The **Finder** is an application that works with the system software to keep track of files and manage the user's desktop display. This chapter describes the programming interface your application should use to interact with the Finder.

To use this chapter, you should be familiar with the Resource Manager. See the chapter "Introduction to the Macintosh Toolbox" in this book for general information about resources; detailed information about the Resource Manager and its routines is provided in the chapter "Resource Manager" in *Inside Macintosh: More Macintosh Toolbox*. Virtually all software intended for Macintosh computers must use the Finder-related resources described in this chapter.

Read this chapter to learn how to

■ set up the resources the Finder needs to display and start up your application

■ set up the resources the Finder uses to display information about other files related to your application

■ check or change Finder-related information stored in a volume's catalog file

■ support stationery pads

■ use the directories generally organized within the System Folder

This chapter does not explain how to use Apple events to communicate with the Finder. When a user opens or prints a file from the Finder, the Finder sends information to your application so that it can open or print the file. In System 7, applications that support high-level events receive this information through the required Apple events.

Refer to *Inside Macintosh: Interapplication Communication* for instructions on how your application should respond to these required Apple events that the Finder sends to your application: Open Application, Open Documents, Print Documents, and Quit Application. In addition, your application can use another set of Apple events—called Finder events—to request services from the Finder. For example, your application can ask the Finder to perform such operations as launching another application on your behalf. Refer to *Inside Macintosh: Interapplication Communication* for more details.

Introduction to the Finder Interface

The Finder is an application that manages the user's desktop interface. The **desktop** is the working environment displayed on the Macintosh computer—namely, the gray background area on the screen.

On the desktop, the Finder displays icons representing your application and the documents it creates, and it tracks user activity. An **icon** is an image that the Finder displays to graphically represent some object—such as a file, a folder, or the Trash— that the user can manipulate. For example, Figure 7-1 on the next page shows icons that the Finder displays for several sample applications (called SurfWriter 3.0, SurfPainter, and SurfDB) and for a text document (named Some Memo) that a user has created with the SurfWriter application. These icons are displayed in a window that the Finder uses to display the contents of the disk icon labeled Essentials.

Figure 7-1   Application and document icons in a window on the desktop
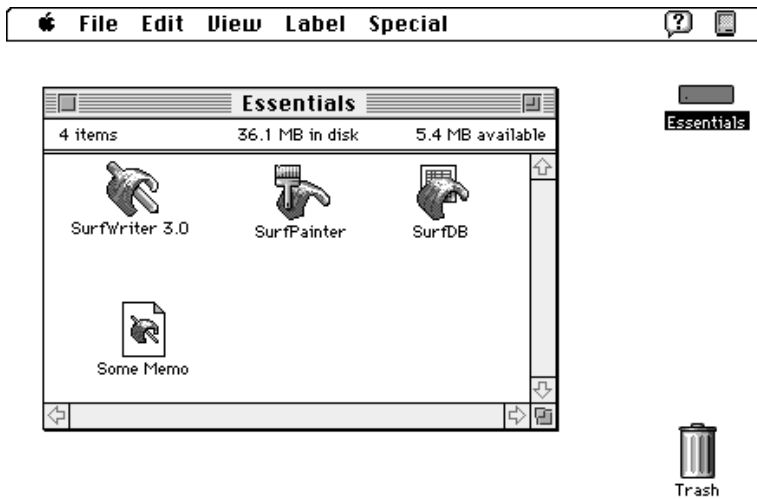
To distinguish your product for the user, you should design your own icons for all the files associated with your application. For each file type that your application uses or creates, you should define large, small, black-and-white, and 4-bit and 8-bit color icons—each in a separate resource. Your application can then use another resource, called a *bundle resource*, to assign these icons to all your files of a particular type. For example, the document icon representing Some Memo in Figure 7-1 is the icon that the SurfWriter application assigns to all text files that it creates. When double-clicking the icon for Some Memo, the user asks the Finder to launch the SurfWriter application, which in turn responds by opening the document Some Memo in a window.

Stationery pads are files that a user creates to serve as templates for other documents. **Editions** are special files that contain data to be shared among applications. **Query documents** contain commands and data in a format appropriate for a database or other data source. If your application supports any of these document types, you can create icons for the Finder that distinguish the stationery pads, editions, and query documents that users create with your application. For example, Plate 4 at the front of this book shows customized stationery pad and edition icons used for documents created with the SurfWriter application. (Editions are described in *Inside Macintosh: Interapplication Communication*. Query documents are described in *Inside Macintosh: Communications*.)

You might also like your application to create customized icons for documents on the desktop. Or, if instead of producing an application, you produce and distribute information documents (such as database files, stationery pads, query documents, clip art libraries, or dictionaries) to be used by other applications, you can also provide customized icons for the Finder that distinguish your documents.

Macintosh users have access to online assistance in the form of help balloons. You can customize the help balloon that the Finder displays for your application icon. For example, Figure 7-2 shows a customized help balloon for the SurfWriter application icon.

Figure 7-2   A customized help balloon for an application icon
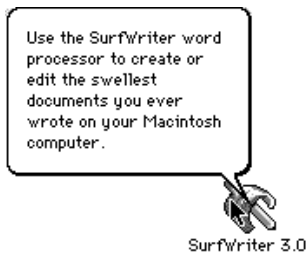
When appropriate, the Finder starts up your application and uses Apple events to tell your application what documents to open or print. To perform these tasks, the Finder relies on information you provide through resources. When the user creates or installs a file, the File Manager (described in *Inside Macintosh: Files*) initially stores some of this information in the volume's catalog file. (The **catalog file** is a special file, located on a volume, that contains information about the hierarchical organization of files and folders on that volume.)

The Finder extracts from the catalog file the information you provide in your resources and, for quick access to your resource information, the Finder uses that information to build either a desktop database for all volumes over 2 MB or a Desktop file for volumes under 2 MB. (The **desktop database** is a Finder-maintained database of icons, file types, applications, version data, and comments; the **Desktop file** is a resource file in which the Finder stores this information for volumes under 2 MB.)

You can even specify resources that identify your application when the user tries to open a document and your application is missing. For example, if a user tries to open a document named Instructions and the SurfWriter application is missing from the user's computer system, the Finder displays the alert box in Figure 7-3.

Figure 7-3   A Finder message identifying a missing application
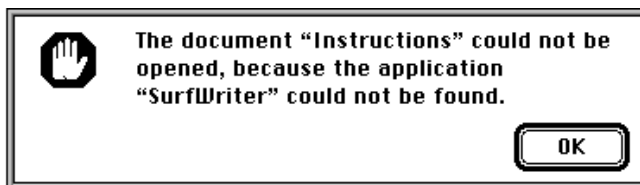
The **System Folder** is a directory that contains the software that Macintosh computers use to start up. The System Folder includes a set of folders for storing related files. Your application may use several of these folders for storing its files. For example, you may want to use the Preferences folder to store preferences files that your application needs when starting up.

About the Finder Interface

You can use the Finder interface to

■ create the resources—such as those describing icons—that the Finder uses to extract and to display information about your application and its documents (Generally, all applications should provide these resources for their files.)

■ determine and change the Finder information structure stored in a catalog file (Generally, most applications need to determine—and many might wish to set— information in the catalog file.)

■ support stationery pads so that users can easily use templates for their documents (Generally, most applications that create documents should support stationery pads.)

■ locate the directories typically located in the System Folder (Generally, many applications will want to access these directories.)

Using the Finder Interface

The Finder needs quick access to some key information about your application, such as what icons to use when displaying your application and its documents. You supply most of this information in the resource fork of your application file.

The Finder extracts this information and uses it to maintain its own database of the resources it needs. The Finder records the location of your application on disk in this database so that it can find your application quickly when the user opens one of your documents.

For compatibility with the Finder, your application should have

■ a signature resource, so that the Finder can identify and start up your application when a user double-clicks documents created by your application

■ a set of resources that describe icons that visually represent your application and any documents it creates

■ a set of file reference resources, to link icons with the file types they represent and to allow users to launch your application by dragging document icons to your application icon

■ a bundle resource, to group together your application's signature, icon, and file reference resources

■ a size resource, to tell the Finder how much memory to allocate for your application when it starts up and whether your application supports various system software features

■ either a missing-application name string resource in your application's documents (to display the name of your application if the user tries to open or print a document

created by your application when your application is missing) or an application-missing message string resource in your application's documents (to explain why the user can't open or print a document used only by your application)

Note
Supply a missing-application name string resource for documents that you intend for users to open with your application; supply an application-missing message string resource for documents (such as preferences files) that your application uses but that users shouldn't open. You supply only one of these resources in a document—never both. ◆

Your application can also make use of these resources:

■ version resources, so that users can easily find out the version of your application and, if applicable, the version of your application's superset of files

■ a help resource, which the Finder uses to display your customized Balloon Help message for your application, control panel, system extension, or desk accessory icon

If you sell or distribute data in the form of a document to be used by other applications, you can assist users by providing

■ an appropriate file type to allow users to open your document from the Finder by dragging its icon to an application icon or by choosing the Open command from the File menu within an application

■ the resources describing an icon family to represent your document to the user

■ a missing-application name string resource or an application-missing message string resource, so the Finder can assist users who try to open or print your documents from the Finder

■ version resources, so that users can easily find out the version of your document and, if your document file is one of a larger collection of files, the version of the entire superset of files

A catalog file exists on every volume to maintain relationships between the files and directories on that volume. (A volume is any storage medium formatted to contain files.) Although it's used mostly by the File Manager, the catalog file also contains information used by the Finder. You can always check the information in the catalog file. In particular, you may want to check the file type or creator for a file, or you may want to check or set one of the Finder flags for a document. When opening a document, your application should check a Finder flag to determine if the document is a stationery pad, and, if it is, your application should copy the document's contents into a new document and open the new document in an untitled window.

Your application might wish to use the folders located in the System Folder. Those you're most likely to want to access are Preferences, Temporary Items, and Trash. For example, you might wish to check for the existence of a user's configuration file in Preferences, create a temporary file in Temporary Items, or—if your application runs out of storage when trying to save a file—check how much storage is taken by items in the Trash directory and report this to the user. You can use the `FindFolder` function to get the path information you need to gain access to these system-related directories.

In System 7, users can create Finder objects called *aliases* to aid them in organizing their files. Ordinarily, when the user wants to open or print files, your application does not need to be concerned with whether they are aliases because the Finder resolves aliases before passing them to your application. However, if your application bypasses the Finder (or the Standard File Package, which is described in *Inside Macintosh: Files*) when manipulating documents, it should check for and resolve aliases itself by using the Alias Manager function `ResolveAliasFile`.

The rest of this chapter describes in detail how to use these Finder features in your application.

Giving a Signature to Your Application and a Creator and a File Type to Your Documents

The Finder identifies your application through its **signature,** a unique four-character sequence. The signature must not conflict with the signature of any other application. To ensure uniqueness, you must register your application's signature with Apple Computer, Inc., at Macintosh Developer Technical Support.

Note

There is no need to register your own resource types because they're usually used in only your own applications or documents. ◆

You must include in your resource file a special resource that has your application's signature as its resource type. By convention, the signature resource has a resource ID number of 0. The signature resource typically contains a string that specifies the name, version number, and release date of your application. If you do not provide specific version information through a version resource (described in "Providing Version Resources" beginning on page 7-31), the Finder displays the string stored in the signature resource when the user selects your application and chooses Get Info from the File menu.

Listing 7-1 illustrates a signature resource in Rez input format. (Rez is the resource compiler provided with Apple's Macintosh Programmer's Workshop [MPW], available from APDA.)

Listing 7-1   Rez input for a signature resource

```
type 'WAVE' as 'STR ';                /*WAVE is the signature*/
resource 'WAVE' (0, purgeable) {      /*resource ID is 0*/
   "SurfWriter 3.0 © 1992"            /*default Get Info string*/
};
```

Note

The signature resource alone is not sufficient to establish your application's signature. You must also supply a bundle resource, described in "Creating a Bundle Resource" beginning on page 7-20. ◆

Whenever your application creates a document, it assigns the document a creator and a file type. Typically, as described in "Using Finder Information in the Catalog File" beginning on page 7-32, your application sets its signature as the document's creator. When a user double-clicks a document or selects it and chooses Open or Print from the Finder's File menu, the Finder reads the creator field of that file to find the document's creator. The Finder then searches for an application with a signature by that name. When it finds that application, the Finder launches it.

If the document's creator is your application's signature, for example, the Finder calls the Process Manager to start your application. The Finder then passes to your application the information it needs to open or print the document; since the introduction of System 7, the Finder has used Apple events to pass this information to your application. *Inside Macintosh: Interapplication Communication* describes how your application processes the required Apple events to open or print files.

As described in "Using Finder Information in the Catalog File" beginning on page 7-32, your application typically assigns a file type to a document when it creates one. The file type can be a type especially defined for your application, or it can be one of the existing general types, such as those listed here.

File type	Description
'APPL'	Launchable application
'DFIL'	File for storing desk accessories
'DRVR'	Driver
'FFIL'	File for storing fonts
'INIT'	System extension
'PICT'	QuickDraw picture
'PRER'	Printer driver
'RDEV'	Chooser extension
'TEXT'	Stream of ASCII characters
'adev'	Network extension (such as EtherTalk 2.0)
'appe'	Background-only application
'cdev'	Control panel
'edtp'	Edition for sharing graphics-oriented data
'edts'	Edition for sharing sound-oriented data
'edtt'	Edition for sharing text-oriented data
'ffil'	Font
'ifil'	Script system resource collection
'kfil'	Keyboard layout
'pref'	Preferences file
'qery'	Query document for database access
'scri'	System extension for script systems
'sfil'	Sound

File type	Description
'tfil'	TrueType font
'ttro'	TeachText read-only file
'zsys'	A system file (such as the System file itself)

Note

Apple reserves the use of all signatures and file types whose names contain only lowercase and nonalphabetic characters. Your signature and the file types created especially for your application must each contain at least one uppercase character. Since the system software never displays signatures and file types to users, signatures and file types can consist of character combinations that might otherwise be incomprehensible to anyone but you. ◆

Like signatures, file types must be registered with Apple. Your application must have a file type of 'APPL'. The creator field of your application file should contain its own signature. Most programming environments provide a simple tool for setting the creator field of your application file.

Your application can create documents of any type, and it can specify any application as the creator. You could write a utility application, for example, that creates a new document by opening one text file and appending onto it another text file. The application would give the new document the same creator as the first original text file so that the Finder can call on that application when the user wants to open or print the new document.

Assign the standard file type 'TEXT' to files that consist of only text—that is, a stream of characters with return characters at the ends of paragraphs. Most word processors allow the user to create text-only files. A document of file type 'TEXT' can be opened or printed by any application that accepts such file types. Your application can still assign its own signature as the file's creator so that the Finder can call on it to open or print the file when appropriate.

Users can also open a document created by your application—as well as a document of a file type supported by your application—by selecting its icon and dragging it to your application's icon. Because the document's file type is stored in the catalog file and the Finder stores a list of your application's supported file types in the desktop database, the Finder can determine whether to launch your application. If the document's file type is supported by your application, the Finder launches your application and passes it the name of the document. (These topics are detailed in subsequent sections of this chapter.)

For example, if your application is a page-layout program, it might create documents of its own file type while also supporting documents of 'TEXT' and 'PICT' file types. A user can launch your application by dragging a document of any of these file types to your application icon.

Your application also relies on file types to determine which files to let the user open when your application is running. When your application calls the Standard File Package to open a file, your application supplies either a list of the file types that your application can open or a filter function for those types. The open file dialog box then displays only files of the specified types. (See *Inside Macintosh: Files* for details.)

Creating Icons for the Finder

The Finder represents your files as icons. To distinguish your product for the user, you can design your own icons for all the files associated with your application, including

- your application file itself

- standard documents created by your application

- stationery pads that users create from your application's documents

- data-sharing editions that users create from your application's documents

- other special documents, such as read-only, graphics, and query documents, which are either created by your Macintosh application or provided by you for use by other Macintosh applications

For most effective display, you should create an icon family for each of your files. An **icon family** is the set of icons that represent a single object, such as an application or a document, that the Finder displays. An entire icon family consists of large (32-by-32 pixel) and small (16-by-16 pixel) icons, each with a mask, and each available in three different versions of color: black and white, 4 bits of color data per pixel, and 8 bits of color data per pixel. Specifically, the following icons make up the icon family for a single file:

- a large (32-by-32 pixel) black-and-white icon and mask—both of which you define in an icon list (`'ICN#'`) resource

- a small (16-by-16 pixel) black-and-white icon and mask—both of which you define in a small icon list (`'ics#'`) resource

- a large (32-by-32 pixel) color icon with 4 bits of color data per pixel—which you define in a large 4-bit color icon (`'icl4'`) resource

- a small (16-by-16 pixel) color icon with 4 bits of color data per pixel—which you define in a small 4-bit color icon (`'ics4'`) resource

- a large (32-by-32 pixel) color icon with 8 bits of color data per pixel—which you define in a large 8-bit color icon (`'icl8'`) resource

- a small (16-by-16 pixel) color icon with 8 bits of color data per pixel—which you define in a small 8-bit color icon (`'ics8'`) resource

Plate 3 in the front of this book shows how the SurfWriter sample application uses these resources to define the icon family for its application icon.

