, described in "Custom Control Setting Constants" (page 96), in the `selector` parameter of the function `NavCustomControl` (page 81), and passing one of the constants defined in the `NavSortKeyField` data type in the `parms` parameter of the `NavCustomControl` function.

```
enum {
    kNavSortNameField = 0,
    kNavSortDateField = 1
};
typedef UInt16  NavSortKeyField;
```

Constant descriptions

`kNavSortNameField` Sort by filename.

`kNavSortDateField` Sort by modification date.

Menu Item Selection Constants

To set the default selection for the Show pop-up menu of an Open dialog box, your application passes the `kNavCtlSelectAllType` constant, described in "Custom Control Setting Constants" (page 96), in the `selector` parameter of the `NavCustomControl` function (page 81) and passes one of the constants defined in the `NavPopupMenuItem` data type in the `parms` parameter of the `NavCustomControl` function.

```
enum {
    kNavAllKnownFiles       = 0,
    kNavAllReadableFiles    = 1,
    kNavAllFiles            = 2
};
typedef UInt16  NavPopupMenuItem;
```

Constant descriptions

`kNavAllKnownFiles` Tells Navigation Services to display all files identified as readable or translatable by your application. For more information on how to identify readable files, see “Providing File Opening Options” (page 19). For more information on how to specify file types for translation, see “Translating Files on Open” (page 21).

`kNavAllReadableFiles` Tells Navigation Services to display all files identified as readable by your application. For more information on how to identify readable files, see “Providing File Opening Options” (page 19).

`kNavAllFiles` Tells Navigation Services to display all files.

Object Filtering Constants

Navigation Services passes one of the constants defined by the `NavFilterModes` data type to the `filterMode` parameter of your application-defined filter function to tell your application whether the browser list or one of the navigation option pop-up menus contains the object currently being filtered. For more information, see “Filtering File Objects” (page 50).

```
enum {
    kNavFilteringBrowserList        = 0,
    kNavFilteringFavorites          = 1,
    kNavFilteringRecents            = 2,
    kNavFilteringShortCutVolumes    = 3,
    kNavFilteringLocationPopup      = 4
};
typedef UInt32  NavFilterModes;
```

Constant descriptions

`kNavFilteringBrowserList`
The browser list contains the object being filtered.

`kNavFilteringFavorites`
The Favorites pop-up menu contains the object being filtered.

`kNavFilteringRecents`
The Recent pop-up menu contains the object being filtered.

`kNavFilteringShortCutVolumes`
The Shortcuts pop-up menu contains the object being filtered.

`kNavFilteringLocationPopup`
The object being filtered is the path described by the Location menu.

Save Changes Action Constants

In the `reply` parameter of the function `NavAskSaveChanges` (page 72), your application receives a value represented by one of the constants defined by the `NavAskSaveChangesResult` data type.

```
enum {
    kNavAskSaveChangesSave      = 1,
    kNavAskSaveChangesCancel    = 2,
    kNavAskSaveChangesDontSave  = 3
};
typedef UInt32  NavAskSaveChangesResult;
```

Constant descriptions

`kNavAskSaveChangesSave`
User clicked the Save button.

`kNavAskSaveChangesCancel`
User clicked the Cancel button.

`kNavAskSaveChangesDontSave`
User clicked the Don’t Save button.

Save Changes Request Constants

Your application requests a Save Changes alert box by specifying one of the following constants, defined by the `NavAskSaveChangesAction` data type, in the `action` parameter of the function `NavAskSaveChanges` (page 72).

```
enum {
    kNavSaveChangesClosingDocument      = 1,
    kNavSaveChangesQuittingApplication  = 2,
    kNavSaveChangesOther                = 0
};
typedef UInt32  NavAskSaveChangesAction;
```

Constant descriptions

`kNavSaveChangesClosingDocument`
Requests a Save Changes alert box that asks the user whether to save changes when closing a document.

`kNavSaveChangesQuittingApplication`
Requests a Save Changes alert box that asks the user whether to save changes when quitting your application.

`kNavSaveChangesOther`
Requests a Save Changes alert box that asks the user whether to save changes at some time other than closing or quitting. This is useful when your application prompts the user to save documents at timed intervals, for example.

