Programming With Navigation Services in Mac OS 9

(Legacy)

Mac OS 9 & Earlier > Unsupported

2005-07-07

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

Simultaneously published in the United States and Canada.

Contents

Figures and Listings

Introduction to Programming With Navigation Services in Mac OS 9

Important: The information in this document is obsolete and should not be used for new development.

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 is available in Mac OS 8.6 and later.

Who Should Read This Document?

This document is for application developers who want to use Navigation Services for such tasks as opening and saving files and choosing file objects. Those who plan to implement custom features in Navigation Services dialogs will also find this document useful.

Organization of This Document

This document contains the following chapters:

- "Navigation Services Concepts" (page 9), provides an overview of Navigation Services, describes the user interface in Mac OS 8 and 9, lists the keyboard equivalents supported by Navigation Services, and discusses the persistence of user settings, such as the default directory.
- "Navigation Services Tasks" (page 17), shows how to perform basic and advanced programming tasks and how to create application-defined functions to implement custom view filtering and previews.

See Also

Navigation Services Reference in Carbon User Experience Documentation provides a complete reference for the Navigation Services application programming interface.

See Also

Navigation Services Concepts

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 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

Starting with Mac OS 8.5, Navigation Services is built into the Mac OS System file.

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.
- AppleGuide or Apple Help for online help.

The Standard File Package is supported in Mac OS 9 and earlier. You will obtain greater functionality and compatibility, however, by using 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 does 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:

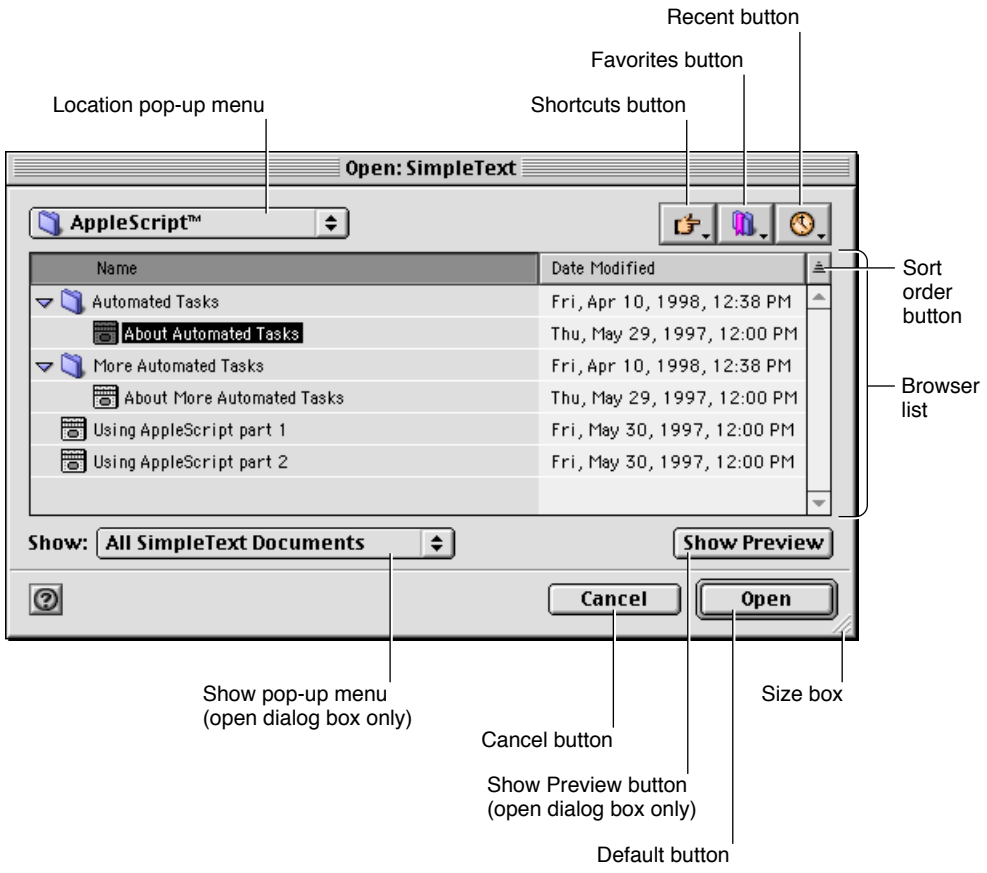
- Save Change
- Discard Changes

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
- a size box

Figure 2-1 shows these elements used in an Open dialog box.