Somewhat related to these resources are the icon (`'ICON'`) resource and the color icon (`'cicn'`) resource. You can use either to describe a 32-by-32 pixel icon within some element of your application. However, the Finder does *not* use or display any resources that you create of type `'ICON'` or type `'cicn'`. Instead, your application uses these resources to display icons within your application. Generally, you use an icon resource to display a black-and-white icon in a menu or dialog box, as described in the chapters "Menu Manager" and "Dialog Manager" in this book. (For example, the color alert box in Plate 2 in the front of this book specifies a resource of type `'cicn'` for the color icon in the upper-left corner of the alert box.) If you provide a color icon (`'cicn'`) resource with the same resource ID as the icon (`'ICON'`) resource, the Menu Manager and the Dialog Manager display the color icons instead of the black-and-white icons for users with color monitors.
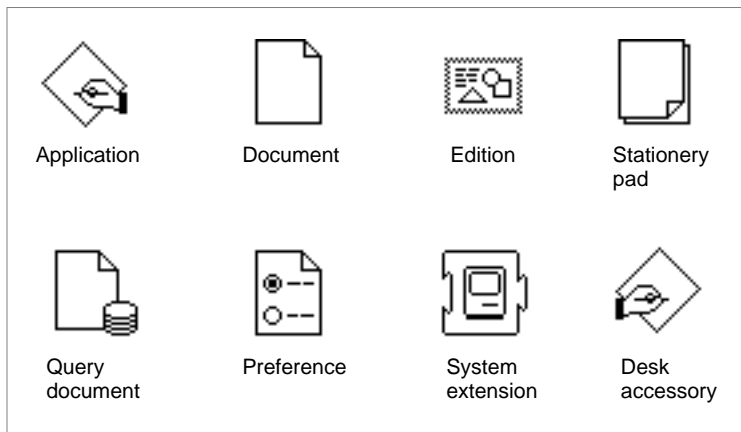
Before creating icon families for your files, you should begin by designing a graphic element that all of your icon families can share and that can help the users quickly identify the files associated with your product. Figure 7-4, for example, illustrates how a company uses the image of a wave in all of its application icons; these icons represent the SurfWriter text-editing application, the SurfPainter graphics application, and the SurfDB database application. As illustrated in Plate 4 at the front of this book, the wave element is also included in icons representing the documents, stationery pads, and editions that users create with these applications.

Figure 7-4   Large black-and-white application icons for a company's product line

Text-editing application   Graphics application   Database application
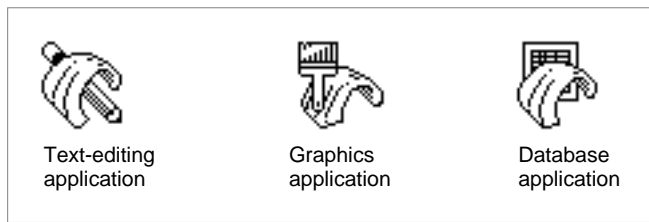
If you do not design your own icons, the Finder uses a set of its own default application and document icons for display. Figure 7-5 shows the Finder's default large black-and-white icons.

Figure 7-5   Default large black-and-white icons

Application   Document   Edition   Stationery pad

Query document   Preference   System extension   Desk accessory

Note

Desk accessories, displayed by default with the icon shown in
Figure 7-5, were designed for early versions of Macintosh system
software that did not support cooperative multitasking. Desk
accessories and applications are much more alike in their appearance
and behavior in System 7. Because there are no longer any compelling
reasons for creating desk accessories, you should generally write a
small application instead of a desk accessory if you wish to create
a small or simple program. ◆

If you don't want the Finder to display the default icons for your application or
documents, you must at least define an icon list (`'ICN#'`) resource for each icon.

The term *icon list* has become a bit of a misnomer, because you can define only two
images in the icon list resource: a 32-by-32 pixel black-and-white icon and its mask.
To define color and 16-by-16 pixel icons for a file, you create additional resources, as
described later in this section. (If you don't define color versions of your icons, the
Finder displays the black-and-white icon defined in your icon list resource on all
displays, and if you don't define 16-by-16 pixel icons, the Finder algorithmically reduces
the 32-by-32 pixel icon to half size when needed.)

An icon list resource defines one icon. It contains two icon descriptions: the actual icon
for display and an all-black mask that shows the area covered by the icon. The Finder
uses the mask to crop the icon's outline into whatever background color or pattern is on
the desktop. The Finder then draws the icon into this shape. Therefore, it's important
that the mask be exactly the same shape as the icon. The mask also defines the area that
users need to click to select the icon. Therefore, it's best not to have any holes in the
mask; otherwise, users may have trouble selecting your icon.

Figure 7-6 illustrates a black-and-white icon and its mask for an application. The area
around the pencil just underneath the wave creates a problem with this sample icon and
its mask: like a hole in a mask, it creates two small areas within the middle of the icon
that the user cannot select with the cursor.

Figure 7-6   A black-and-white icon and its mask for an application

An icon list resource is defined to be an array of two items of type `String[128]`; each bit in the first array represents a pixel in the 32-by-32 pixel icon, and each bit in the second array represents a pixel in the 32-by-32 pixel mask. Typically, you use a high-level tool such as the ResEdit application, which is available through APDA, to create your icon list resources. Figure 7-7 shows how the icon list resource for the icon in Figure 7-6 was created using the ResEdit icon editor. When you are satisfied with the appearance of your icons, you can use the DeRez decompiler to convert your icon list resources into Rez input.

Listing 7-2 is a partial listing of the icon list resource's Rez input that describes the application icon shown in Figure 7-7; Listing 7-2 also shows partial listings for the icon list resources used for the icons that represent the documents created by the application. This listing and those that follow in this chapter use Rez input format to help you understand the format of the resources and see how they work together.

Figure 7-7   The ResEdit view of an icon
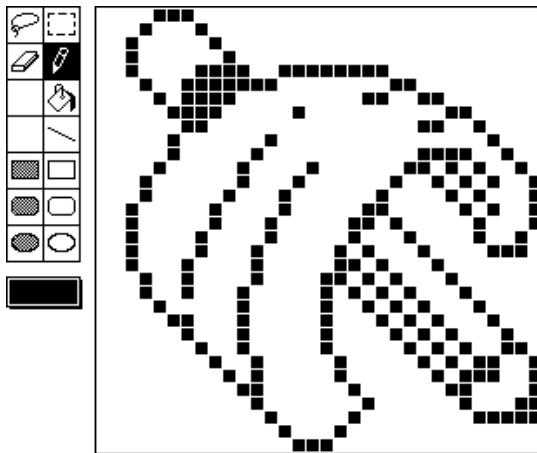
Listing 7-2   Rez input for an icon list resource

```
data 'ICN#' (128, purgeable) {        /*application icon & mask*/
                        /*array: 2 elements*/
                        /*[1]: the application icon*/
    $"0E 00 00 00"      /*1st line of icon: 4 bytes (32 bits)*/
    .                   /*32 lines total in icon*/
    .
    .
    ,                   /*[2]: the mask*/
    $"0E 00 00 00       /*1st line of mask: 4 bytes (32 bits)*/
    .                   /*32 lines total in mask*/
    .
    .
};
```

```
data 'ICN#' (129, purgeable) {    /*text document icon and mask*/
                      /*icon data goes here*/
};
data 'ICN#' (130, purgeable) {     /*stationery pad icon & mask*/
                      /*icon data goes here*/
};
data 'ICN#' (131, purgeable) {     /*edition icon & mask*/
                      /*icon data goes here*/
};
```

You can also define a small (16-by-16 pixel) version of your icon in a small icon list resource (that is, in a resource of resource type 'ics#'). On black-and-white monitors, the Finder displays the small icon in windows when the user chooses by Small Icon from the View menu. On black-and-white monitors, the small icon also appears in the Application menu after the user launches your application and in the Apple menu if the user places your application or an alias to it in the Apple Menu Items folder. (Alias files and the Apple Menu Items folder are described, respectively, in "Using Aliases" beginning on page 7-39 and "Using the System Folder and Its Related Directories" beginning on page 7-41.)

You should also define color versions of both large and small icons by using several resource types. The resource for each icon variation has the same resource ID as the icon list resource that defines the large black-and-white icon. For example, if the resource ID number of your application icon's icon list resource is 128, its small icon list resource should have a resource ID number of 128; and the following resources should also have resource IDs of 128: the large 4-bit color icon resource, the small 4-bit color icon resource, the large 8-bit color icon resource, and the small 8-bit color icon resource.

Don't define masks for the resources that define color icons. The large 4-bit color icon resource and large 8-bit color icon resource use the black-and-white icon mask defined in their companion icon list resource, and the small 4-bit color icon resource and small 8-bit color icon resource use the black-and-white icon mask defined in their companion 'ics#' resource. Because of this, the outline shapes of your color icons should exactly match those defined in your 'ICN#' and 'ics#' resources.

ResEdit 2.1 includes an icon family editor to help you easily manage the creation of these related resources. See the *ResEdit Reference* for details.

See *Macintosh Human Interface Guidelines* for information about the most effective use of color and shape for your icons. It is generally best that you first create the black-and-white icons in the icon list resource and small icon list resource and then add color to them using the resources that define color icons. Don't alter the shapes of your icons among these resources; otherwise, the masks defined in the icon list resource and the small icon list resource won't match these shapes. Choose your colors from the 36 recommended icon colors in the system palette. (If you use ResEdit 2.1, these colors appear in a palette when you choose Apple Icon Colors from the Color menu.) Note that you cannot specify your own color table for these resources.

For more information about color palettes, see *Inside Macintosh: Imaging.* Although the Palette Manager allows you to define a palette for the system to use when it needs to define the color environment, you should rely on the system palette colors for your

icons. Users may often use the Finder when your application is not running, and the user can switch to another application when your application is running. Relying on the system palette gives your icons a more consistent look in the Finder regardless of what the active application is. Also, because users can change the desktop color and pattern, your application gives users more control over their work environment if your icons rely on the system palette. Users can always alter your color definitions by selecting an icon and choosing a color from the Label menu. The Finder then blends the chosen color into those of the selected icon. To restore the original colors, users must choose None from the Label menu.

If your application creates documents, it should also define at least two additional icon families: one to be displayed for documents created by your application and another to be displayed when the user creates a stationery pad from one of your applica-tion's documents. ("Supporting Stationery Pads" beginning on page 7-34 describes stationery pads.)

If your application creates other variations of its documents, you can assist your users by providing different icons for the different documents. For example, TeachText has separate icon families to distinguish its read-only and graphics documents.

If your application supports data sharing through the Edition Manager, your application should also define an icon family for editions. The Edition Manager (described in *Inside Macintosh: Interapplication Communication*) allows users to share and automatically update data from numerous documents and applications. For example, a user might want to capture sales figures and totals from within a spreadsheet and then include this information in a word-processing document that summarizes sales for a given month. If both the spreadsheet and word-processing applications support the Edition Manager, the user begins by selecting data within the spreadsheet document and creating a publisher. The spreadsheet application then writes a copy of that data to a separate file, called an *edition*. The edition is represented by an icon; by default, it appears as the edition icon shown in Figure 7-5 on page 7-12. If the user opens a word-processing document and creates a subscriber to the spreadsheet document's edition, the word-processing application then incorporates the desired sales figures and totals from the spreadsheet document's edition into the document.

If you design your application to create editions, consider creating an icon that uniquely identifies your editions and that associates them with your application's documents. The file type for your edition containers should be `'edtt'` (for text-oriented data), `'edtp'` (for graphics-oriented data), or `'edts'` (for sound-oriented data); and the creator, of course, should be the signature of your application.

If your Macintosh application is a database program or serves as a source for data (as a spreadsheet program often does), you might wish to create query documents so that other Macintosh applications can gain access to that data through the Data Access Manager; in this case, your application should also define an icon family for its query documents. (See *Inside Macintosh: Communications* for information on sharing data in this manner.)

Plate 4 at the front of this book shows the large color icons for the various documents that the sample SurfWriter application creates: text documents, stationery pads, and editions.

Defining icon resources is not enough to display your icons. In addition, you must follow one of two sets of procedures:

■ If you are an application developer, you must define file reference resources and a bundle resource for your application, as described in "Creating File Reference Resources" beginning on page 7-18 and "Creating a Bundle Resource" beginning on page 7-20.

■ If you are an information provider or a database developer—that is, if you provide documents that are used by other applications—you don't need to create file reference resources or a bundle resource to provide document icons on Macintosh computers running System 7. You can instead create customized icons for your documents as described in the following section.

Creating Customized Document Icons

You can create customized icons for your documents. Users can also create customized icons. When an icon list resource is stored with a resource ID of –16455 in the resource fork of a file, the Finder uses the large, small, 4-bit and 8-bit color, and black-and-white icons defined in resources with that resource ID as **customized icons** in place of the Finder's default icon and in place of any icons listed in the file's bundle resource.

Note
Although an application can assign icons to it all of its documents by associating their icons with the documents' file types in a bundle resource (as explained in "Creating File Reference Resources" beginning on page 7-18 and "Creating a Bundle Resource" beginning on page 7-20), a customized icon can represent only one specific file—that file that has an icon list resource with a resource ID of –16455 in its resource fork. ◆

Users of System 7 are able to customize individual icons. By selecting a file and choosing Get Info from the File menu, the user sees the information window for that file. The user can then select the icon displayed in the upper-left corner of the information window and use the Paste command in the Edit menu to replace it with a picture from the Clipboard. The Finder creates a family of icons based on the user's customized icon, assigns a resource ID number of –16455 to each resource in the icon family, stores these resources in the resource fork of the file that the icon represents, and sets the hasCustomIcon bit in the file's Finder flags field. (Finder flags are described in detail in "File Information Record" beginning on page 7-47.)

Your application can use the same strategy to provide customized icons for the documents that it creates. For example, a drawing application might create miniature versions of the illustrations contained within its documents and use those for the documents' icons.

If you are a database developer who creates and distributes query documents that support the Data Access Manager, you can also use this strategy to create icons that identify your database's query documents. Similarly, if instead of producing an application you produce and distribute information (such as database files, stationery pads, clip art libraries, or dictionaries) to be used by other applications, you might want to provide icons that distinguish your documents.

To make the Finder display customized icons for a document, you must create—at least—an icon list resource with resource ID –16455 and store it in the document's resource fork. (To create this while your application is running, your application can call the `AddResource` procedure, described in the chapter "Resource Manager" in *Inside Macintosh: More Macintosh Toolbox.*) You can use the following constant in place of the ID number:

```
CONST kCustomIconResource = –16455; {res ID for custom icon}
```

If you provide only an icon list resource, the Finder uses a black-and-white icon on all screen displays and automatically reduces it when a small version of the icon is required. To create color versions and to define a small version of the icon, create an entire icon family as described in "Creating Icons for the Finder" beginning on page 7-11.

After creating resources for icons using the `kCustomIconResource` constant as their IDs, you must set the `hasCustomIcon` bit in the file's Finder flags field. To prevent users from changing these icons, set the `nameLocked` bit in the file's Finder flags field. (Most development environments provide tools for setting these bits. "Using Finder Information in the Catalog File" beginning on page 7-32 describes how to determine and set these Finder flags.)

Creating File Reference Resources

File reference (`'FREF'`) resources perform two main functions. First, they associate icons you define with file types used by your application. Second, they allow users to drag document icons to your application icon in order to open them from your application.

Create a file reference resource for your application file itself and create separate file reference resources for each file type that your application can open. Listing 7-3 shows, in Rez input format, the file reference resources for the SurfWriter application file, text documents, stationery pads, and editions and for TeachText read-only documents.

Each file reference resource specifies the following items:

■ a file type
■ the local ID of an icon list resource as assigned in the bundle resource
■ an empty string

The file type can be defined for files created by your application only, for files created by other applications that your application supports, or for files of the existing general types, such as `'TEXT'` and `'PICT'`.

As described in the next section, "Creating a Bundle Resource," the local ID maps the file type to an icon list resource that is assigned the same local ID in the bundle resource. If you wanted two file types to share the same icon, for example, you could create two separate file reference resources that share the same local ID, which the bundle resource would map to the same icon list resource. (Creating two file types that share the same icon is not recommended, however, because a shared icon would make it very difficult for the user to distinguish between the different file types while using the Finder.)

Listing 7-3   Rez input for file reference resources

```
resource 'FREF' (208, purgeable) {  /*SurfWriter application*/
   'APPL',  /*type 'APPL'*/
   0,       /*maps to icon list resource w/ local ID 0 in bundle resource*/
   ""       /*leave empty string for name: not implemented*/
};
resource 'FREF' (209, purgeable) {  /*SurfWriter document*/
   'TEXT',  /*type 'TEXT'*/
    1,      /*maps to icon list resource w/ local ID 1 in bundle resource*/
   ""
};
resource 'FREF' (210, purgeable) {  /*SurfWriter stationery pad*/
   'sEXT',  /*type 'sEXT'*/
   2,       /*maps to icon list resource w/ local ID 2 in bundle resource*/
   ""
};
resource 'FREF' (211, purgeable) {  /*SurfWriter edition*/
   'edtt',  /*type 'edtt'*/
   3,       /*maps to icon list resource w/ local ID 3 in bundle resource*/
   ""
};
resource 'FREF' (212, purgeable) {/*TeachText read-only files*/
   'ttro', 4, ""  /*These documents have TeachText as their */
                  /* creator. Finder uses TeachText's icon list resource */
                  /* for these documents. Included here so users */
                  /* can drag these docs to SurfWriter's app icon*/
};
```

If you provide your own icon for the stationery pads that users create from your application's documents, create a file reference resource for your stationery pads. Assign this file reference resource a file type in the following manner: use the file type of the document upon which the stationery pad is based, but replace the first letter of the original document's file type with a lowercase *s*. As with other file reference resources, you map this to an icon list resource in the bundle resource. (This convention necessitates that you make the names of your documents' file types unique in their last three letters.)

For example, in Listing 7-3, the 'sEXT' file type assigned within the file reference resource is used for stationery pads created from documents of the 'TEXT' file type. In this case, when the isStationery bit (described in "Using Finder Information in the Catalog File" beginning on page 7-32) is set on a document of file type 'TEXT', the Finder looks in the SurfWriter application's bundle ('BNDL') resource to determine what icon is mapped to documents of type 'sEXT'. The Finder then displays the document using the stationery pad icon shown in Plate 4 at the front of this book.

When the user drags a document icon to your application icon, the Finder checks a list that it maintains of your file reference resources. If the document's file type appears in this list, the Finder launches your application with a request to open that document.

If your application supports file types for which it doesn't provide icons, you can still define file reference resources for them, and then users can launch your application by dragging these document icons to your application icon. For example, the file reference resource with resource ID 212 in Listing 7-3 on page 7-19 is created so that the Finder launches the SurfWriter application when users drag TeachText read-only documents to the SurfWriter application icon. Since these documents have TeachText as their creator, the Finder displays the icon that the TeachText application defines for them in its own bundle resource.