Sort Order Constants

Your application can specify the sort order for displayed files by passing the `kNavCtlSortOrder` constant (page 97) in the `selector` parameter of the `NavCustomControl` function (page 81) and passing one of the constants defined in the `NavSortOrder` data type in the `parms` parameter of the `NavCustomControl` function.

```
enum {
    kNavSortAscending   = 0,
    kNavSortDescending  = 1
};
typedef UInt16  NavSortOrder;
```

Constant descriptions

`kNavSortAscending` Sort in ascending order.

`kNavSortDescending` Sort in descending order.

Translation Option Constants

Your application passes one of the `NavTranslationOptions` constants to the `howToTranslate` parameter to specify how files are to be translated by the function `NavTranslateFile` (page 62).

```
enum {
    kNavTranslateInPlace    = 0,
    kNavTranslateCopy       = 1
};
typedef UInt32  NavTranslationOptions;
```

Constant descriptions

`kNavTranslateInPlace`
Tells Navigation Services to replace the source file with the translation. This setting is the default for Save dialog boxes.

`kNavTranslateCopy` Tells Navigation Services to create a translated copy of the source file. This setting is the default for Open dialog boxes. The function `NavGetFile` (page 58) always uses this setting under automatic translation.

Result Codes for Navigation Services

All Navigation Services functions return result codes. The codes specific to Navigation Services are listed here. In addition, functions that communicate

with other Mac OS managers, such as the File Manager or the Translation Manager, may return result codes from those managers.

`paramErr`	-50	Your application passed an invalid parameter for dialog box options.
`userCancelledErr`	-128	User cancelled the action.
`kNavInvalidSystemConfigErr`	-5696	One or more Navigation Services–required system components is missing or out of date.
`kNavCustomControlMessageFailedErr`	-5697	Navigation Services did not accept a control message sent by your application.
`kNavInvalidCustomControlMessageErr`	-5698	Your application sent an invalid custom control message.
`kNavMissingKindStringErr`	-5699	No kind strings were provided to describe your application's native file types. For more information, see “Providing File Format Options” (page 33).

Glossary

action button A button that initiates the action associated with a dialog box; Save, Open, and Choose are common examples. The action button is often, but not always, the default button. See also **default button**.

alert box A dialog box that appears on screen to warn the user or to report an error.

Appearance Manager The part of Mac OS that manages all aspects of user interface appearance and themes, including controls, sounds, and color data.

banner In a dialog box, an application-defined static text field that provides a specific prompt to the user.

bevel button A control that resembles a square, beveled push button. It can take the behavior of other controls, such as radio buttons, checkboxes, push buttons, and pop-up menus.

browser list A list box that displays file objects for navigation and selection.

CFM-68K Runtime Enabler A Mac OS system extension that provides Code Fragment Manager services to non–PowerPC-based systems.

control An onscreen object that the user can manipulate by using the mouse or keyboard equivalents in order to cause instant action with visible results or change settings to modify a future action.

current location The folder or volume whose name appears as the title of the Location button and whose contents are being displayed in a dialog box's browser list.

customization rectangle The area in a dialog box that is available for application-defined items.

default button In an alert or dialog box, the button whose action is invoked if the user presses the Return key or the Enter key. The Appearance Manager identifies the default button by drawing a ring around it. The default button should invoke the preferred action which, whenever possible, should be a "safe" action – that is, one that doesn't cause loss of data.

default location The folder or volume whose contents are displayed in the browser list when a dialog box is first opened.

deferred translation The process of saving a file in native format and waiting to provide translation until the user closes the file.

desktop The onscreen background upon which all applications display their user interface. The desktop is composed of the objects in the startup volume's desktop folder plus the icons of all other mounted volumes.

dialog box A box that appears on the screen to solicit information from the user or to report that the computer is waiting for a process to complete.

disabled A menu item or control that cannot be chosen; the item may appear dimmed.

disclosure triangle A control that expands a view to disclose additional information about the item being viewed.

drag region The frame of a window, including the title bar and window outline, but excluding the close box, zoom box and size box. Dragging inside this region moves the window to a new location and makes it the active window (unless the user is holding down the Command key).

editable text field A rectangular box inside a dialog box in which the user enters text to provide information to an application.

file object A file, folder, or volume.

focus ring A border that highlights the currently active edit text field or scrolling list in a dialog box in order to indicate to user which item has keyboard focus. The focus ring appears in the current accent color. See also **keyboard focus**.