Figure 1-1 Standard dialog box elements

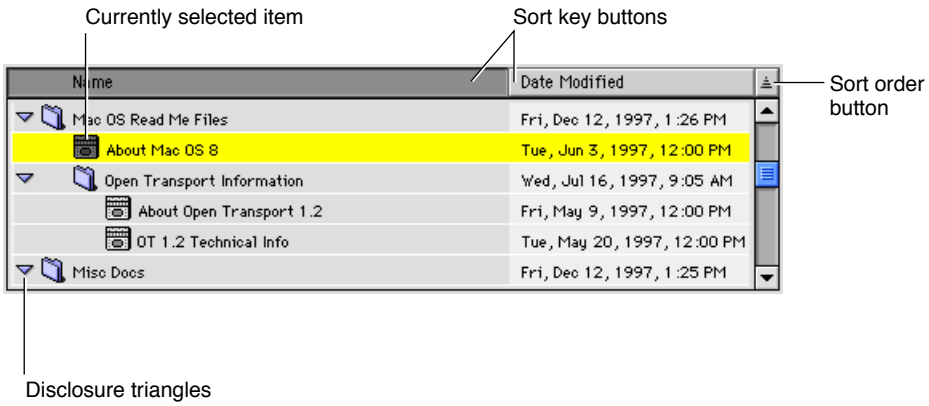


Note: Minor changes were made in the user interface elements of Navigation Services dialog boxes between version 1.0 and version 1.1. Figures in this document have been updated for Navigation Services version 1.1 if appropriate.

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 2-2 (page 12).

Figure 1-2 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:

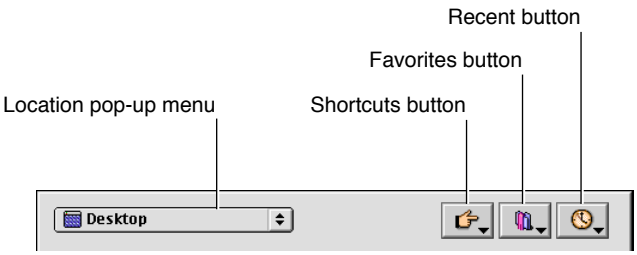
- The smallest size uses the format <MM/DD/YY> (for US systems); for example, 7/16/97.
- If more space is available, the format expands to <DayOfWeek, MMM DD, YY, Time>; for example, Wed, Jul 16, 1997, 9:05 AM.
- The widest format features fully spelled-out names; for example, Wednesday, July 16, 1997, 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 2-3, provide quick navigation tools.

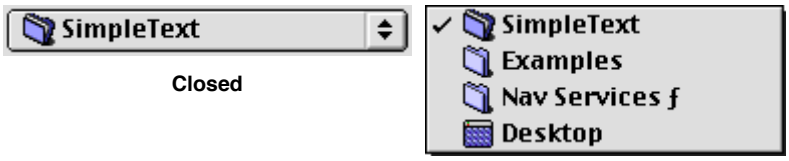
Figure 1-3 Navigation options



Location Button

The Location pop-up menu displays the current location and provides a familiar way to navigate the file system hierarchy. Figure 2-4 (page 13) shows the pop-up menu in its open and closed states.

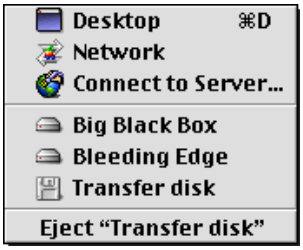
Figure 1-4 Location pop-up menu in closed and open states



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 2-5 (page 13).

Figure 1-5 Shortcuts pop-up menu



Navigation Services 1.1 and later has two network connection options in 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 and later 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 2-6 (page 14).

Figure 1-6 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.