By supporting the Open Documents event, you can also specify disks, folders, and a pair of wildcard file types in your file reference resources so that users can launch your application by dragging their icons to your application icon. As explained in *Inside Macintosh: Interapplication Communication,* the Open Documents event is one of the four required Apple events. After the Finder uses the Process Manager to launch an application that supports high-level events, the Finder sends your application an Open Documents event, which includes a list of alias records for objects that the application should open.

Because alias records can specify volumes and directories as well as files, an Open Documents event gives you the opportunity to handle cases in which users drag disk or folder icons to your application. (Alias records are described in "Using Aliases" beginning on page 7-39.) Create a file reference resource and specify `'disk'` as the file type to allow users to drag hard disk and floppy disk icons to your application icon. Create a file reference resource and specify `'fold'` as the file type to allow users to drag folder icons to your application icon.

You can create a file reference resource that specifies `'****'` as the file type to allow users to drag all file types—including applications, system extensions, documents, and so on, but not including disks or folders—to your application icon. If you create three file reference resources that specify `'disk'`, `'fold'`, and `'****'` as their file types and if your application supports the Open Documents event, you effectively allow users to launch your application by dragging any icon to your application icon. It is up to your application to open disks, folders, or all possible file types in a manner appropriate to the needs of the user.

Creating a Bundle Resource

A bundle (`'BNDL'`) resource associates all of the resources used by the Finder for your application; in particular, it associates your application and its documents with their icons. The bundle resource contains

- the application's signature
- the resource ID number of its signature resource (which should always be 0)
- the assignment of local IDs to the resource IDs of all icon list resources defined for the application; the local IDs must be the same as those assigned within corresponding file reference resources

■ the assignment, for compatibility reasons, of local IDs to file reference resource IDs (For consistency, these can be the same local IDs that are assigned inside the file reference resources, but they don't have to be—they only need to be unique for every file reference resource.)

When the Finder first displays your application on the user's desktop, it checks the catalog file (described in detail in "Using Finder Information in the Catalog File" beginning on page 7-32) to see if your application has a bundle resource. If it doesn't, the Finder displays the default icons shown in Figure 7-5 on page 7-12. If your application has a bundle resource, the Finder installs the information from the bundle resource and all its bundled resources into either the desktop database for a hard disk or into the Desktop file for a floppy disk and uses this information to display icons for the file types associated with your application.

You must assign local IDs to your icon list resources within your bundle resource. Make sure that for all your file types with icons, these local IDs match the local IDs you assigned inside their corresponding file reference resources. In the Desktop file on floppy disks (and on hard disks running earlier versions of system software), the Finder renumbers the resource IDs that you've assigned to your resources to avoid conflicts with the resources of other applications. Therefore, the bundle resource has to rely on these local IDs to map icon list resources to their file reference resources; that is, the bundle resource uses the local ID you assign to an icon list resource to map it to the file reference resource that has specified the same local ID.

For example, the file reference resource with resource ID 208 in Listing 7-3 on page 7-19 shows that the file type `'APPL'` (the SurfWriter application file) is assigned a local ID of 0. In the bundle resource shown in Listing 7-4, you see that local ID 0 is assigned to the icon list resource with resource ID 128. This maps the icon defined by this resource (see Figure 7-7 on page 7-14) to the SurfWriter application file. Listing 7-4 shows the bundle resource for the icons and file reference resources defined in Listing 7-2 on page 7-14 and in Listing 7-3 on page 7-19.

Listing 7-4   Rez input for a bundle resource

```
resource 'BNDL' (128, purgeable) {  /*SurfWriter bundle resource*/
   'WAVE',     /*SurfWriter signature*/
   0,          /*resource ID of signature resource: should be 0*/
   {
      'ICN#', {   /*mapping local IDs in 'FREF's to 'ICN#' IDs*/
      0, 128,  /*'FREF' w/ local ID 0 maps to 'ICN#' res ID 128*/
      1, 129,  /*'FREF' w/ local ID 1 maps to 'ICN#' res ID 129*/
      2, 130,  /*'FREF' w/ local ID 2 maps to 'ICN#' res ID 130*/
      3, 131   /*'FREF' w/ local ID 3 maps to 'ICN#' res ID 131*/
               /*no 'FREF' with local ID 4 in this list: */
               /* TeachText's icons used for 'ttro' file type*/
      },
```

```
     'FREF', {    /*local res IDs for 'FREF's: no duplicates*/
     10, 208, /*local ID 10 assigned to 'FREF' res ID 208*/
     11, 209, /*local ID 11 assigned to 'FREF' res ID 209*/
     12, 210, /*local ID 12 assigned to 'FREF' res ID 210*/
     13, 211, /*local ID 13 assigned to 'FREF' res ID 211*/
     14, 212  /*local ID 14 assigned to 'FREF' res ID 212*/
     }
   }
};
```

In Listing 7-4, notice that you also assign local IDs to file reference resources inside the bundle resource. This assignment is superfluous because the Finder doesn't map these local IDs to any other resources. The local ID assignment for file reference resources inside the bundle resource was implemented for the earliest versions of Macintosh system software, and it remains this way today to maintain backward compatibility. For compatibility with the format of the bundle resource, assign local IDs to file reference resource IDs. You may number them any way you like, except that each local ID in this particular list must be unique.

Of all the icon resource types that make up an icon family, you need to list only the icon list resource in the bundle resource. The Finder automatically recognizes and loads all the other members of the icon family—provided that you have given them the same resource IDs that you have assigned to your icon list resource.
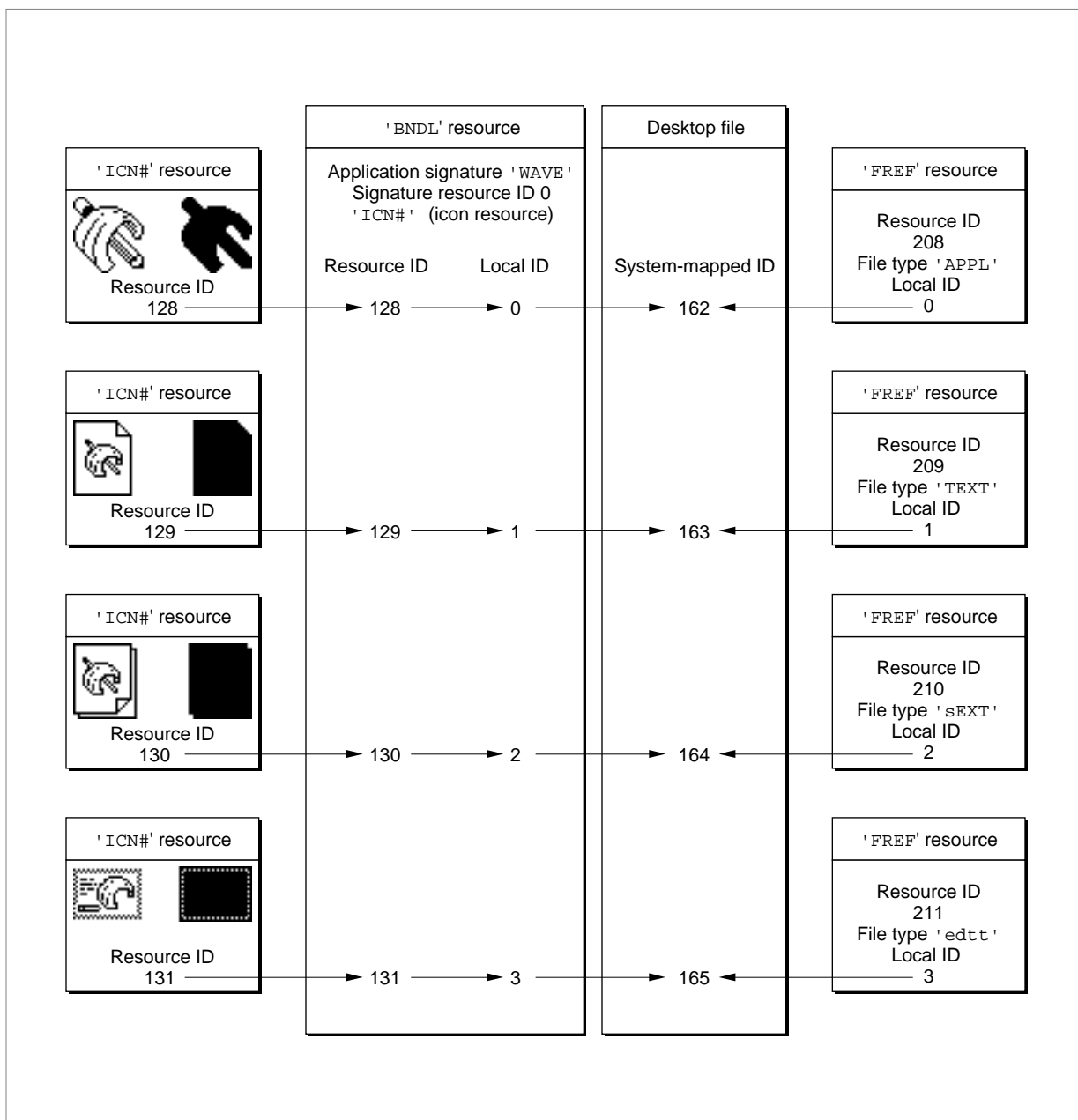
If the user drags documents created by other applications to your application icon, and if you have created file reference resources for these documents' file types, the Finder launches your application and passes it the names of the documents. You should create file reference resources for all file types that your application supports. Do not provide icon resources for file types created by other applications because the Finder won't use them, but will instead use the icon resources defined by the documents' creators. Though the local IDs of such a file reference resource are superfluous in the file reference resource and at the bottom of the bundle resource, the resource formats require that you provide local IDs in both.

For example, notice in Listing 7-3 on page 7-19 that the file reference resource with resource ID 212 is assigned a local ID of 4, but that no icon list resource is assigned to local ID 4 in the bundle resource in Listing 7-4 on page 7-21. This file reference resource, which specifies a file type of 'ttro', was created in Listing 7-3 to make the Finder launch the SurfWriter application when users drag TeachText read-only documents to the SurfWriter application icon. No icon mapping is made for this file type in the SurfWriter application's bundle resource because the Finder displays the icons defined for it by the TeachText application. The file reference resource with resource ID 212 is assigned to local ID 14 in the bundle resource in Listing 7-4 because the format of the resource requires a local ID for all associated file reference resources.

You alert the Finder that your application has a bundle resource by setting a bit in the file's Finder flags field. (Most development environments provide a simple tool for setting the bundle bit. "Using Finder Information in the Catalog File" beginning on page 7-32 describes Finder flags.)

Figure 7-8 illustrates how the bundle resource created in Listing 7-4 uses local IDs to map icon list resources to file reference resources. This figure illustrates two main concepts: first, that one bundle resource ties together all the icon resources and file reference resources for your application and all of its documents; and second, that the icon resources and their associated file reference resources are mapped together by local IDs.

Figure 7-8   Linking icon list resources and file reference resources in a bundle resource

In Figure 7-8, the application file's icon list resource has resource ID 128 while its file reference resource has resource ID 208. For easier code maintenance, you should probably assign the same resource ID to a file's file reference resource that you assign to its icon list resource. However, because the Finder renumbers these whenever it adds them to a Desktop file on floppy disks, you must map them by using local IDs. In Figure 7-8, the application file's icon list resource is assigned local ID 0. This maps the icon to the file type described by the file reference resource with local ID 0—in this case, the file reference resource with resource ID 208.

The general steps you must take to provide icons for applications and documents are enumerated here and assume that you are using a tool, such as ResEdit, that allows you to open and edit several resources simultaneously. (Remember that these resources must have resource IDs of 128 or greater.)

To provide your application with icon families for itself and for its documents, follow these steps:

1. Design a graphic element that all of your icon families can share in common and that can help users quickly identify the files associated with your product.

2. Create an icon list (`'ICN#'`) resource for your application file.

3. Create the other members of the icon family of the application file—resources of types `'ics#'`, `'icl8'`, `'icl4'`, `'ics8'`, and `'ics4'`—and give each of these the same resource ID as the icon list resource.

4. Create a bundle (`'BNDL'`) resource.

5. Within the bundle resource, list the resource ID number of the application file's icon list resource and assign it a local ID of 0.

6. Create a file reference resource for the application file.

7. Within the file reference resource, assign the application a file type of `'APPL'` and assign it a local ID of 0.

8. Within the bundle resource, list the resource ID number of the file reference resource for the application file and assign it a unique local ID—for example, 0 to maintain consistency with the local ID assigned in the file reference resource.

9. Create another icon family—consisting of resources of types `'ICN#'`, `'ics#'`, `'icl8'`, `'icl4'`, `'ics8'`, and `'ics4'`—to represent one type of document that your application creates.

10. Within the application's bundle resource, list the resource ID number of the document's icon list resource and assign it a local ID of 1.

11. Create a file reference resource for the document.

12. Within the file reference resource for the document, assign it a file type (for example, `'TEXT'` or `'edtt'`) and assign it a local ID of 1.

13. Within the bundle resource, list the resource ID number of the file reference resource for the document and assign it a unique local ID—for example, 1 to maintain consistency with the local ID assigned in the file reference resource.

14. Assigning unique local IDs for every type of document your application creates, repeat steps 9 through 13.

15. If your application supports file types of other applications, define file reference resources for them, but do not create icon resources for them.

16. Create a signature resource (as described in "Giving a Signature to Your Application and a Creator and a File Type to Your Documents" beginning on page 7-8) with resource ID 0.

17. Set the file's `hasBundle` bit and clear the `hasBeenInited` bit in the file's Finder flags. (Finder flags are described in "Using Finder Information in the Catalog File" beginning on page 7-32.)

18. Save and close all of the resources. (When you restart your Macintosh computer, your application should appear with its own icon. If you later alter any of your icons, clear the `hasBeenInited` bit and rebuild your desktop database by pressing Command-Option when restarting.)

How and When the Finder Launches Your Application

The previous sections in this chapter explain the resources that the Finder uses to display and launch your application. This section provides a brief summary of how the Finder—using the previously described resources—starts up your application whenever the user requests the Finder to launch your application or to open or print a document supported by your application.

The simplest scenarios under which the Finder launches your application occur when the user double-clicks your application icon or selects it and chooses Open from the Finder's File menu. In these cases, the Finder calls the Process Manager to start your application. As explained in *Inside Macintosh: Processes*, the Process Manager creates a partition of memory for your application, loads your code into this partition, and sets up the stack, heap, and A5 world for your application. The Process Manager returns control to the Finder.

If your application supports the required Apple events (as explained in *Inside Macintosh: Interapplication Communication*), the Finder sends your application an Open Application event and then relinquishes control to your application. Your application then performs the tasks necessary to open itself—displaying an untitled document window, for example.

When the user requests the Finder to open or print a document supported by your application, the Finder calls the Process Manager and launches your application in the same way, except that the Finder sets up the information your application needs to open or print the document and passes this information to your application. This information includes a list of files to open or print. In System 7, applications receive this information through Apple events, which are described in *Inside Macintosh: Interapplication Communication*.

The user can request the Finder to open documents created by your application by double-clicking one of their icons, and the user can request the Finder to open or print documents by selecting one or more icons and choosing Open or Print from the Finder's File menu. The Finder reads the creator field of each selected file to find the document's creator. Typically (as described in "Using Finder Information in the Catalog File" beginning on page 7-32), your application sets the four-character string specified in its

signature resource as the creator of its documents. The Finder searches for the application whose signature matches each document's creator. If the document's creator matches your application's signature, the Finder calls the Process Manager, launches your application, and then passes your application the name of the selected document or selected multiple documents in an Open Documents or a Print Documents event. Your application should then open the documents in titled windows or print them, as appropriate. (See *Inside Macintosh: Files* for detailed information about opening documents; see *Inside Macintosh: Imaging* for detailed information about printing them.)

If the user tries to open documents created by your application and your application is missing, the Finder displays an alert box telling the user that your application is missing. The Finder displays the name of your application in this alert box if you provide your documents with a missing-application name string resource, as described in "Displaying Messages When the Finder Can't Find Your Application" beginning on page 7-27.

Sometimes when your application is already running, the user might double-click a document created by your application. In this case, the Finder sends your application the Open Documents event.

The user can also request the Finder to launch your application by dragging one icon or several icons to your application's icon. The Finder determines whether to launch your application by comparing the document's file type (which is stored in the catalog file) against the list of your application's supported file types. The Finder compiles this list from the file reference resources you create for your application; the Finder stores this list in the desktop database. If the document's file type appears in the file reference resource list for your application, the Finder calls the Process Manager, launches your application, and passes it the name of the selected document or selected multiple documents in an Open Documents event. Your application should then open the documents in titled windows.

You can also specify disks, folders, and a wildcard file type for all other files in your file reference resources so that users can launch your application by dragging their icons to your application icon, in which case the Finder launches your application and sends it an Open Documents event. An Open Documents event includes a list of alias records for objects that the application should open. It is up to your application to open disks, folders, or all possible file types in a manner appropriate to the needs of the user. (Alias records are described in "Using Aliases" beginning on page 7-39.)

To support stationery, your application should specify the `isStationeryAware` constant in its `'SIZE'` resource and always check the `isStationery` bit of a document passed to it by the Finder. If the `isStationery` bit is set for a file that the user wants to open, your application should copy the stationery pad's contents into a new document and open the document in an untitled window. This is described in "Supporting Stationery Pads" beginning on page 7-34.

In System 7, users can create aliases, which are objects that represent other files, directories, or volumes. If the user opens an alias that represents a document created by your application, the Finder resolves the alias for you; that is, it passes your application the name and location of the document itself, not the alias.
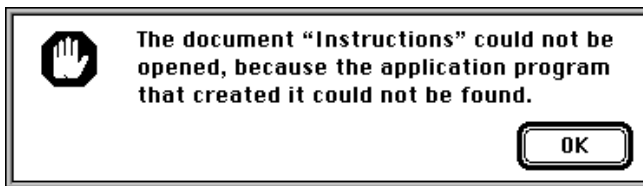
Displaying Messages When the Finder Can't Find Your Application

When the user double-clicks a file or selects it and chooses either the Open or the Print command from the Finder's File menu, the Finder looks for the application whose signature is stored in the file's creator field. The Finder starts up that application and tells it which documents the user wants to open or print. If the Finder cannot find the creator application, it displays an alert box.