Help menu The menu that provides access to on-screen help information.

highlight To make something visually distinct, typically when it's selected. This is generally done by reversing black and white or changing colors to provide a sharp contrast.

invisible file A file that the Finder does not normally display to the user.

keyboard equivalent Keystrokes that invoke a corresponding menu command. A keyboard equivalent is usually a combination of one or more modifier keys and a character key.

keyboard focus A property that determines which control in a dialog box will receive all keystrokes. The user can change keyboard focus by using keyboard navigation or clicking. See also **focus ring**.

keyboard navigation In a dialog box, moving the keyboard focus by pressing the Tab key. See also **keyboard focus**.

kind string The string displayed in the Kind column in a Finder window's list view.

list box A control that combines a rectangular frame, one or two scroll bars, and a scrolling list.

modal dialog box A dialog box that puts the user in the state or "mode" of being able to work only in the dialog box. The user cannot move a modal dialog box; it can be dismissed only by clicking its buttons. Compare **modeless dialog box**, **movable modal dialog box**.

modeless dialog box A dialog box that resembles a document window without a collapse box. The user can move a modeless dialog box, make it inactive and active again, and close it like a document window. Compare **modal dialog box**, **movable modal dialog box**.

movable alert box An alert box with a title bar that allows the user to move the alert box.

movable modal dialog box A modal dialog box that has a title bar (with no close box) that allows the user to move the dialog box. Compare **modal dialog box**, **modeless dialog box**.

multiple selection Selecting more than one item in a scrolling list, usually by Shift-clicking or Command-clicking.

Navigation Services An application programming interface that allows your application to provide a user interface for navigating, opening, and saving Mac OS file objects.

native file type A file type that an application identifies as one it can open without requesting additional translation.

object filtering The process of determining whether a file object should be displayed to the user.

'open' resource A Translation Manager resource that declares which file types an application can open.

persistence The ability to recall the user-set properties of a dialog box after it has been closed.

pop-up menu A menu that appears somewhere other than the menu bar. A pop-up menu opens when the user presses the control that the menu is associated with, which is usually a pop-up menu button or a bevel button.

pop-up menu button A button that, when pressed, presents a pop-up menu. The button's label indicates the current menu setting.

preview area The area of an Open dialog box reserved for drawing previews of files selected in the browser list.

rebound The ability of a file browser to recall the location last viewed. See also **default location**.

scrolling list A list of user-selectable items that can be scrolled if it is longer than the available display area.

Shift-click To click while the Shift key is down. The Shift key modifies what a simple click does in a given situation – for example, when a user is selecting items in a list, Shift-clicking extends or shortens the selection.

size box A box in the lower-right corner of some active windows. Dragging the size box resizes the window.

sort key Data associated with a file that can be used to determine display order in a browser list. Sort keys commonly include filename, date, and kind. See also **sort order**.

sort order Determines whether files in a browser list will be displayed in ascending or descending order. See also **sort key**.

Standard File Package The part of the Mac OS system files that provides a way for users to identify files to open and save. Starting with Mac OS 8.5, Standard File Package is superseded by Navigation Services.

type pop-up menus The collective name for pop-up menu buttons appearing below the browser list in Navigation Services dialog boxes. Type pop-up menu buttons include the Show button in Open dialog boxes and the Format button in Save dialog boxes.

type selection The ability to select an item from a list of items by typing the first character or characters of the item's name.

volume A portion of a storage device that is formatted to contain files.

Index

K

L

M

N

O

P

Q

R

S

T

U

V

W

This Apple manual was written, edited, and composed on a desktop publishing system using Apple Macintosh computers and FrameMaker software. Line art was created using Adobe™ Illustrator and Adobe Photoshop.

Text type is Palatino® and display type is Helvetica®. Bullets are ITC Zapf Dingbats®. Some elements, such as program listings, are set in Adobe Letter Gothic.

WRITER
Otto Schlosser

DEVELOPMENTAL EDITORS
Donna S. Lee, Laurel Rezeau

ILLUSTRATOR
David Arrigoni

PRODUCTION EDITOR
Glen Frank

ACKNOWLEDGMENTS
Yan Arrouye, Andy Bachorski, Gail DeCamp, Sean Findley, Tony Francis, Gordon Garb, Pete Gontier, Tim Holmes, Susan McGarry, Keith Mortensen, Quinn "The Eskimo!"