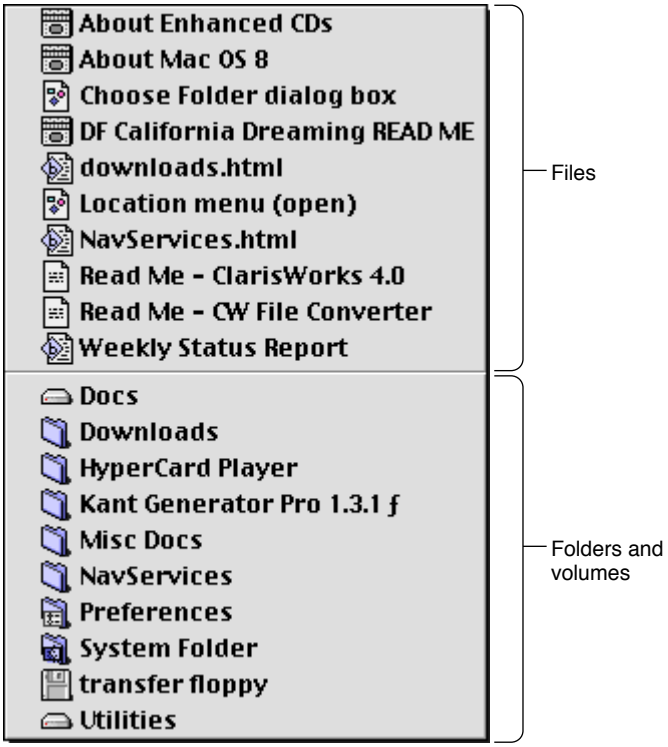
In an Open dialog box, Navigation Services displays favorite files and folders, but in a Save dialog box, only folders are displayed.

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 2-7 (page 15).

Figure 1-7 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.

Navigation Services Tasks

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.

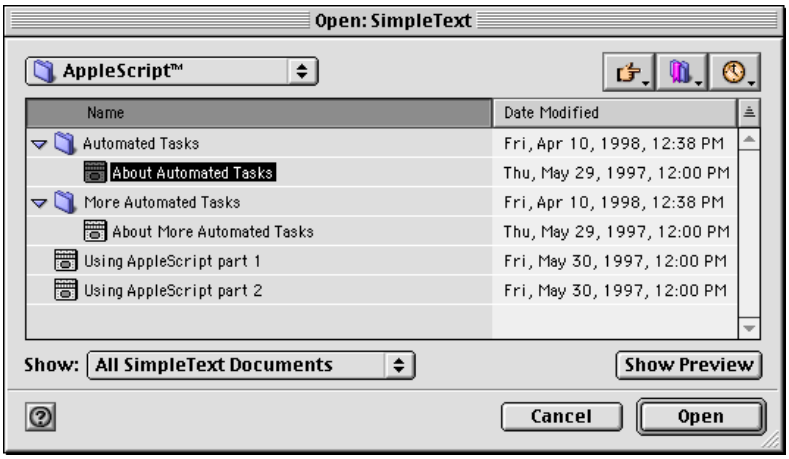
Basic Tasks

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` displays an Open dialog box that prompts the user to select one or more files to open, as shown in Figure 3-1.

Figure 2-1 Open dialog box



For a description of the elements of an Open dialog box, see "User Interface" (page 10).

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 37).

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` when you pass this structure in the `dialogOptions` parameter of a file-opening function such as `NavGetFile`.

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 3-2 shows an example of a Show pop-up menu with file translation options.

Figure 2-2 A Show pop-up menu with file translation option

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 uses the same format as an `'open'` resource and may be loaded from a resource or passed in as a `NavTypeList` structure created on the fly.

Note: Navigation Services does 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 `kNavNoTypePopup` 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 31).

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 3-2 (page 18). 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` 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` as needed.