If the document is of file type `'TEXT'` or `'PICT'` and if the TeachText application is available, an alert box asks the user whether the TeachText application should be used to open the document. For documents of any other file type, or if the TeachText application is not present, the Finder displays an alert box like the one shown in Figure 7-9. Your application should store one of two string resources in its documents to make the alert box message more useful than the default shown in Figure 7-9.

Figure 7-9   The default application-unavailable alert box

Before displaying the default message shown in Figure 7-9, the Finder looks in the document for one of two special `'STR '` resources with resource ID numbers of –16396 and –16397: the missing-application name string and the application-missing message string, respectively. If the Finder can't find the document's creator on any mounted volume, it looks first for the application-missing message string resource. Provide an application-missing message string resource if you do not intend for users to open the file. The message should explain why the file can't be opened. If the Finder does not find an application-missing message string resource, it looks for the missing-application name string resource. Provide a missing-application name string resource if you intend for users to open the file. The missing-application name string should be your application's name; the Finder displays it in an alert box to inform the user that your application is needed.

Supply either the application-missing message string resource or the missing-application name string resource; don't supply both. Supply an application-missing message string resource for documents (such as a preferences file) that your application uses but that users should not open; supply a missing-application name string resource for documents that you intend for users to open with your application.

Your missing-application name string resource (an `'STR '` resource with a resource ID number of –16396) should contain the name of your application. Listing 7-5 on the next page shows a missing-application name string resource for the SurfWriter application.

Listing 7-5   Rez input for a missing-application name string resource

```
resource 'STR ' (-16396, purgeable) {  /*the application name*/
   "SurfWriter"
};
```

You can store this resource in the resource fork of your application. When your application saves a document for the first time, it should copy the missing-application name string resource from your application's resource fork to the resource fork of the newly created document. Listing 7-6 shows a fragment of an application-defined function called DoSaveAsCmd, which the application calls when the user chooses the Save As command from the File menu. (For a description of the File Manager routines used here to create, open, and save the resource file, see *Inside Macintosh: Files*.)

Listing 7-6   Storing a missing-application name string resource in the resource fork of
                    a document

```
VAR
   myData:  MyDocRecHnd;    {handle to document record}
   myErr:   OSErr;
   myFile:  Integer;        {file reference number}

{with the DoSaveAsCmd routine: create document's resource fork}
FSpCreateResFile(myData^^.fileFSSpec, 'MYAP', 'TEXT',
                 smSystemScript);
myErr := ResError;
IF myErr = noErr THEN    {open the resource fork}
   myFile := FSpOpenResFile(myData^^.fileFSSpec, fsRdWrPerm);
IF myFile > 0 THEN   {copy the missing-application name string}
   myErr := DoCopyResource('STR ', -16396, gAppsResFile, myFile)
ELSE
   myErr := ResError;
IF myErr = noErr THEN
   myErr := FSClose(myFile);  {close the resource fork}
```

Listing 7-7 shows the application-defined function DoCopyResource, which copies the missing-application name string resource from the application's resource fork into the newly created document's resource fork. (For a description of the Resource Manager routines used here to set, open, and write the resource file, see the chapter "Resource Manager" in *Inside Macintosh: More Macintosh Toolbox*.)

Listing 7-7   Copying the missing-application name string resource into the resource fork of a document

```
FUNCTION DoCopyResource (theType: ResType; theID: Integer;
                         source: Integer; dest: Integer): OSErr;
VAR
   myHandle:   Handle;  {handle to resource to copy}
   myName:     Str255;  {name of resource to copy}
   myType:     ResType; {ignored; used for GetResInfo}
   myID:       Integer; {ignored; used for GetResInfo}

BEGIN
   UseResFile(source);  {set the source resource file}
   myHandle := GetResource(theType, theID);  {open the source}
   IF myHandle <> NIL THEN
   BEGIN
      GetResInfo(myHandle, myID, myType, myName);  {get resource }
                                                   { name}
      DetachResource(myHandle);  {detach resource}
      UseResFile(dest);     {set the destination resource file}
      AddResource(myHandle, theType, theID, myName);
      IF ResError = noErr THEN
         WriteResource(myHandle);   {write resource data}
   END;
   DoCopyResource := ResError;       {return result code}
END;
```
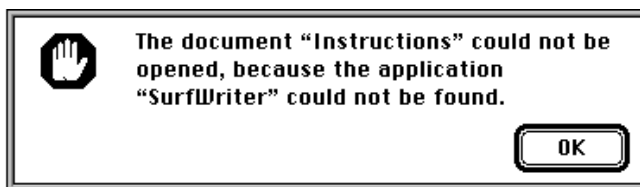
If a user tries to open or print one of the application's documents when the application is not present, the Finder specifies the application's name in the alert box, as illustrated in Figure 7-10.

Figure 7-10   The application-unavailable alert box specifying an application's name

Your application-missing message string resource (an 'STR ' resource with a resource ID number of –16397) should explain why the user cannot open or print a document. Use this resource for files—such as your application's preferences file— that are not intended to be opened or printed by the user. Register a signature (as explained in "Giving a Signature to Your Application and a Creator and a File Type to Your Documents" beginning on page 7-8) that is different from the signature of your

application and set this signature as the creator of files that you don't want your users to open. This ensures that the Finder displays your message instead of launching your application when the user double-clicks these documents.

Listing 7-8 illustrates an application-missing string resource that explains why a user cannot open a preferences file.

Listing 7-8   Rez input for an application-missing message string resource

```
resource 'STR ' (-16397, purgeable) {/*the message*/
   "This document describes user preferences for the application "
"SurfWriter. You cannot open or print this document. To be "
"effective, this document must be stored in the Preferences "
"folder in the System Folder."
};
```

Figure 7-11 shows the alert box generated by Listing 7-8.

Figure 7-11   The application-unavailable alert box with a customized message
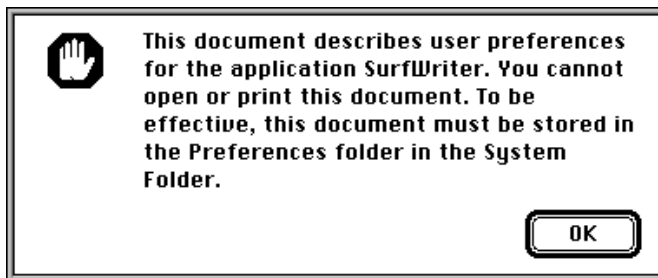
Note that if your application creates documents of file type `'TEXT'` or `'PICT'`, if the TeachText application is available, and if your application is missing when the user tries to open these documents from the Finder, the Finder always displays the alert box shown in Figure 7-12. For these file types, the Finder displays this alert box even if you provide missing-application name string resource or application-missing message string resource.

Figure 7-12   The application-unavailable alert box for `'TEXT'` and `'PICT'` documents
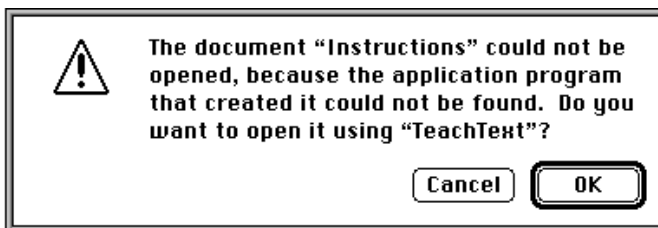
Providing Version Resources

You can use version (`'vers'`) resources to record version information for your application. If the user opens the Views control panel, clicks the Show version box, and then chooses any command from the View menu other than by Icon or by Small Icon, filenames and their version numbers from the version resource appear in the active Finder window. The Finder also displays version information when the user selects your application and chooses Get Info from the File menu.

The version resource allows you to store a version number, a version message, and a region code. (Because the Get Info command's information window already displays the name of your application, the version message should not include the name of your application.) You can use version resources to assign version information to an individual file and, if it is a part of a larger collection of files, to the entire superset of files. The version resource with a resource ID number of 1 specifies the version of the file; the version resource with a resource ID number of 2 specifies the version of the set of files.

Each version resource should contain these elements:

■ Major revision level in binary-coded decimal format. Although the Finder doesn't display it anywhere, you can store this information here; most programming environments provide a tool for setting this element.

■ Minor revision level in binary-coded decimal format. Although the Finder doesn't display it anywhere, you can store this information here; most programming environments provide a tool for setting this element.

■ Development stage. You can use any of these values or the constants that represent them:

Value	Constant	Description
0x20	development	Prealpha file
0x40	alpha	Alpha file
0x60	beta	Beta file
0x80	release	Released file

■ Prerelease revision level. This number specifies the version if the software is still prerelease.

■ Region code. This identifies the script system for which this version of the software is intended. See the chapter "Script Manager" in *Inside Macintosh: Text* for information about the values represented by the various region codes that can be specified here.

■ Version number. This string identifies the version number of the software. When the user opens the Views control panel, clicks the Show version box, and then chooses any command from the View menu other than by Icon or by Small Icon, the Finder window containing this application displays this string.

■ Version message. This string identifies the version number and either a company copyright for a file or a product name for a superset of files. When the user selects this file and chooses the Get Info command, the Finder displays this string in the information window as follows:

□ For a version resource with a resource ID number of 1, this string is displayed in the version field of the information window.

□ For a version resource with a resource ID number of 2, this string is displayed beneath the file's name next to the file's icon at the top of the information window.

Listing 7-9 illustrates the version resources for a graphics application and for the document-processing system of which it is a part. Notice that the paint program is version 1.0 while the set of files that compose the entire document-processing system is version 2.0.

Listing 7-9   Rez input for a pair of version resources

```
resource 'vers' (1, purgeable) {
    0x01, 0x00, release, 0x00, verUS,
    "1.0",
    "1.0 (US), © My Company, Inc. 1992"
};
resource 'vers' (2, purgeable) {
    0x02, 0x00, release, 0x00, verUS,
    "2.0",
    "(for SurfWriter 3.0)"
};
```

Figure 7-13 illustrates how the Finder displays the information from these resources in its information window.

You can store version resources in any kind of file, not just an application. If your application does not contain a version resource with a resource ID number of 1, the Finder displays the string from your signature resource as the version information in the information window for your application.

Using Finder Information in the Catalog File

A catalog file exists on every volume to maintain relationships between the files and directories on that volume. (A volume is any storage medium formatted to contain files.) Although it's used mostly by the File Manager, the catalog file also contains information used by the Finder. The information for files is listed in file information records (data structures of type `FInfo`) and in extended file information records (data structures of type `FXInfo`). The information for directories is listed in directory information (`DInfo`) records and in extended (`DXInfo`) directory information records.

Figure 7-13   The version data in the information window
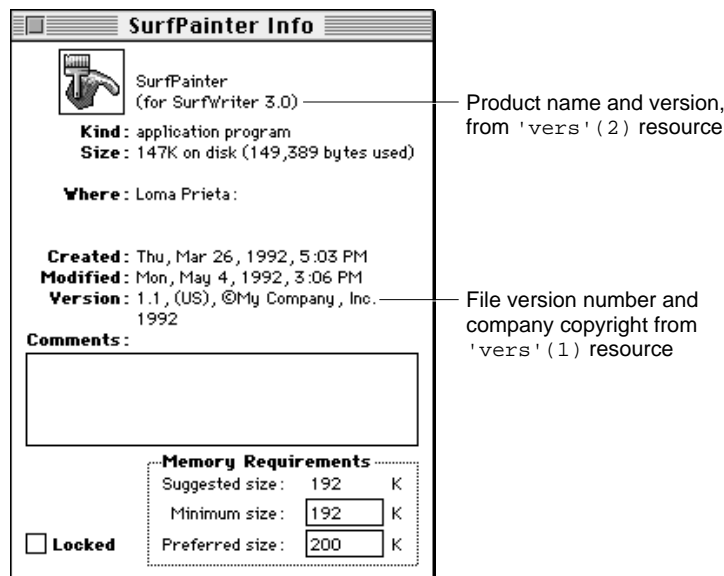
The Finder manipulates the fields in the file information, directory information, and extended directory information records; your application shouldn't have to directly check or set any of these fields.

Normally, your application sets the file type and the creator information in fields of the file's file information record when your application creates a new file; for example, the File Manager function `FSpCreate` (described in *Inside Macintosh: Files*) takes a creator and a file type as parameters. The Finder manipulates the other fields in the file information record, which is shown here:

```
TYPE  FInfo =
      RECORD
         fdType:     OSType;     {file type}
         fdCreator:  OSType;     {file creator}
         fdFlags:    Integer;    {Finder flags}
         fdLocation: Point;      {file's location in window}
         fdFldr:     Integer;    {directory that contains file}
      END;
```

After you have created a file, you can use the File Manager function `FSpGetFInfo` to return the file information record, then change the `fdType` and `fdCreator` fields by using the File Manager function `FSpSetFInfo`.

You can check the information in this record by calling the File Manager function
`FSpGetFInfo` or `PBGetCatInfo`. In particular, you may want to check the file type
or creator for a file, or you may want to check or set one of your document's Finder flags.
See "File Information Record" beginning on page 7-47 for a list of all the Finder flags.
The only Finder flags you might ever want to set are described here:

■ `isInvisible`. This flag specifies that a file is invisible from the Finder and from the
Standard File Package dialog boxes. Making a file invisible is generally not
recommended. Not even temporary files need to be invisible because the Temporary
Items folder into which they should be written is invisible. The Temporary Items
folder is described in "Using the System Folder and Its Related Directories" beginning
on page 7-41.

■ `hasBundle`. This flag specifies that a file has a bundle resource that associates the file
with your own icons. When the Finder displays or manipulates a file, it checks the
file's `hasBundle` bit (also called the **bundle bit**). If that bit is not set, the Finder
displays a default icon for that file type. If the `hasBundle` bit is set, the Finder checks
the `hasBeenInited` bit. If the `hasBeenInited` bit is set, the Finder uses the
information in the desktop database to display that file's icon. If the `hasBeenInited`
bit is not set, the Finder installs the information from the bundle resource in the
desktop database and sets the `hasBeenInited` bit. Most development environments
provide a simple tool for setting the bundle bit when you create your application.

■ `nameLocked`. This flag specifies that a file cannot be renamed from the Finder and
that the file cannot have customized icons assigned to it by users.

■ `isStationery`. This flag specifies that a file is a stationery pad. To support
stationery pads, your application should check this bit for every document passed to
it by either the Finder or the Standard File Package. (The File Manager functions
`StandardGetFile` and `CustomGetFile` return this flag in the `sfFlags` field of the
standard file reply record.) If the `isStationery` bit is set for a file that a user wants
to open, your application should copy the template's contents into a new document
and open the document in an untitled window. Stationery pads are described in the
next section.

■ `isShared`. This flag specifies that a file is an application that multiple users on a
network can execute simultaneously.

■ `hasCustomIcon`. This flag specifies that a file has a customized icon. "Creating
Customized Document Icons" beginning on page 7-17 explains how users or your
application can use customized icons.

Supporting Stationery Pads

Stationery pads are special documents that the user creates as templates. Opening
a stationery pad should not open the document itself; instead, it should open a new
document with the same contents as the stationery pad. To turn any document into
a stationery pad, the user selects it, chooses Get Info from the File menu, and clicks
the Stationery pad checkbox in the information window. The Finder tags a document as
being a stationery pad by setting the `isStationery` bit in the file's Finder flags field.

When the user opens a stationery pad from the Finder, the Finder first checks your application's size resource to see if your application supports stationery. The 'SIZE' resource tells the Finder and the Process Manager which features your application supports and how much memory to allocate when it starts up your application. Listing 7-10 illustrates a size resource.

Listing 7-10   Rez input for a size resource

```
resource 'SIZE' (-1, purgeable)  {
   reserved,
   acceptSuspendResumeEvents,
   reserved,
   canBackground,
   doesActivateOnFGSwitch,
   backgroundAndForeground,
   dontGetFrontClicks,
   ignoreAppDiedEvents,
   is32BitCompatible,
   isHighLevelEventAware,
   localAndRemoteHLEvents,
   isStationeryAware,          /*support stationery pads*/
   dontUseTextEditServices,
   reserved, reserved, reserved,
   kPrefSize * 1024,
   kMinSize * 1024
};
```

Notice that the twelfth field, isStationeryAware, tells the Finder that this application supports stationery pads.

If the isStationeryAware bit is not set in the size resource, the Finder creates a new document from the template and prompts the user for a name. The Finder then starts up your application as usual, passing it the name of the new document.

If the isStationeryAware bit is set, as shown in Listing 7-10, the Finder informs your application that the user has opened a document and passes your application the name of the stationery pad.

To support stationery, your application should

■ specify the isStationeryAware constant in its size resource

■ always check the isStationery bit of a document before opening it

Listing 7-11 on page 7-36 illustrates a simple function that takes a file system specification record and returns TRUE or FALSE, indicating whether the file is a stationery document or not.

Listing 7-11   Determining whether a document is a stationery pad

```
FUNCTION IsStationeryDoc (myFSSpec: FSSpec): Boolean;
VAR
    myErr:      OSErr;
    myFInfo:    FInfo;
BEGIN
    myErr := FSpGetFInfo(myFSSpec, myFInfo);
    IF myErr = noErr THEN
        IsStationeryDoc := BTST(myFInfo.fdFlags, isStationery)
    ELSE
        IsStationeryDoc := FALSE;
END;
```

The `isStationery` bit alone identifies whether a document is stationery. If the
`isStationery` bit is set for a file that the user wants to open, your application should
copy the template's contents into a new document and open the document in an untitled
window. (For information about opening documents and about the File Manager
function `FSpGetFInfo`, see *Inside Macintosh: Files*.)

Your application can check the `sfFlags` field of the standard file reply record to
determine whether the `isStationery` bit is set. Unlike the Finder, the Standard File
Package always passes your application the stationery pad itself, not a copy of it,
regardless of the setting of the `isStationery` bit. When the user opens a stationery
pad from within your application, the Standard File Package checks your application's
size resource. If your application does not support stationery, the Standard File Package
displays an alert box warning the user that the stationery pad itself, not a copy of it,
is being opened. As you can see, the user can still easily change the template and
mistakenly write over it by choosing Save without assigning a new name. You can
prevent this unnecessary user frustration by making your application stationery-aware.

You can supply the icon to be displayed for stationery pads created from your
application's documents by using the resources described in "Creating Icons for the
Finder" beginning on page 7-11. If you do not supply your own stationery pad icon, the
Finder uses the default stationery pad icon illustrated in Figure 7-5 on page 7-12.

In your documentation, tell users to choose the Get Info command to make stationery
pads. You may also want to give examples of useful stationery pads created with your
application. For example, if your application supports text and graphics, you may
provide samples of stationery pads for business letterheads or billing statements.

Distributing Fonts, Sounds, and Other Movable Resources

If you create fonts, sounds, keyboard layouts, and script system resource collections, you
can distribute them in individual, movable resource files.

Movable resources such as fonts, keyboard layouts, and sounds are represented on the
screen by icons. To install these resources, the user drags their icons to the System Folder

icon. The Finder puts font resources in the Fonts folder, and it puts the other resources in the System file. The user can determine which fonts are currently installed by double-clicking the System Folder to open it and then double-clicking the Fonts folder. By double-clicking the System file so that it opens like a folder, the user can see which other movable resources are installed. (For a description of the new organization of the System Folder, see "Using the System Folder and Its Related Directories" beginning on page 7-41.)

To make one of these resources visible on the screen, assign it one of the special file types defined by the Finder for movable resources. The following list shows the resources that can be moved, their assigned file types, and their icons:

Resource	File type	Large black-and-white icon
Font	`'ffil'`	
Keyboard layout	`'kfil'`	
Script system resource collection	`'ifil'`	
Sound	`'sfil'`	
TrueType font	`'tfil'`	

Note

You or your users can give customized icons to these file types (as described in "Creating Customized Document Icons" beginning on page 7-17) as long as the files are not installed in the System file or in a suitcase file. As soon as users install them in the System file or in a suitcase file, the Finder displays them using the icons shown in the previous list. Font and TrueType font movable resources retain their custom icons when installed in the Fonts folder. ◆

The user can still store fonts (as well as desk accessories) in files that have suitcase icons, which is how they were distributed for installation or saved by the user using the Font/DA Mover in versions of system software that preceded System 7. A suitcase file that holds desk accessories is of type `'DFIL'`, and a suitcase file that holds fonts is of type `'FFIL'`. All suitcase files have a creator of `'DMOV'`.

In your documentation, tell users to install fonts, sounds, or script system resource collections by dragging their icons to the System Folder icon. A dialog box appears asking the user to verify that the resource should be installed in either the Fonts folder or the System file. The user clicks OK to accept the installation. The user also has the option to click Cancel to prevent the installation.
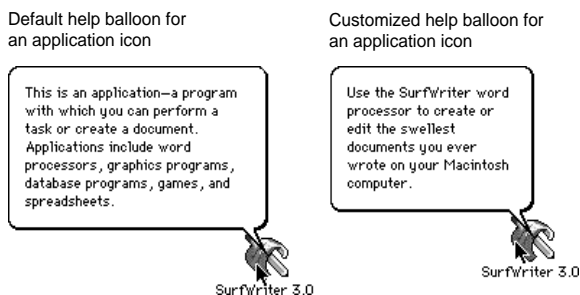
Note

If users drag icons to the open System Folder window instead of to the System Folder icon, the Finder copies or moves the files into the System Folder directory instead of installing them into either the Fonts folder or the System file. ◆

Providing Balloon Help for Nondocument Icons

The Finder offers Balloon Help online assistance for users. After the user chooses Show Balloons from the Help menu, descriptive help balloons appear when the user moves the cursor to an area of the screen (such as a menu, a window control, or a dialog box) that has a help resource associated with it.

The Finder provides default help balloons for application, control panel, and system extension icons. You can provide a customized help balloon for your application, control panel, or system extension icon by adding an `'hfdr'` resource with resource ID –5696 to the resource fork of your application. Figure 7-14 compares the default help balloon with a customized help balloon for the SurfWriter application icon.

Figure 7-14   Default and customized help balloons for application icons

Listing 7-12 shows a Finder help override resource and its associated `'STR '` resource, which are used for the customized help balloon shown in Figure 7-14.

Note

You cannot override the default help balloon that the Finder uses for document icons. ◆

The chapter "Help Manager" in *Inside Macintosh: More Macintosh Toolbox* describes in detail how to provide Balloon Help for your application icon and for other elements of your application.

Listing 7-12   Rez input for a help balloon resource for an application icon

```
resource 'hfdr' (-5696, purgeable) {    /*help for SurfWriter icon*/
   HelpMgrVersion, hmDefaultOptions, 0, 0,    /*header information*/
   {HMSTRResItem {kIconHelpString}}
};
resource 'STR ' (kIconHelpString, purgeable) {/*help message for app icon*/
   "Use the SurfWriter word processor to create or edit the "
   "swellest documents you ever wrote on your Macintosh computer."
};
```

Using Aliases

The Finder allows the user to create multiple icons to represent a single document or other desktop object (such as a disk, a folder, or the Trash). One of the icons represents the actual file; the others are aliases that point to the file. An **alias** is an object that represents some other file, directory, or volume. An alias looks like the icon of its target, but its name is displayed in a different style. The style depends on the system script; for Roman and most other scripts, alias names are displayed in italic.

To the user, the icons of the actual file and its aliases are functionally identical. Aliases give the user more flexibility in organizing files and offer a convenient way to store a local copy of a large or dynamic file that resides on a file server.

Ordinarily, when the user wants to open or print files, your application does not need to be concerned with whether they are aliases because both the Finder and the Standard File Package resolve aliases before passing them to your application. If the user opens an alias that represents a document created by your application, the Finder passes your application the name and location of the document itself, not the alias. Similarly, when the user opens an alias from within your application, the Standard File Package passes your application the name of the target document.

If your application opens a file or a directory without going through the Finder or the Standard File Package (if, for example, it uses preference files or dictionary files), your application should always call the ResolveAliasFile function just before opening the file.

As a Finder object, the alias depicts a file called the **alias file,** which contains a record that points to the file, directory, or volume represented by the icon. Alias files are created and managed by the user through the Finder.

Although your application shouldn't create alias files or change users' aliases, your application can create and use its own alias records for storing identifying information about files or directories. An **alias record** is a data structure that identifies a file, folder, or volume. Whenever your application needs to store file or directory information, you can record the location and other identifying information in an alias record. The next time your application needs the file or directory, you can use the Alias Manager to locate it, even if the user has renamed it, copied it, restored it from backup, or moved it. You can

also use alias records to identify objects on other volumes, including AppleShare volumes. See the chapter "Alias Manager" in *Inside Macintosh: Files* for details about creating and managing information in alias records.

An alias file contains an alias record, stored as a resource of type `'alis'`, that points to the target of the alias. (The **alias target** is the file, directory, or volume described by the alias record.) The alias file might also contain the target object's icon descriptions. The Finder identifies an alias file by setting the `isAlias` bit in the file's Finder flags field (see "File Information Record" beginning on page 7-47 for a description of Finder flags).

An alias file that represents a document typically has the same type and creator as the file it represents. However, many Finder objects—such as disks, folders, and the Trash—do not have file types. Instead, alias files for these objects are assigned special file types, called *alias types.* Here are the alias types for those objects for which users can create aliases:

Object	Alias type	Constant
Apple Menu Items folder	`'faam'`	`kAppleMenuFolderAliasType`
AppleShare drop folder	`'fadr'`	`kDropFolderAliasType`
Application	`'adrp'`	`kApplicationAliasType`
Control Panels folder	`'fact'`	`kControlPanelFolderAliasType`
Exported AppleShare folder	`'faet'`	`kExportedFolderAliasType`
Extensions folder	`'faex'`	`kExtensionFolderAliasType`
File server	`'srvr'`	`kContainerServerAliasType`
Floppy disk	`'flpy'`	`kContainerFloppyAliasType`
Folder	`'fdrp'`	`kContainerFolderAliasType`
Hard disk	`'hdsk'`	`kContainerHardDiskAliasType`
Mounted AppleShare folder	`'famn'`	`kMountedFolderAliasType`
Other objects that can hold files	`'drop'`	`kContainerAliasType`
Preferences folder	`'fapf'`	`kPreferencesFolderAliasType`
PrintMonitor Documents folder	`'fapn'`	`kPrintMonitorDocsFolderAliasType`
Shared AppleShare folder	`'fash'`	`kSharedFolderAliasType`
Startup Items folder	`'fast'`	`kStartupFolderAliasType`
System Folder	`'fasy'`	`kSystemFolderAliasType`
Trash	`'trsh'`	`kContainerTrashAliasType`

(The Extensions, Preferences, Apple Menu Items, Control Panels, Startup Items, and PrintMonitor Documents folders are described in "Using the System Folder and Its Related Directories" beginning on page 7-41.)

When opening a file without going through the Finder or the Standard File Package, you call `ResolveAliasFile` immediately before opening the file. (The `ResolveAliasFile` function is described in detail on page 7-52.) In Listing 7-13, the customized open function `MyOpen` ensures that the file to be opened is the target file and then opens the data fork with the File Manager function `FSpOpenDF`.

Listing 7-13   Using the `ResolveAliasFile` function to open a file

```
FUNCTION MyOpen (VAR theSpec: FSSpec; perm: SignedByte;
                VAR fRefNum: Integer): OSErr;
VAR
   myErr:           OSErr;
   targetIsFolder:  Boolean;
   wasAliased:      Boolean;
BEGIN
   myErr := ResolveAliasFile(theSpec, TRUE, targetIsFolder, wasAliased);
   IF targetIsFolder THEN
      myErr := paramErr                  {cannot open a folder}
   ELSE IF (myErr <> noErr ) THEN      {try to open it}
      myErr := FSpOpenDF(theSpec, perm, fRefNum);
   MyOpen := myErr;
END;
```

Using the System Folder and Its Related Directories

The System Folder is a directory that stores essential system software such as the System file, the Finder, and printer drivers. System 7 introduced a new organization for the System Folder, which contains a set of new subdirectories to hold related files. The Finder uses these subdirectories to facilitate file management for the user. For example, by sorting and storing such files as desk accessories, control panels, fonts, preferences files, system extensions, and temporary files into separate folders for the user, the Finder keeps the top level of the System Folder from being cluttered with dozens, or even hundreds, of files.

The user can easily install and remove fonts, sounds, keyboard layouts, control panels, and system extensions by dragging their icons to the System Folder icon. The Finder then moves them into the proper subdirectories. When a control panel icon is dragged to the System Folder icon, for example, the Finder presents a dialog box that asks the user, "Place this control panel into the 'Control Panels' folder?" The user accepts by clicking the OK button or declines by clicking the Cancel button.

Note
If users drag icons to the open System Folder window instead of to the System Folder icon, the Finder copies or moves the files into the System Folder directory instead of copying or moving them to the proper subdirectories. ◆

Figure 7-15 shows a user's view of the new directory organization typically found within the System Folder.

Figure 7-15 The System Folder and related folders
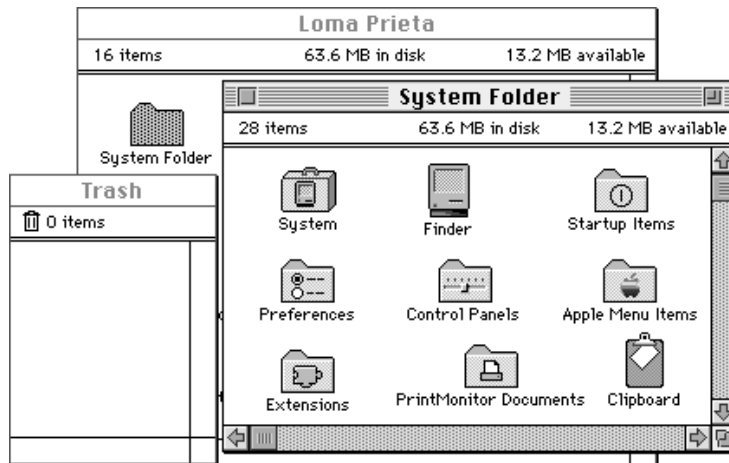
Additional related directories are located at the root directory. Notice the Trash window. It shows the contents of the Trash directory, which is represented to the user by the Trash icon. The Trash directory exists at the root level of the volume. A Macintosh sharing files among users in a network environment maintains separate Trash subdirectories within a shared Trash directory. That is, the server creates a separate, uniquely named Trash subdirectory for every user who opens a volume on a Macintosh server and drags an object to the Trash icon. All Trash subdirectories within a shared Trash directory are invisible to users. On the desktop, the user sees only the Trash icon of the local Macintosh computer. When the user double-clicks the Trash icon, a window reveals the names of only those files that the user has thrown away; no distinction is made to the user as to which computers any of these files originated on.

At the root level of the volume, the Finder also maintains a Temporary Items folder and a Desktop Folder, both of which are invisible to the user and so don't appear in Figure 7-15.

Figure 7-15 illustrates the folder organization typically found on single-user systems. Of all the related directories shown, your application is likely to use only the Preferences folder and the Temporary Items folder. However, you cannot be certain of the location of these or any of the other system-related directories. In the future, these system-related directories may not be located in the System Folder or in the root directory.

You can use the `FindFolder` function (described on page 7-54) to get the path information to these directories. Of these directories, the only ones you are ever likely to need are Preferences, Temporary Items, and Trash. For example, you might wish to check for the existence of a user's configuration file in Preferences, create a temporary file in Temporary Items, or—if your application runs out of storage when trying to save a file—check how much storage is taken by items in the Trash directory and report this to the user.

Your application may freely use these two directories for storing and locating important files:

■ **Preferences,** located in the System Folder, holds preferences files to record local configuration settings. Your application can store its preferences file in this directory. The active Finder Preferences file is always stored in the Preferences folder. Do not use the Preferences folder to hold information that is to be shared by users on more than one Macintosh computer on a network. Ensure that your application can always operate even if its preferences file has been deleted.

■ **Temporary Items,** located at the root level of the volume, holds temporary files created by applications. The Temporary Items folder is invisible to the user. Your application can place its temporary files in this directory. A temporary file should exist only as long as your application needs to keep it open. As soon as your application closes the file, your application should remove the temporary file. You should also ensure that you are assigning a unique name to your temporary file so that you don't write over another application's file.

It's important to bear in mind a few rules about storing your application's files. First, don't store any files at the top level of the System Folder. Use the Preferences directory or one of the other directories described in the following list.

Second, use the `FindFolder` function to locate or put files in the right place. Don't assume files are on the same volume as your application; they could be on a different local volume, or on a remote volume on the network.

Third, don't store any files that multiple users may need to access, such as dictionaries and format converters, in the Preferences directory or in any of the directories located in the System Folder. Remember that the files in the System Folder are generally accessible only to the person who starts up from the System file in that System Folder.

There are additional directories that either the user or the Finder uses for storing and locating important files; these directories are described here. Generally, your application should not store files in these directories.

■ **Apple Menu Items,** located in the System Folder, holds the standard desk accessories plus any other desk accessories, applications, files, folders, or aliases that the user wants to display in the Apple menu. Only the user and the Installer should put things into the Apple Menu Items folder.

■ **Control Panels,** located in the System Folder, holds control panels. The Apple Menu Items folder holds an alias to the Control Panels folder so that the user can also reach the control panels through the Apple menu. Only the user and the Installer should put things into the Control Panels folder.

■ **Desktop Folder,** which is invisible to users, is located at the root level of the volume. The Desktop Folder stores information about the icons that appear on the desktop area of the screen. The user controls the contents of the Desktop Folder by arranging icons on the screen. What appears on the screen to the user is the union of the contents of Desktop Folders for all mounted volumes.

■ **Extensions,** located in the System Folder, holds extensions—that is, code that is not part of the basic system software but that provides system-level services, such as printer drivers and system extensions. Files of type `'INIT'`, previously called startup documents, and of type `'appe'`, also known as background-only applications, are

routed by the Finder to this folder. Files of type `'scri'` (system extensions for script systems) are also routed to this folder. Only the user and the Installer should put files into the Extensions folder.

■ **Fonts,** located in the System Folder on computers using system software version 7.1 or later, holds fonts. Only the user and the Installer should put fonts into the Fonts folder.

■ **PrintMonitor Documents,** located in the System Folder, holds spooled documents waiting to be printed. Only the printing software uses the PrintMonitor Documents folder.

■ **Rescued Items from *volume name*,** located in the Trash directory, is a directory created by the Finder at system startup, restart, or shutdown only when the Finder finds items in the Temporary Items folder. Since applications should remove their temporary files when they close them, the existence of a file in a Temporary Items folder indicates a system crash. When the Finder discovers a file in the Temporary Items folder, the Finder creates a Rescued Items from *volume name* directory that is named for the volume on which the Temporary Items folder exists. For example, the Finder creates a directory called Rescued Items from Loma Prieta when a file is discovered in the Temporary Items folder on a volume named Loma Prieta. The Finder then moves the temporary file to that directory so that users can examine the file in case they want to recreate their work up to the time of the system crash. When a user empties the Trash, all Rescued Items folders disappear. Only the Finder should put anything into Rescued Items directories.

■ **Startup Items,** located in the System Folder, holds applications and desk accessories (or their aliases) that the user wants started up every time the Finder starts up. Only the user should put things into the Startup Items folder. Note that there is a distinction between startup applications that users put in the Startup Items folder and system extensions of file type `'INIT'` (previously called startup documents), which are typically installed in the Extensions folder.

■ **System file,** located in the System Folder, contains the basic system software plus some system resources, such as sound and keyboard resources. The System file behaves like a folder in this regard: although it looks like a suitcase icon, double-clicking it opens a window that reveals movable resource files (such as sounds, keyboard layouts, and script system resource collections) stored in the System file. ("Distributing Fonts, Sounds, and Other Movable Resources" beginning on page 7-36 describes the resources that can be moved into the System file.) Only the user and the Installer should put resources into the System file.