You can obtain translation information from the structure of type `NavReplyRecord` 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 3-1 shows a sample function illustrating one way to call the function `NavGetFile`. 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 2-1 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 default location option
    dialogOptions.dialogOptionFlags |= kNavSelectDefaultLocation;
    //  Clear preview option
    dialogOptions.dialogOptionFlags ^= kNavAllowPreviews;

    // make descriptor for default location
    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;

        // Call NavGetFile() with specified options and
        // declare our app-defined functions and type list
        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);
            // Set up index for file list
            if (anErr == noErr)
            {
                longindex;

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

                    // Get a pointer to selected file
                    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;
}
```

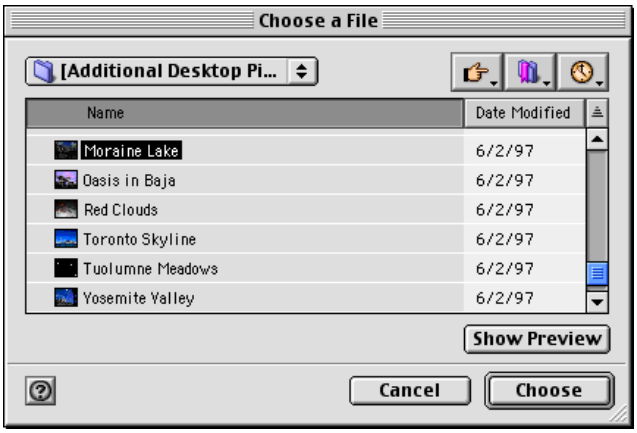
Choosing File Objects

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` 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 3-3shows an example of this dialog box.

Figure 2-3 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

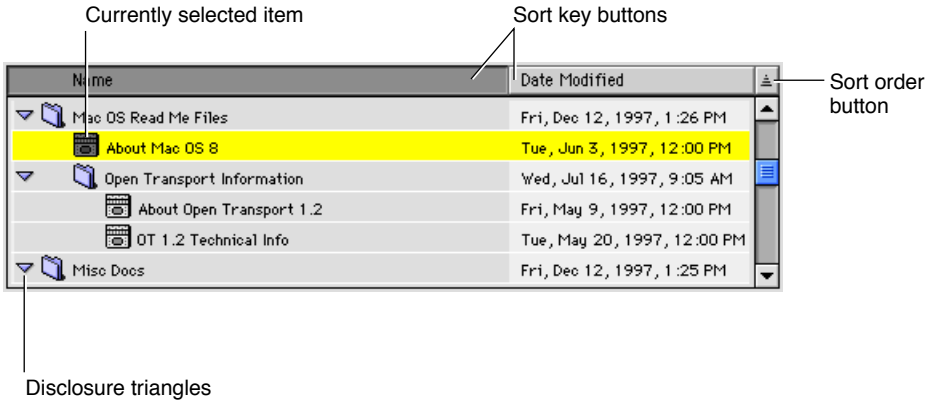
the file types displayed in the browser list. The `NavChooseFile` function ignores the `componentSignature` field of the `NavTypeList` structure, so all files of the types specified in the `typeList` parameter will be shown, regardless of their application signature. For more information on filtering options, see "Filtering File Objects" (page 31).

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` 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` displays a dialog box that prompts the user to choose a folder, as shown in Figure 3-4. This could be useful when performing installations, for example.

Figure 2-4 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.

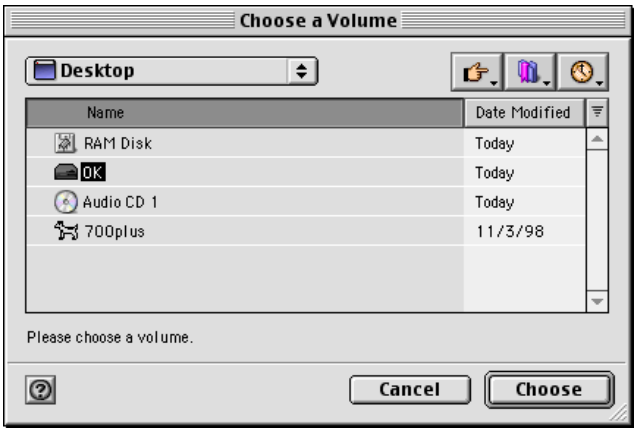
Note: If the user navigates to the desktop, deselects the current selection by shift-clicking and presses the Choose button, `NavChooseFolder` identifies the desktop as the folder chosen.

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` that you pass in the `dialogOptions` parameter of the `NavChooseFolder` function. This string is displayed below the browser list. In Figure 3-4, 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` displays a dialog box that prompts the user to choose a volume, as shown in Figure 3-5.

Figure 2-5 Choose a Volume dialog box



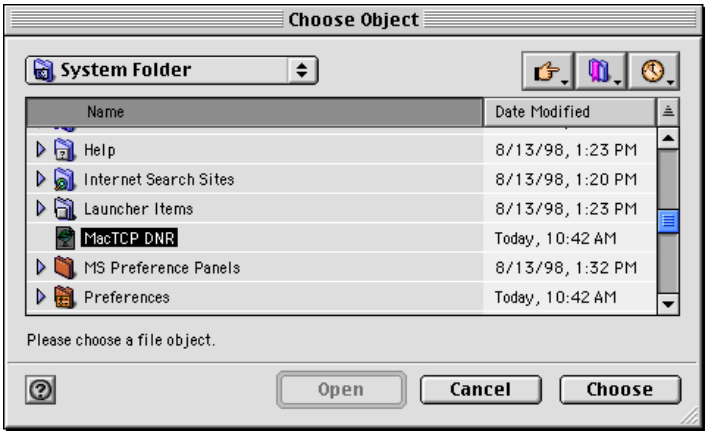
This function is useful when you want the user to select a volume or storage device.

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` 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 File System Objects

The function `NavChooseObject` displays a dialog box that prompts the user to select a file object, as shown in Figure 3-6.

Figure 2-6 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 `NavChooseFile`.

Note: The `NavChooseObject` function does not allow the user to choose zones, servers, or any other network object.

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` that you pass in the `dialogOptions` parameter of the `NavChooseObject` function. This string is displayed below the browser list. In <xRefNoLink>Figure 3-6, 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` displays a dialog box that prompts the user to create a new folder, as shown in Figure 3-7.

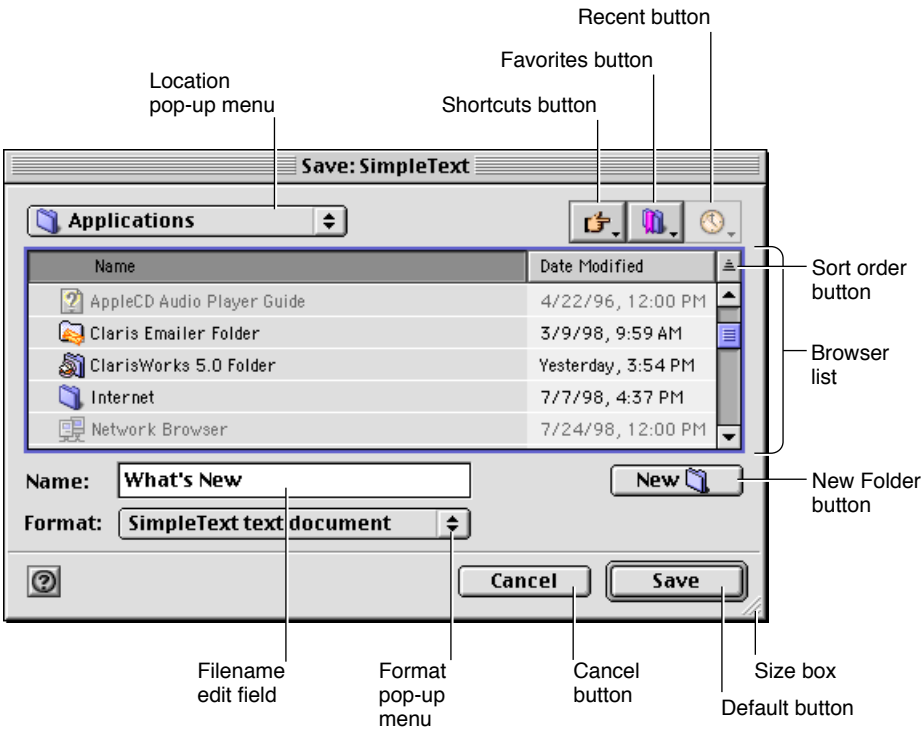
Figure 2-7 New Folder dialog box

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` 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` displays a Save dialog box, as shown in Figure 3-8.

Figure 2-8 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 37).

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` 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 `NavPutFile` function 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 3-9 shows an example of this menu.

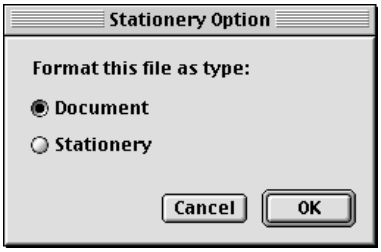
Figure 2-9 Format pop-up menu

Note: If you specify the `kNavNoTypePopup` constant in the dialogOptionFlags field of the structure `NavDialogOptions` 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 26).

The last item in the menu is the Stationery Option command. This displays the Stationery Option dialog box, shown in Figure 3-10, 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 2-10 Stationery Option dialog box



Note: If you clear the `kNavAllowStationery` constant in the `dialogOptionFlags` field of the structure `NavDialogOptions` 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`. 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`. 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` is intended to be used only while opening files. Always call the function `NavCompleteSave` 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`.

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, in the `howToTranslate` parameter of the `NavCompleteSave` function.

A Sample File-Saving Function

Listing 3-2 illustrates how to save files by using the function `NavPutFile`. 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 2-2 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,
                            fileTypeToSave, creatorType, nil);
        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

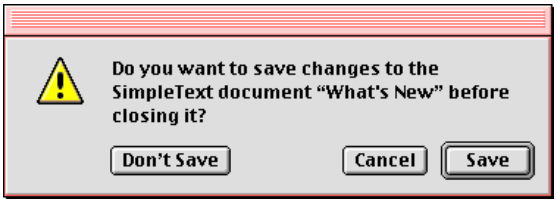
Navigation Services allows you to display a standard alert box for saving changes or quitting an application, and to customize this alert box for other uses.

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 37).

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`, which displays the alert box shown in Figure 3-11.

Figure 2-11 Standard Save Changes alert box

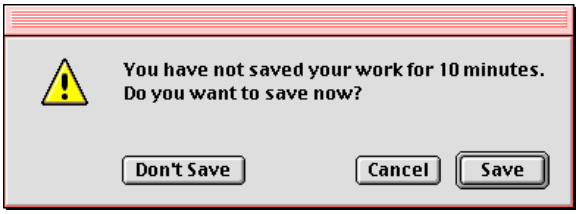


After the user closes the Save Changes alert box, Navigation Services tells your application which button the user clicked by returning one of the `NavAskSaveChangesResult` constants, as described in Save Changes Action Constants.

Customizing the Save Changes Alert Box

You can display a customized Save Changes alert box by using the function `NavCustomAskSaveChanges`. Figure 3-12shows an example of a customized Save Changes alert box.

Figure 2-12 Custom Save Changes alert box

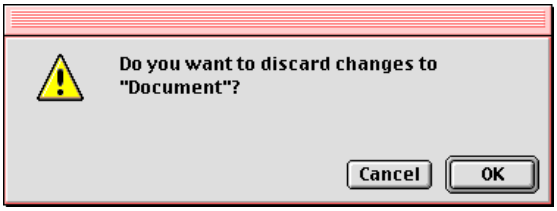


You must provide the message to be displayed in a custom Save Changes alert box by specifying a string in the message field of a `NavDialogOptions` structure.

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 3-13. This alert box is created by calling the function `NavAskDiscardChanges`.

Figure 2-13 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.

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`.

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 is normally of type `'typeFSS'` describing 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 you specify in the `dialogOptions` parameter of a Navigation Services function such as `NavGetFile`. 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 and later displays the desktop as the default location.