■ **Trash,** located at the root level of a volume, holds items that the user moves to the Trash icon. After opening the Trash icon, the user sees the collection of all items that he or she has moved to the Trash icon—that is, the union of all appropriate Trash directories from all mounted volumes. A Macintosh set up to share files among users in a network environment maintains separate Trash subdirectories for remote users within its shared Trash directory. That is, the server creates a separate, uniquely named Trash subdirectory for every remote user who opens a volume on a Macintosh file server and drags an object to the Trash icon. All Trash subdirectories and the shared Trash directory are invisible to users. The Finder empties a Trash directory (or, in the case of a file server, a Trash subdirectory) only when the user of that directory chooses the Empty Trash command.

Although the names of the visible system-related folders vary on different international systems, the invisible directories Temporary Items and Desktop Folder keep these names on all systems. System software assigns unique names for invisible Trash subdirectories.

Generally, you should store application-specific files in the folder with your application, not in any of these system-related directories. Your application may want to provide users with a mechanism to specify a directory in which to look for auxiliary files. For example, you could design a customized version of the open file dialog box that allows users to specify a path to locations where files are stored. This technique may be useful for finding files that are shared by several applications. It's also possible to track the location of files by using the Alias Manager. For details, see the chapter "Alias Manager" in *Inside Macintosh: Files*.

When you design your application, it's important to consider the user's view of the tools that you provide. In most cases you'll want to build your application so that the user deals with one icon that represents the entire set of abilities your application provides. This scheme simplifies the user's world by restricting the complexity of installing and maintaining your product. If you provide optional tools—such as a dictionary and thesaurus—that have their own icons, it's a good idea to allow these tools to work from any location in the file system rather than relying on their storage somewhere in the System Folder.

The Desktop Database

For quick access to the resources it needs, the Finder maintains a central desktop database of information about the files and directories on a volume. The Finder updates the database when applications are added, moved, renamed, or deleted.

Normally, your application won't need to use the information in the desktop database or to use Desktop Manager routines to manipulate it. Instead, your application should let the Finder manipulate the desktop database and handle such Desktop Manager tasks as launching applications when users double-click icons, maintaining user comments associated with files, and managing the icons used by applications.

In case you discover some important need to retrieve information from the desktop database or even to change the desktop database from within your application, Desktop Manager routines are provided for you to do so. While your application probably won't ever need to use them, for the sake of completeness they are described in *Inside Macintosh: More Macintosh Toolbox*.

Much of the information in the desktop database comes from the bundle resources for applications and other files on the volume. (See "Using Finder Information in the Catalog File" beginning on page 7-32 for a discussion on setting the bundle bit of an application so that its bundled resources get stored in the desktop database.) The desktop database contains all icon definitions and their associated file types. It lists all the file types that each application can open and all copies or versions of the application that's listed as the creator of a file. The desktop database also lists the location of each application on the disk and any comments that the user has added to the information windows for desktop objects.

The Finder maintains a desktop database for each volume with a capacity greater than 2 MB. For most volumes, such as hard disks, the database is stored on the volume itself. For read-only volumes—such as some compact discs—that don't contain their own desktop database, the Desktop Manager creates it and stores it in the System Folder of the startup drive.

For compatibility with older versions of system software, the Finder keeps the information for ejectable volumes with a capacity smaller than 2 MB in a resource file instead of a database.

Finder Interface Reference

This section describes the data structures, routines, and resources that are specific to the Finder interface.

The "Data Structures" section shows the data structures for the file information record, the extended file information record, the directory information record, and the extended directory information record. The "Routines" section describes the routines for resolving alias files and for finding system-related folders. The "Resources" section describes the resources you supply for your files so that the Finder can relay information about them to your users.

Data Structures

A catalog file exists on every volume to maintain relationships between the files and directories on that volume. (A volume is any storage medium formatted to contain files.) Although it's used mostly by the File Manager, the catalog file also contains information used by the Finder. The information for files is listed in file information records and extended file information records; the information for directories is listed in directory information records and extended directory information records.

Normally, your application sets the file type and the creator information in fields of a file information record when your application creates a new file. (For a complete discussion of the File Manager and the functions available for creating files, see *Inside Macintosh: Files*.) The Finder manipulates the other fields in the file information record. You can check the information in this record by calling the File Manager function `FSpGetFInfo` or `PBGetCatInfo`. In particular, you may want to check the file type or creator for a file, or you may want to check or set one of your document's Finder flags.

The Finder manipulates the fields in the extended file information, directory information, and extended directory information records; your application shouldn't have to directly check or set any of these fields. These data structures are described here for completeness.

File Information Record

You typically set a file's type and creator when you create the file; for example, you pass a creator and a file type to the File Manager function `FSpCreate` as parameters. The Finder manipulates the other fields in the file information record, which is a data structure of type `FInfo`. After you have created a file, you can use the File Manager function `FSpGetFInfo` to return the file information record, then change the `fdType` and `fdCreator` fields by using the File Manager function `FSpSetFInfo`.

```
TYPE  FInfo =
      RECORD
          fdType:    OSType;     {file type}
          fdCreator: OSType;     {file creator}
          fdFlags:   Integer;    {Finder flags}
          fdLocation: Point;     {file's location in window}
          fdFldr:    Integer;    {window that contains file}
      END;
```

Field descriptions

`fdType`	File type. For a discussion of file types, see "Giving a Signature to Your Application and a Creator and a File Type to Your Documents" beginning on page 7-8.
`fdCreator`	The signature of the application that created the file. For a discussion about creators, see "Giving a Signature to Your Application and a Creator and a File Type to Your Documents" beginning on page 7-8.
`fdFlags`	Finder flags. There are only a few flags that your application might ever need to set; these are described in "Using Finder Information in the Catalog File" beginning on page 7-32. All of the Finder flags are listed here for completeness.

Flag name	Bit number	Description
`isAlias`	15	For a file, this bit indicates that the file is an alias file. For directories, this bit is reserved—in which case, set to 0.
`isInvisible`	14	The file or directory is invisible from the Finder and from the Standard File Package dialog boxes.
`hasBundle`	13	For a file, this bit indicates that the file contains a bundle resource. For directories, this bit is reserved—in which case, set to 0.
`nameLocked`	12	The file or directory can't be renamed from the Finder, and the icon cannot be changed.
`isStationery`	11	For a file, this bit indicates that the file is a stationery pad. For directories, this bit is reserved—in which case, set to 0.

Flag name	Bit number	Description
hasCustomIcon	10	The file or directory contains a customized icon.
Reserved	9	Reserved; set to 0.
hasBeenInited	8	The Finder has recorded information from the file's bundle resource into the desktop database and given the file or folder a position on the desktop.
hasNoINITS	7	The file contains no 'INIT' resources; set to 0. Reserved for directories; set to 0.
isShared	6	The file is an application that can be executed by multiple users simultaneously. Defined only for applications; otherwise, set to 0.
requiresSwitchLaunch	5	Unused and reserved in System 7; set to 0.
colorReserved	4	Unused and reserved in System 7; set to 0.
color	1–3	Three bits of color coding.
isOnDesk	0	Unused and reserved in System 7; set to 0.

You can use these constants as masks for these flags:

```
CONST
fHasBundle  = 8192;    {set if file has a bundle }
                       { resource}
fInvisible  = 16384;   {set if icon is invisible}
kIsOnDesk   = $1;      {unused and reserved in }
                       { System 7}
kColor      = $E;      {three bits of color }
                       { coding}
kIsShared   = $40;     {file can be executed by }
                       { multiple users }
                       { simultaneously}
kHasBeenInited
            = $100;    {file info is in desktop }
                       { database}
kHasCustomIcon
            = $400;    {file or directory has a }
                       { customized icon}
kIsStationery
            = $800;    {file is a stationery pad}
kNameLocked = $1000;   {file or directory can't }
                       { be renamed from Finder, }
                       { and icon can't be }
                       { changed}
kHasBundle  = $2000;   {file has bundle resource}
```

7

Finder Interface

```
                        kIsInvisible= $4000;    {file or directory is }
                                                { invisible from Finder & }
                                                { from Standard File }
                                                { Package dialog boxes}
                        kIsAlias   = $8000;    {file is an alias file}
```

fdLocation   The location—specified in coordinates local to the window—of the
                      file's icon within its window.

fdFldr   The window in which the file's icon appears; this information is
                      meaningful only to the Finder.

Extended File Information Record

The Finder manipulates the fields in the extended file information records, which are
data structures of type FXInfo; your application shouldn't have to check or set any of
these fields directly.

```
TYPE  FXInfo =
     RECORD
        fdIconID:       Integer;    {icon ID}
        fdUnused:       ARRAY[1..3] OF Integer;
                                    {unused but reserved 6 bytes}
        fdScript:       SignedByte; {script flag and code}
        fdXFlags:       SignedByte; {reserved}
        fdComment:      Integer;    {comment ID}
        fdPutAway:      LongInt;    {home directory ID}
     END;
```

Field descriptions

fdIconID   An ID number for the file's icon; the numbers that identify icons are
                      assigned by the Finder.

fdUnused   Reserved.

fdScript   The script system for displaying the file's name. Ordinarily, the
                      Finder (and the Standard File Package) displays the names of all
                      desktop objects in the system script, which depends on the
                      region-specific configuration of the system. The high bit of the byte
                      in the fdScript field is set by default to 0, which causes the Finder
                      to display the filename in the current system script. If the high bit is
                      set to 1, the Finder (and the Standard File Package) displays the
                      filename and directory name in the script whose code is recorded in
                      the remaining 7 bits.

fdXFlags   Reserved.

fdComment   An ID number for the comment that is displayed in the information
                      window when the user selects a file and chooses the Get Info
                      command from the File menu. The numbers that identify comments
                      are assigned by the Finder.

fdPutAway   If the user moves the file onto the desktop, the directory ID of the
                      folder from which the user moves the file.

Directory Information Record

The Finder manipulates the fields in the directory information record, which is a data structure of type `DInfo`. Your application shouldn't have to check or set any of these fields directly.

```
TYPE DInfo =
     RECORD
         frRect:     Rect;     {folder's window rectangle}
         frFlags:    Integer;  {flags}
         frLocation: Point;    {folder's location in window}
         frView:     Integer;  {folder's view}
     END;
```

Field descriptions

frRect	The rectangle for the window that the Finder displays when the user opens the folder.
frFlags	Reserved.
frLocation	Location of the folder in the parent window.
frView	The manner in which folders are displayed; this is set by the user with commands from the View menu of the Finder.

Extended Directory Information Record

The Finder manipulates the fields in the extended directory information records, which are data structures of type `DXInfo`; your application shouldn't have to check or set any of these fields directly.

```
TYPE  DXInfo =
     RECORD
         frScroll:     Point;      {scroll position}
         frOpenChain:  LongInt;    {directory ID chain of open }
                                   { folders}
         frScript:     SignedByte; {script flag and code}
         frXFlags:     SignedByte; {reserved}
         frComment:    Integer;    {comment ID}
         frPutAway:    LongInt;    {home directory ID}
     END;
```

Field descriptions

frScroll	Scroll position within the Finder window. The Finder does not necessarily save this position immediately upon user action.

frOpenChain	Chain of directory IDs for open folders. The Finder numbers directory IDs. The Finder does not necessarily save this information immediately upon user action.
frScript	The script system for displaying the folder's name. Ordinarily, the Finder (and the Standard File Package) displays the names of all desktop objects in the current system script, which depends on the region-specific configuration of the system. The high bit of the byte in the fdScript field is set by default to 0, which causes the Finder to display the folder's name in the current system script. If the high bit is set to 1, the Finder (and the Standard File Package) displays the filename and directory name in the script whose code is recorded in the remaining 7 bits. However, as of system software version 7.1, the Window Manager and Dialog Manager do not support multiple simultaneous scripts, so the system script is always used for displaying filenames and directory names in dialog boxes, window titles, and other user interface elements used by the Finder. Therefore, until the system software's script capability is fully implemented, you should treat this field as reserved.
frXFlags	Reserved.
frComment	An ID number for the comment that is displayed in the information window when the user selects a folder and chooses the Get Info command from the File menu. The numbers that identify comments are assigned by the Finder.
frPutAway	If the user moves the folder onto the desktop, the directory ID of the folder from which the user moves it.

Routines

This section describes the routines your application can use to resolve alias files if it bypasses the Finder when manipulating documents and to find system-related folders if your application needs to determine where they are located.

Resolving Alias Files

Ordinarily, when the user wants to open or print files, your application does not need to be concerned with whether they are aliases because the Finder resolves aliases before passing them to your application. If the user opens an alias that represents a document created by your application, the Finder passes your application the name and location of the document itself, not the alias. (Similarly, when the user opens an alias from within your application, the Standard File Package passes your application the name of the target document.) If your application bypasses the Finder when manipulating documents, it should check for and resolve aliases itself by using the Alias Manager function ResolveAliasFile, which is described here for completeness.

ResolveAliasFile

If your application bypasses the Finder when manipulating documents, it should check for and resolve aliases itself by using the `ResolveAliasFile` function.

```
FUNCTION ResolveAliasFile (VAR theSpec: FSSpec;
                                resolveAliasChains: Boolean;
                                VAR targetIsFolder: Boolean;
                                VAR wasAliased: Boolean): OSErr;
```

`theSpec`   A file system specification record for the file or directory you plan to open.

`resolveAliasChains`
               A Boolean value. Set this parameter to `TRUE` if you want `ResolveAliasFile` to resolve all aliases in a chain, stopping only when it reaches the target file. Set this parameter to `FALSE` if you want to resolve only one alias file, even if the target is another alias file.

`targetIsFolder`
               A return parameter only. The `ResolveAliasFile` function returns `TRUE` in this parameter if the file specification record in the parameter `theSpec` points to a directory or a volume; otherwise, `ResolveAliasFile` returns `FALSE` in this parameter.

`wasAliased`
               A return parameter only. The `ResolveAliasFile` function returns `TRUE` in this parameter if the file specification record in the parameter `theSpec` points to an alias; otherwise, `ResolveAliasFile` returns `FALSE` in this parameter.

DESCRIPTION

The `ResolveAliasFile` function returns in the parameter `theSpec` the name and location of the target file that you initially pass in the parameter `theSpec`.

The `ResolveAliasFile` function first checks the catalog file for the file or directory specified in the parameter `theSpec` to determine whether it is an alias and whether it is a file or a directory. If the object is not an alias, `ResolveAliasFile` leaves `theSpec` unchanged, sets the `targetIsFolder` parameter to `TRUE` for a directory or volume and `FALSE` for a file, sets `wasAliased` to `FALSE`, and returns `noErr`. If the object is an alias, `ResolveAliasFile` resolves it, places the target in the parameter `theSpec`, and sets the `wasAliased` flag to `TRUE`.

When `ResolveAliasFile` finds the specified volume and parent directory but fails to find the target file or directory in that location, `ResolveAliasFile` returns a result code of `fnfErr` and fills in the parameter `theSpec` with a complete file system specification record describing the target (that is, its volume reference number, parent directory ID, and filename or folder name). The file system specification record is valid,

although the object it describes does not exist. This information is intended as a "hint" that lets you explore possible solutions to the resolution failure. You can, for example, use the file system specification record to create a replacement for a missing file with the File Manager function `FSpCreate`.

If `ResolveAliasFile` receives an error code while resolving an alias, it leaves the input parameters as they are and exits, returning an error code. In addition to any of these result codes, `ResolveAliasFile` can also return any Resource Manager or File Manager errors.

SPECIAL CONSIDERATIONS

Before calling the `ResolveAliasFile` function, you should make sure that it is available by using the `Gestalt` function with the `gestaltAliasMgrAttr` selector.

RESULT CODES

noErr	0	No error
nsvErr	–35	Volume not found
fnfErr	–43	Target not found, but volume and parent directory found, and `theSpec` parameter contains a valid file system specification record
dirNFErr	–120	Parent directory not found

SEE ALSO

Listing 7-13 on page 7-41 illustrates how to use `ResolveAliasFile` from an application's own `MyOpen` function. The file system specification record is described in *Inside Macintosh: Files*. Aliases and other Alias Manager and File Manager routines are also described in greater detail in *Inside Macintosh: Files*. The `Gestalt` function is described in the chapter "Gestalt Manager" in *Inside Macintosh: Operating System Utilities*.

Finding Directories

You can use the `FindFolder` function to get the path information you need to gain access to the system-related directories described in "Using the System Folder and Its Related Directories" beginning on page 7-41. Those you're most likely to want to access are Preferences, Temporary Items, and Trash. For example, you might wish to check for the existence of a user's configuration file in Preferences, create a temporary file in Temporary Items, or—if your application runs out of disk storage when trying to save a file—check how much disk storage is taken by items in the Trash directory and report this to the user.

FindFolder

To get the path information to gain access to the system-related directories, use the
`FindFolder` function.

```
FUNCTION FindFolder (vRefNum: Integer; folderType: OSType;
                     createFolder: Boolean;
                     VAR foundVRefNum: Integer;
                     VAR foundDirID: LongInt): OSErr;
```

vRefNum   The volume reference number (or the constant `kOnSystemDisk` for the
             startup disk) of the volume on which you want to locate a directory.

folderType
             A four-character folder type, or a constant that represents the type, for the
             directory you want to find. The constants and the four-character folder
             types they represent are listed here:

```
CONST
kAppleMenuFolderType
                      = 'amnu';    {Apple Menu Items}
kControlPanelFolderType
                      = 'ctrl';    {Control Panels}
kDesktopFolderType    = 'desk';    {Desktop Folder}
kExtensionFolderType
                      = 'extn';    {Extensions}
kFontsFolderType      = 'font';    {Fonts folder}
kPreferencesFolderType
                      = 'pref';    {Preferences}
kPrintMonitorDocsFolderType
                      = 'prnt';    {PrintMonitor  }
                                   { Documents}
kStartupFolderType    = 'strt';    {Startup Items}
kSystemFolderType     = 'macs';    {System Folder}
kTemporaryFolderType
                      = 'temp';    {Temporary Items}
kTrashFolderType      = 'trsh';    {single-user Trash}
kWhereToEmptyTrashFolderType
                      = 'empt';    {shared Trash on net}
```

createFolder
             Pass the constant `kCreateFolder` in this parameter to create a directory
             if it does not already exist; otherwise, pass the constant
             `kDontCreateFolder`.

foundVRefNum

> The volume reference number, returned by `FindFolder`, for the volume containing the directory you specify in the `folderType` parameter.

foundDirID

> The directory ID number, returned by `FindFolder`, for the directory you specify in the `folderType` parameter.