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 requires a particular type of `AEDesc` structure, 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, AppleShare servers, or (in Navigation Services 2.0 or later) other network services like FTP or HTTP. 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*.

Working With Packages

Mac OS 9 introduces the concept of packages. Packages are file objects that combine an application or document with supporting files in a container. By default, Navigation Services does not display packages in the browser list; if you want your application to recognize packages, specify the `kNavSupportPackages` constant in the `dialogOptionFlags` field of the `NavDialogOptions` structure that you pass in the `dialogOptions` parameter of various Navigation Services functions. If you want your application to be able to navigate packages, specify the `kNavAllowOpenPackages` constant in the same way. Package support is available in Navigation Services 2.0 and later.

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 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 by creating conflicts between your type list and filter function.

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 on implementing a filter function, see "Providing Custom Object Filtering" (page 38).

The function `NavGetFile` 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 18).

The function `NavPutFile` 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.

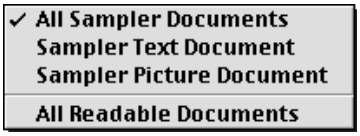
Object Filtering Scenarios

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` 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` that it passes in the `dialogOptions` parameter of the `NavGetFile` function.

The Show pop-up menu contains the items shown in Figure 3-14. Note that the menu does not contain a translatable file section.

Figure 2-14 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 does not 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 or later, if your application passes a `NavTypeList` structure to the `NavGetFile` function and specifies the `kNavNoTypePopup` constant, Navigation Services displays 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` 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 <xRefNoLink>Figure 3-15.

Figure 2-15 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 or later, 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 function such as `NavGetFile`, follow these steps if you are using a version of Navigation Services earlier than 2.0:

1. Supply the `kNavCtlGetLocation` constant in the `selector` parameter of the function `NavCustomControl` 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.

Navigation Services 2.0 and later provides the `kNavCtlBrowserRedraw` constant that you can specify in the the `selector` parameter of the function `NavCustomControl` to force the browser list to be refreshed.

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 automatically displays a preview of any file that contains a valid QuickTime component of type `'pnot'`. Under Navigation Services 2.0 or later, you can call the NavCreatePreview function to create a preview. You can request automatic preview display by setting the `kNavAllowPreviews` constant in the `dialogOptionFlags` field of the structure of type NavDialogOptions that you pass in the `dialogOptions` parameter of the file-opening function. (This option is set by default.) You can provide your own preview function and add custom controls to the preview area, as well. For more information on preview functions, see "Drawing Custom Previews" (page 39). 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` 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` 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 37).