DESCRIPTION

For the folder type on the particular volume (specified, respectively, in the `folderType` and `vRefNum` parameters), the `FindFolder` function returns the directory's volume reference number in the `foundVRefNum` parameter and its directory ID in the `foundDirID` parameter.

The specified folder used for a given volume might be located on a different volume in future versions of system software; therefore, do not assume the volume that you specify in `vRefNum` and the volume returned in `foundVRefNum` will be the same.

Specify a volume reference number (or the constant `kOnSystemDisk` for the startup disk) in the `vRefNum` parameter.

Specify a four-character folder type—or the constant that represents it—in the `folderType` parameter. Use the `kTrashFolderType` constant to locate the current user's Trash directory for a given volume—even one located on a file server. On a file server, you can use the `kWhereToEmptyTrashFolderType` constant to locate the parent directory of all logged-on users' Trash subdirectories.

Use the constant `kCreateFolder` in the `createFolder` parameter to tell `FindFolder` to create a directory if it does not already exist; otherwise, use the constant `kDontCreateFolder`. Directories inside the System Folder are created only if the System Folder directory exists. The `FindFolder` function will not create a System Folder directory even if you specify the `kCreateFolder` constant in the `createFolder` parameter.

The `FindFolder` function returns a nonzero result code if the folder isn't found, and it can also return other file system errors reported by the File Manager or Memory Manager.

SPECIAL CONSIDERATIONS

The Finder identifies the subdirectories of the System Folder, and their folder types, in a resource of type `'fld#'` located in the System file. Do not modify or rely on the contents of the `'fld#'` resource in the System file; use only the `FindFolder` function to find the appropriate directories.

To determine the availability of the `FindFolder` function, use the `Gestalt` function with the Gestalt selector `gestaltFindFolderAttr`. Test the bit field indicated by the `gestaltFindFolderPresent` constant in the response parameter. If the bit is set, then the `FindFolder` function is present.

```
CONST gestaltFindFolderPresent = 0;    {if this bit is set, }
                                       { FindFolder is present}
```

RESULT CODES

noErr	0	No error
fnfErr	–43	Type not found in 'fld#' resource, or disk doesn't have System Folder support or System Folder in volume header, or disk does not have desktop database support for Desktop Folder—in all cases, folder not found
dupFNErr	–48	File found instead of folder

SEE ALSO

The system-related directories located by the FindFolder function are described in "Using the System Folder and Its Related Directories" beginning on page 7-41.

Resources

This section describes the resources you supply for your files so that the Finder can use your files and relay information about them to your users. These resources are

■ the signature resource—defined using a string ('STR ') resource—which the Finder uses to identify and start up your application when a user double-clicks documents created by your application

■ the set of resources (icon list resource, small icon list resource, large 4-bit color icon resource, small 4-bit color icon resource, large 8-bit color icon resource, and small 8-bit color icon resource) that visually represent your application and any documents it creates, and two related resources, the icon ('ICON') resource and the color icon ('cicn') resource

■ the file reference ('FREF') resource, which links icons with the files types they represent and which allows users to launch your application by dragging document icons to your application icon

■ a bundle ('BNDL') resource, which groups together your application's signature, icon list resource, and file reference resources

■ a missing-application name string—that is, a string ('STR ') resource—for your application's documents in order to display the name of your application if the user tries to open or print a document created by your application when your application is missing

■ an application-missing message string—that is, a string ('STR ') resource—in your application's documents in order to explain why the user can't open or print certain documents used by your application

■ the version ('vers') resource, so that users can easily find out the version of a file and, if applicable, the version of the superset of files to which the single file belongs

For information about using the 'SIZE' resource to support stationery pads, see "Supporting Stationery Pads" beginning on page 7-34.

This section describes the structures of these resources after they are compiled by the Rez resource compiler, available from APDA. If you are interested in creating the Rez input files for these resources, see instead "Using the Finder Interface" beginning on page 7-6 for detailed information.
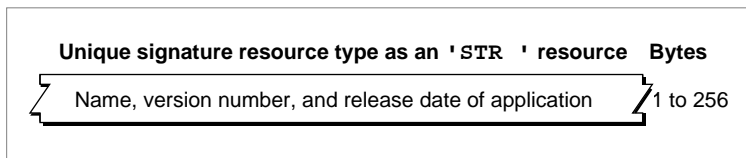
The Signature Resource

Every application that creates documents should define a signature resource, so that the Finder can identify and start up the application when a user double-clicks documents created by the application. A signature resource is typically defined to be a string resource (that is, a resource of type `'STR '`) that is given a unique four-character signature as its resource type. For example, an application with the signature of WAVE would use a string resource to define its signature resource as a resource of type `'WAVE'`. The signature resource should have a resource ID number of 0.

To ensure uniqueness, developers must register their applications' four-character signatures with Apple Computer, Inc., at Macintosh Developer Technical Support.

This section describes the structure of a signature resource defined to be of type `'STR '` after it's compiled by the Rez resource compiler. The format of a Rez input file for a signature resource differs from its compiled output form. If you are concerned only with creating a signature resource, see "Giving a Signature to Your Application and a Creator and a File Type to Your Documents" beginning on page 7-8.

If you examine a compiled version of a signature resource, as shown in Figure 7-16, you find that it contains a Pascal string that specifies the name, version number, and release date of the application.

Figure 7-16   Structure of a signature resource compiled as a string (`'STR '`) resource

Unique signature resource type as an `'STR '` resource   Bytes

Name, version number, and release date of application   1 to 256

If an application does not provide specific version information through a version resource (described in "Providing Version Resources" beginning on page 7-31), the Finder displays the string stored in the signature resource when the user selects the application and chooses Get Info from the File menu.

The Icon List Resource

An icon list resource is one of several icon resources that you create to represent visually for the user your application or one of the document types it creates. An icon list resource is a resource with the resource type `'ICN#'`. All icon list resources must be marked purgeable, and they must have resource IDs greater than 128.

When the user chooses by Icon from the View menu, the Finder displays the black-and-white icon specified in this resource in windows if either the user has a black-and-white monitor or your application has not defined any resources for color icons; otherwise, the Finder displays a color version of the icon.

An icon list resource is defined to be an array of two items of type `String[128]`; each bit in the first array represents a pixel in the 32-by-32 pixel icon, and each bit in the second array represents a pixel in the 32-by-32 pixel mask. You can use a high-level tool such as the ResEdit application, which is available through APDA, to create icon list resources. You can then use the DeRez decompiler to convert your icon list resources into Rez input when necessary. See "Creating Icons for the Finder" beginning on page 7-11 for additional information about creating icon list resources and other resources for representing files to users.

An icon list resource defines one icon, which the Finder uses to display the file it represents. If you examine the compiled version of an icon list resource, as represented in Figure 7-17, you find that it contains the following elements:

■ The 32-by-32 pixel black-and-white icon.

■ The 32-by-32 pixel black icon mask, which shows the area covered by the black-and-white icon and any 32-by-32 pixel color versions of the icon. The Finder uses the mask to crop the icon's outline into whatever background color or pattern is on the desktop. The Finder then draws the black-and-white icon specified in this resource—or the color icons specified in large 4-bit color icon resources or large 8-bit color icon resources—into this shape.

Figure 7-17   Structure of a compiled icon list (`'ICN#'`) resource
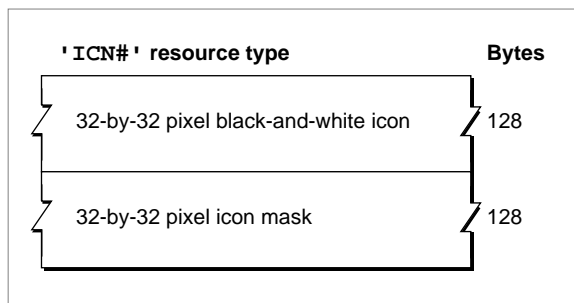
To create 16-by-16 pixel and color versions of the icon defined in an icon list resource (thereby supplying an entire icon family), your application must also create the following resources: a small icon list resource, a large 4-bit color icon resource, a small 4-bit color icon resource, a large 8-bit color icon resource, and a small 8-bit color icon resource. Their compiled formats are described in the next several sections; guidelines for creating them are provided in "Creating Icons for the Finder" beginning on page 7-11.

The Small Icon List Resource

A small icon list resource is one of several resources that you provide for an icon family. A small icon list resource is a resource with the resource type `'ics#'`. A small icon list resource must be marked purgeable, and it must have the same resource ID as the icon list resource that represents the file that the small icon list resource also represents.

When the user chooses by Small Icon from the View menu, the Finder displays the small black-and-white icon specified in this resource in windows if either the user has a black-and-white monitor or the application has not defined any resources for color icons; otherwise, a color version of the icon is displayed. Similarly, the small black-and-white icon or its color version appears in the Application menu after the user launches the application and in the Apple menu if the user places the application or an alias to it in the Apple Menu Items folder.

A small icon list resource is defined to be an array of two items of type `String[32]`; each bit in the first array represents a pixel in the 16-by-16 pixel icon, and each bit in the second array represents a pixel in the 16-by-16 pixel mask. You can use a high-level tool such as the ResEdit application to create small icon list resources. You can then use the DeRez decompiler to convert your small icon list resources into Rez input when necessary. See "Creating Icons for the Finder" beginning on page 7-11 for information about creating small icon list resources and other resources for representing files to users.

A small icon list resource defines one icon, which the Finder uses to display the file it represents. If you examine the compiled version of a small icon list resource, as represented in Figure 7-18, you find that it contains the following elements:

■ The 16-by-16 pixel black-and-white icon for display on the desktop.

■ The 16-by-16 pixel black icon mask, which shows the area covered by the icon. The Finder uses the mask to crop the icon's outline into whatever background color or pattern is on the desktop. The Finder then draws the black-and-white icon specified in this resource—or the color icons specified in the small 4-bit color icon resource or the small 8-bit color icon resource—into this shape.

The format for the compiled icon list resource is described on page 7-57; the format for the compiled small 4-bit color icon resource is described on page 7-60; and the format for the compiled small 8-bit color icon resource is described on page 7-62.

Figure 7-18   Structure of a compiled small icon list (`'ics#'`) resource
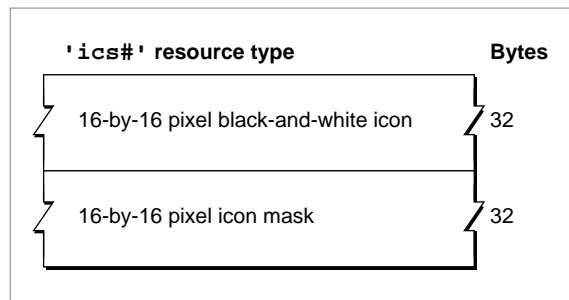
The Large 4-Bit Color Icon Resource

A large 4-bit color icon resource is one of several resources that you provide for an icon family. A large 4-bit color icon resource is a resource with the resource type `'icl4'`. A large 4-bit color icon resource must be marked purgeable, and it must have the same
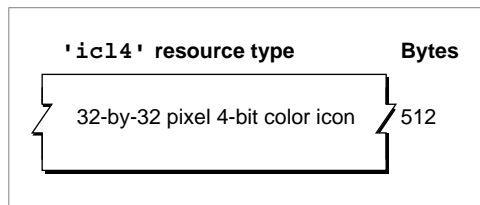
resource ID as the icon list resource that represents the file that the large 4-bit color icon resource also represents.

When the user chooses by Icon from the View menu, the Finder displays the large 4-bit color icon specified in this resource in windows if the user has a monitor displaying 4 bits of color data per pixel. Similarly, the large 4-bit color icon appears in the Application menu after the user launches the application and in the Apple menu if the user places the application or an alias to it in the Apple Menu Items folder.

A large 4-bit color icon resource is defined to be of type `String[512]`; every 4 bits in the string represent a pixel in the 32-by-32 pixel icon. You can use a high-level tool such as the ResEdit application to create large 4-bit color icon resources. You can then use the DeRez decompiler to convert your large 4-bit color icon resources into Rez input when necessary. See "Creating Icons for the Finder" beginning on page 7-11 for information about creating resources for visually representing files.

A large 4-bit color icon resource defines one icon, which the Finder uses to display the file it represents. If you examine the compiled version of a large 4-bit color icon resource, as represented in Figure 7-18, you find that it contains only the 32-by-32 pixel 4-bit color icon for display by the Finder. This resource does not specify a mask for the icon; instead, the Finder uses the mask specified for the icon list resource with the same resource ID number as this resource.

Figure 7-19   Structure of a compiled large 4-bit color icon (`'icl4'`) resource

The format for the compiled icon list resource is described on page 7-57.

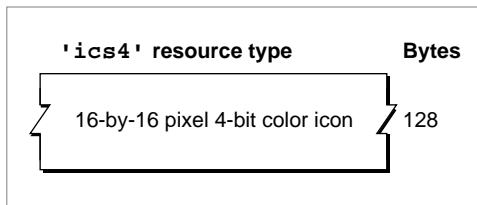## The Small 4-Bit Color Icon Resource

A small 4-bit color icon resource is one of several resources that you provide for an icon family. A small 4-bit color icon resource is a resource with the resource type `'ics4'`. A small 4-bit color icon resource must be marked purgeable, and it must have the same resource ID as the icon list resource that represents the file that the small 4-bit color icon resource also represents.

When the user chooses by Small Icon from the View menu, the Finder displays the small 4-bit color icon specified in this resource in windows if the user has a monitor displaying 4 bits of color data per pixel. Similarly, the small 4-bit color icon appears in the Application menu after the user launches the application and in the Apple menu if the user places the application or an alias to it in the Apple Menu Items folder.

A small 4-bit color icon resource is defined to be of type `String[128]`; every 4 bits in the string represent a pixel in the 16-by-16 pixel icon. You can use a high-level tool such as the ResEdit application to create small 4-bit color icon resources. You can then use the DeRez decompiler to convert your small 4-bit color icon resources into Rez input when necessary. See "Creating Icons for the Finder" beginning on page 7-11 for information about creating resources for representing files to users.

A small 4-bit color icon resource defines one icon, which the Finder uses to display the file it represents. If you examine the compiled version of a small 4-bit color icon resource, as represented in Figure 7-18, you find that it contains only the 16-by-16 pixel 4-bit color icon for display by the Finder. This resource does not specify a mask for the icon; instead, the Finder uses the mask specified for the small icon list resource with the same resource ID number as this resource.

Figure 7-20   Structure of a compiled small 4-bit color icon (`'ics4'`) resource

```
'ics4' resource type              Bytes

   16-by-16 pixel 4-bit color icon      128
```

The format for the compiled icon list resource is described on page 7-57. The format for the compiled small icon list resource is described on page 7-58.

The Large 8-Bit Color Icon Resource

A large 8-bit color icon resource is one of several resources that you provide for an icon family. A large 8-bit color icon resource is a resource with the resource type `'icl8'`. A large 8-bit color icon resource must be marked purgeable, and it must have the same resource ID as the icon list resource that represents the file that the large 8-bit color icon resource also represents.

When the user chooses by Icon from the View menu, the Finder displays the large 8-bit color icon specified in this resource in windows if the user has a monitor displaying 8 bits of color data per pixel. Similarly, the large 8-bit color icon appears in the Application menu after the user launches the application and in the Apple menu if the user places the application or an alias to it in the Apple Menu Items folder.

A large 8-bit color icon resource is defined to be of type `String[1024]`; every byte in the string represents a pixel in the 32-by-32 pixel icon. You can use a high-level tool such as the ResEdit application to create large 8-bit color icon resources. You can then use the DeRez decompiler to convert your large 8-bit color icon resources into Rez input when necessary. See "Creating Icons for the Finder" beginning on page 7-11 for information about creating resources for visually representing files.

A large 8-bit color icon resource defines one icon, which the Finder uses to display the file it represents. If you examine the compiled version of a large 8-bit color icon resource, as represented in Figure 7-21, you find that it contains only the 32-by-32 pixel 8-bit color icon for display by the Finder. This resource does not specify a mask for the icon; instead, the Finder uses the mask specified for the icon list resource with the same resource ID number as this resource.

The format for the compiled icon list resource is described on page 7-57.

Figure 7-21   Structure of a compiled large 8-bit color icon (`'icl8'`) resource

`'icl8'` resource type	Bytes
32-by-32 pixel 8-bit color icon	1024

The Small 8-Bit Color Icon Resource

A small 8-bit color icon resource is one of several resources that you provide for an icon family. A small 8-bit color icon resource is a resource with the resource type `'ics8'`. A small 8-bit color icon resource must be marked purgeable, and it must have the same resource ID as the icon list resource that represents the file that the small 8-bit color icon resource also represents.

When the user chooses by Small Icon from the View menu, the Finder displays the small 8-bit color icon specified in this resource in windows if the user has a monitor displaying 8 bits of color data per pixel. Similarly, the small 8-bit color icon appears in the Application menu after the user launches the application and in the Apple menu if the user places the application or an alias to it in the Apple Menu Items folder.
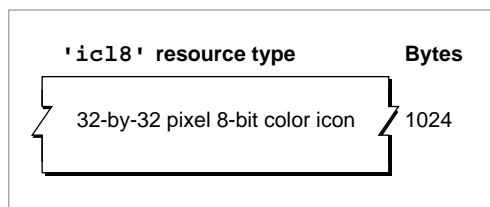
A small 8-bit color icon resource is defined to be of type `String[256]`; every byte in the string represents a pixel in the 16-by-16 pixel icon. You can use a high-level tool such as the ResEdit application to create small 8-bit color icon resources. You can then use the DeRez decompiler to convert your small 8-bit color icon resources into Rez input when necessary. See "Creating Icons for the Finder" beginning on page 7-11 for information about creating resources for visually representing files.