You are not required 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` constant, described in Custom Control Setting Constants, 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` passed in the `callBackParms` parameter of your event-handling function. The `param` field of the `NavCBRec` structure points to a structure of type NavMenuItemSpec describing the menu item. Your application can respond to a particular menu item by comparing the `type` and `creator` fields, for example.

You can set the Show pop-up menu so that it displays only custom items during a call to a file-opening function such as `NavGetFile`, for instance. The procedure is as follows:

1. Define your custom menu items by using structures of type `NavMenuItemSpec`.
2. Specify a handle to the `NavMenuItemSpec` structures in the `popupExtension` field of the structure of type `NavDialogOptions` that you pass in the `dialogOptions` parameter.

3. 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.
4. Set your filter function to display only your custom items in the pop-up menu.
5. Ensure that your event-handling function takes care of any selection made from the pop-up menu.

If your application tries to extend the Show 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 31).

You can set the Format pop-up menu so that it displays only custom items during a call to the `NavPutFile` function. The procedure is as follows:

1. Define your custom menu items by using structures of type `NavMenuItemSpec`.
2. Specify a handle to the NavMenuItemSpec structures in the `popupExtension` field of the structure of type NavDialogOptions that you pass in the `dialogOptions` parameter.
3. Set your filter function to display only your custom items in the pop-up menu.
4. Ensure that your event-handling function takes care of any selection made from the pop-up menu.
5. Pass the `kNavGenericSignature` constant in the `fileCreator` parameter of the `NavPutFile` function.
6. If you don't want the Stationery Option menu item to appear, clear the `kNavAllowStationery` constant in the `dialogOptions` field of the `NavDialogOptions` structure you pass to the `NavPutFile` function.

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 37).
2. Respond to the `kNavCBCustomize` constant, described in Event Message Constants, which your application can obtain from the `param` field of the structure of type `NavCBRec` 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`. <xRefNoLink>Listing 3-3 illustrates one way to do this.

Listing 2-3 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, in the `selector` parameter of the function `NavCustomControl`. <xRefNoLink>Listing 3-4 illustrates this approach.

Listing 2-4 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.

Controlling User Navigation

In Navigation Services 2.0 and later, you can block certain user navigation actions such as file opening and saving by passing the `kNavCtlSetActionState` constant in the selector parameter of the function `NavCustomControl` and one or more of the values defined by the `NavActionState` enumeration (described in Action State Constants) in the `parms` parameter. This is useful when you want to prevent a dialog box from being dismissed until certain conditions are met, for example.

The actions you can block include:

1. Opening files
2. Saving files

3. Choosing file objects
4. Creating new folders

In the following example, the `kNavDontOpenState` constant is used to disable the Open button in an Open dialog box:

```
NavAction newState = kNavDontOpenState;
NavCustomControl( context, kNavCtlSetActionState, &newState );
```

In the next example, the `kNavDontChooseState` constant is used in addition to the `kNavDontOpenState` constant. During a call to the `NavChooseFolder` function, for example, this disables the Open and Choose buttons:

```
NavCustomControl( context, kNavCtlSetActionState, kNavDontOpenState +
        kNavDontChooseState);
```

In the final example, we add the `kNavDontNewFolderState` constant to disable the Open, Choose, and New Folder buttons during a call to the `NavNewFolder` function:

```
NavCustomControl( context, kNavCtlSetActionState, kNavDontOpenState +
        kNavDontChooseState + kNavDontNewFolderState );
```

If you block a user action by passing one or more of the `NavActionState` constants, make sure you set the `kNavNormalState` constant before exiting in order to restore the default state. If you fail to set the `kNavNormalState` constant, the user may be unable to dismiss the dialog box.

Creating Application-Defined Functions

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

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`. 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`. 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`. 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.

When calling your event-handling function, Navigation Services passes only the update events that apply to your windows and only the mouse-down events that 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. <xRefNoLink>Listing 3-5 shows an example of such a function.

Listing 2-5 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)
                    .eventDataParms).event->what)
            {
                case updateEvt:
                    MyHandleUpdateEvent(window,
                        (EventRecord*)callBackParms->eventData.event);
                    break;
            }
            break;
    }
}
```

In your event-handling function, you can also call the function `NavCustomControl` 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 34).

Providing Custom Object Filtering

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`, 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`. 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`. 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 *`theItem`* parameter is a '`typeFSS`' Apple event descriptor (`AEDesc`). If you intend to use the data passed in the `theItem` parameter, you must first determine its Apple event descriptor type by attempting to coerce the data into a type that you expect or support. If the `AEDesc` structure cannot be coerced into a valid file specification, for example, the data in the *`info`* parameter does not refer to an HFS file object. For more information on working with `AEDesc` structures, see *Inside Macintosh:Interapplication Communication*.

<xRefNoLink>Listing 3-6 illustrates a sample filter function that allows only text files to be displayed.

Listing 2-6 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 31).

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. 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. 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`.

Your preview-drawing function obtains information from a structure of type `NavCBRec` 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` 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`, 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`.

Document Revision History

This table describes the changes to *Programming With Navigation Services in Mac OS 9*.

Date	Notes
2005-07-07	Removed a redundant code listing.
2003-12-10	Corrected the parameter order in the function call `NavPutFile` in Listing 3-2 (page 27).
	Corrected the parameter order in the function call `NavCustomControl` in the section "Controlling User Navigation" (page 36).
	Removed old reference chapter from this document. The updated reference appears as a separate document—*Navigation Services Reference* in Carbon User Experience Documentation.
	Structured document and performed other production tasks to update the formatting.
	Changed the title from *Programming With Navigation Services* to *Programming With Navigation Services in Mac OS 9*.
1999-10-25	Updated for Navigation Services 2.0.
1998-11-20	Updated for Navigation Services 1.1.
1998-03-26	Released for Navigation Services 1.0 (SDK).

Document Revision History