A small 8-bit color icon resource defines one icon, which the Finder uses to display the file it represents. If you examine the compiled version of a small 8-bit color icon resource, as represented in Figure 7-22, you find that it contains only the 16-by-16 pixel 8-bit color icon for display by the Finder. This resource does not specify a mask for the icon; instead, the Finder uses the mask specified for the small icon list resource with the same resource ID number as this resource.

Figure 7-22   Structure of a compiled small 8-bit color icon (`'ics8'`) resource
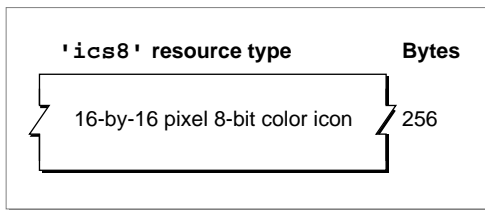
The format for the compiled icon list resource is described on page 7-57. The format for the compiled small icon list resource is described on page 7-58.
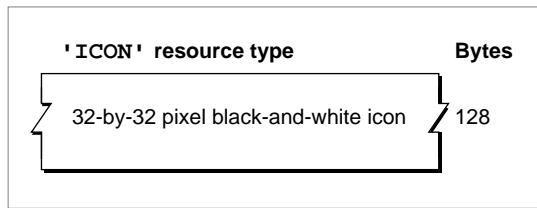
The Icon Resource

When you want to display a 32-by-32 pixel black-and-white icon within some element of your application (such as within a menu, an alert box, or a dialog box), you can create an icon resource. An icon resource is a resource with the resource type `'ICON'`. All icon resources must be marked purgeable, and they must have resource IDs greater than 128.

Using icon resources, you can create icons similar to the ones the Finder uses to display your application's files on the desktop; however, unlike the resource types previously described in this section, the Finder does *not* use or display any resources that you create of type `'ICON'`. Instead, your application uses icon resources of type `'ICON'` to display icons from within your application. Icon resources are described here for completeness and to mitigate the confusion that sometimes arises concerning icon (`'ICON'`) resources (which your application creates for its own use), icon list (`'ICN#'`) resources, and the other previously described resources necessary for defining an icon family (which your application creates for the Finder's use).

See "Creating Icons for the Finder" beginning on page 7-11 for additional information about creating icon list resources and other resources for representing files to users.

Generally, you use icon resources in menus and dialog boxes, as described in the chapters "Menu Manager" and "Dialog Manager" in this book. If you provide a color icon (`'cicn'`) resource with the same resource ID as the icon resource, the Menu Manager and the Dialog Manager display the color icons instead of the black-and-white icons for users with color monitors. (For example, the color alert box in Plate 2 specifies a resource of type `'cicn'` for the color icon in the upper-left corner of the alert box.)

An icon resource is defined to be of type `String[128]`; each bit represents a pixel in the 32-by-32 pixel icon. As illustrated in Figure 7-23 on the next page, an icon resource resembles an icon list resource without the array that specifies the icon's mask. You can use a high-level tool such as the ResEdit application to create icon resources. You can then use the DeRez decompiler to convert your icon resources into Rez input when necessary.

Figure 7-23   Structure of a compiled icon (`'ICON'`) resource

The Color Icon Resource

When you want to display a color icon within some element of your application (such as within a menu, an alert box, or a dialog box), you can create a color icon resource. A color icon resource is a resource with the resource type `'cicn'`. All color icon resources must be marked purgeable, and they must have resource IDs greater than 128.

Using color icon resources, you can create icons similar to the ones the Finder uses to display your application's files on the desktop; however, the Finder does *not* use or display any resources that you create of type `'cicn'`. Instead, your application uses icon resources of type `'cicn'` to display icons from within your application. Color icon resources (that is, those of resource type `'cicn'`) are mentioned here to mitigate the confusion that sometimes arises concerning color icon resources (which your application creates for its own use) and the small and large 4-bit and 8-bit color icon resources (types `'ics4'`, `'icl4'`, `'ics8'`, and `'icl8'`) necessary to define an icon family (which your application creates for the Finder's use).

See "Creating Icons for the Finder" beginning on page 7-11 for information about creating an icon family that includes color icons for representing files to users.

Generally, you use color icon resources in menus, alert boxes, and dialog boxes, as described in the chapters "Menu Manager" and "Dialog Manager" in this book. If you provide a color icon (`'cicn'`) resource with the same resource ID as an icon resource (described on page 7-63), the Menu Manager and the Dialog Manager display the color icon instead of the black-and-white icon for users with color monitors.You can use a high-level tool such as the ResEdit application to create color icon resources. You can then use the DeRez decompiler to convert your color icon resources into Rez input when necessary. (For example, the color alert box in Plate 2 specifies a resource of type `'cicn'` for the color icon in the upper-left corner of the alert box.)

See *Inside Macintosh: Imaging* for more information about color icon resources.

The File Reference Resource

To link icons with the files types they represent and to allow users to launch your application by dragging document icons to your application icon, create a file reference resource for every icon list resource you create. A file reference resource is a resource with the resource type `'FREF'`. All file reference resources must have resource IDs greater than 128, and each must be marked purgeable.

This section describes the structure of a file reference resource after it is compiled by the Rez resource compiler. The format of a Rez input file for a file reference resource differs from its compiled output form. If you are concerned only with creating a file reference resource, see "Creating File Reference Resources" beginning on page 7-18.

If you examine a compiled version of a file reference resource, as illustrated in Figure 7-24, you find that it contains the following elements:

■ File type. This is the four-character code that identifies the type of file represented by this resource. File types are described in "Giving a Signature to Your Application and a Creator and a File Type to Your Documents" beginning on page 7-8.

■ Local ID. The Finder uses this number to map the file type specified in this resource to an icon list resource that is assigned the same local ID in the bundle resource. The icon list resource is described on page 7-57; the bundle resource is described in the next section.

■ Empty string. This element should always contain an empty Pascal string.

Figure 7-24   Structure of a compiled file reference (`'FREF'`) resource

`'FREF'` resource type	Bytes
File type	2
Local ID for an icon list	4
Empty string	1

The Bundle Resource

To group together your application's signature, icon list resource, and file reference resources, create a bundle resource. A bundle resource is a resource with the resource type `'BNDL'`. All bundle resources must have resource ID numbers greater than 128, and all must be made purgeable.

This section describes the structure of the bundle resource after it is compiled by the Rez resource compiler. The format of a Rez input file for a bundle resource differs from its compiled output form. If you are concerned only with creating a bundle resource, see "Creating a Bundle Resource" beginning on page 7-20.

Figure 7-25   Structure of a compiled bundle ('BNDL') resource
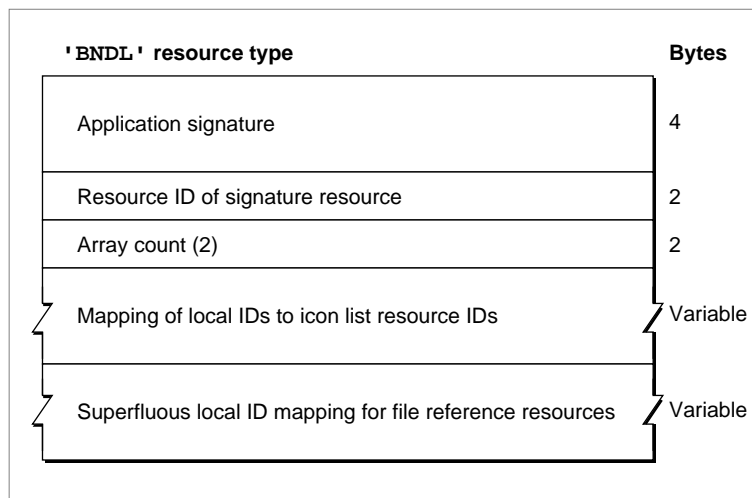
If you examine a compiled version of a file reference resource, as illustrated in
Figure 7-25, you find that it contains the following elements:

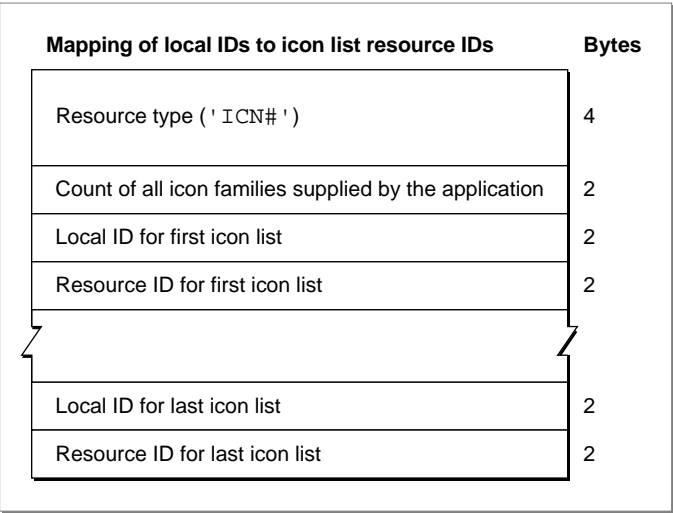- Application signature. This is the unique four-character code that identifies the
  application to the Finder. (Application signatures are described in "Giving a Signature
  to Your Application and a Creator and a File Type to Your Documents" beginning on
  page 7-8.)
- Resource ID of the signature resource. By convention, this should always be 0.
- Array count. This element should always contain the value 2.
- Mapping of local IDs to icon list resource IDs for all icons supplied by the application.
  This is illustrated in Figure 7-26.
- Superfluous local ID mapping for file reference resources. This is illustrated in
  Figure 7-27.

If you examine the compiled portion of a bundle resource that maps local IDs to icon list
resource IDs, you find that it contains the following elements:

- Resource type. This element should always specify the resource type 'ICN#' (that is,
  an icon list resource).
- Count of all the icon families supplied by the application. This is the number of local
  ID–to–icon list resource ID mapping pairs in the rest of this resource.
- Local ID for an icon list resource. This local ID must match the local ID assigned to the
  icon list resource within a file reference resource.
- Resource ID for the icon list resource assigned a local ID in the preceding element. To
  visually represent files of the type described in the file reference resource that contains
  the local ID in the preceding element, the Finder uses the black-and-white icon and
  mask described in this icon list resource. The Finder also uses the icons defined in the
  following resources with this same resource ID: small icon list resource, small 4-bit
  color icon resource, small 8-bit color icon resource, large 4-bit color icon resource, and
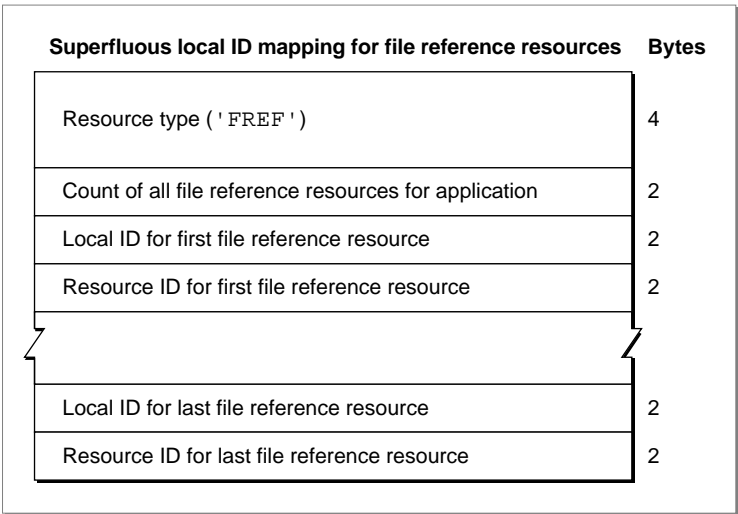 large 8-bit color icon resource.

Figure 7-26 Mapping local IDs to icon list resource IDs in a bundle resource

Mapping of local IDs to icon list resource IDs	Bytes
Resource type (`'ICN#'`)	4
Count of all icon families supplied by the application	2
Local ID for first icon list	2
Resource ID for first icon list	2
Local ID for last icon list	2
Resource ID for last icon list	2

- Local ID–to–icon list resource ID mapping pairs for the rest of the icons representing file types for an application.

Figure 7-27 illustrates the remainder of a bundle resource, which assigns local IDs to file reference resource IDs. This assignment is superfluous because the Finder doesn't map these local IDs to any other resources. This ID assignment was implemented for the earliest versions of Macintosh system software, and it remains this way today to maintain backward compatibility.

Figure 7-27 Structure of superfluous local ID mapping for file reference resources in a bundle resource

Superfluous local ID mapping for file reference resources	Bytes
Resource type (`'FREF'`)	4
Count of all file reference resources for application	2
Local ID for first file reference resource	2
Resource ID for first file reference resource	2
Local ID for last file reference resource	2
Resource ID for last file reference resource	2

If you examine the compiled portion of the remainder of a bundle resource, you find that it contains the following elements:

■ Resource type. This element should always specify the resource type 'FREF' (that is, a file reference resource).

■ Count of all the file reference resources representing file types for an application. This is the number of local ID–to–file reference resource mapping pairs in the rest of this resource.

■ Local ID for a file reference resource. The local ID can be any integer so long as no other file reference resource is given that same local ID within this resource.

■ Resource ID for the file reference resource assigned a local ID in the preceding field.

■ Local ID–to–file reference resource ID mapping pairs for the rest of the file reference resources that represent file types with application-supplied icons.

The Missing-Application Name String

When your application creates a document that the user can open, your application should include a missing-application name string in the resource file of the document. The missing-application name string is a resource with the resource type 'STR ', it must have a resource ID number of –16396, and it must be made purgeable. The string resource should contain your application's name only. See "Displaying Messages When the Finder Can't Find Your Application" beginning on page 7-27 for additional information about copying this resource into the resource fork of your documents.

If you examine a compiled missing-application name string, as illustrated in Figure 7-28, you find that it consists entirely of a Pascal string that names the application that created the document. The Finder displays this string in an alert box if the user tries to open or print a document created by the application whenever the application is missing.

Figure 7-28   Structure of a compiled missing-application name string resource

```
   'STR ' resource with resource ID –16396     Bytes

     Application name                          1 to 256
```
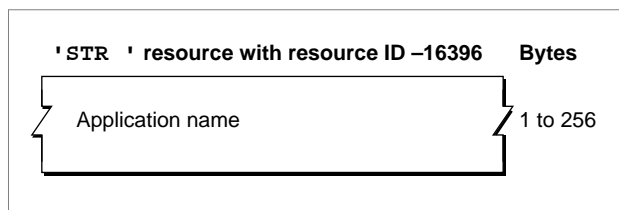
The Application-Missing Message String

When your application creates a document that your application uses but that the user cannot open (such as a preferences file), your application should set the creator of the document to a registered signature that is not the same as your or anyone else's application, and include an application-missing message string in the resource file of the document. The application-missing name string is a resource with the resource type

'STR ', it must have a resource ID number of –16397, and it must be made purgeable. The string resource should contain a message that explains why the user cannot open or print the document, as explained in "Displaying Messages When the Finder Can't Find Your Application" beginning on page 7-27.

If you examine a compiled application-missing message string, as illustrated in Figure 7-29, you find that it consists entirely of a Pascal string that explains why the user cannot open the document. The Finder displays this string in an alert box if the user tries to open or print a document that is given a special creator that is not used as a signature by any application file. (File creators and application signatures are explained in "Giving a Signature to Your Application and a Creator and a File Type to Your Documents" beginning on page 7-8.)

Figure 7-29   Structure of a compiled application-missing message string resource
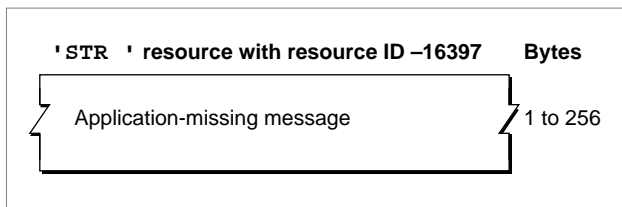

The Version Resource

You can use a version resource in any file so that users can easily find out the version of the file and, if it is a part of a larger collection of files, of the entire superset of files. A version resource is a resource with the resource type 'vers'. The version resource with a resource ID number of 1 specifies the version of an individual file; the version resource with a resource ID number of 2 specifies the superset of files to which the individual file belongs.

If your application does not contain a version resource with a resource ID number of 1, the Finder displays the string from your application's signature resource (described in "Giving a Signature to Your Application and a Creator and a File Type to Your Documents" beginning on page 7-8) in the information window when the user chooses the Get Info command from the File menu.

This section describes the structure of this resource after it is compiled by the Rez resource compiler. The format of a Rez input file for a version resource differs from its compiled output form. If you are concerned only with creating version resources, see "Providing Version Resources" beginning on page 7-31.

If you examine a compiled version of version resource, as illustrated in Figure 7-30 on page 7-70, you find that it contains the following elements:

■ Major revision level in binary-coded decimal format.

■ Minor revision level in binary-coded decimal format.

Figure 7-30   Format of a compiled version (`'vers'`) resource
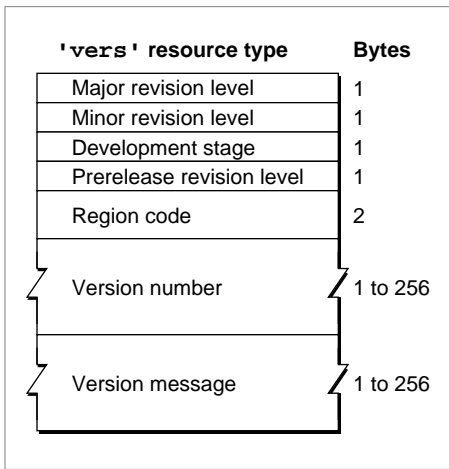
- Development stage. The values that can appear in this field, as well as the constants that can be used to specify them in a Rez input file, are the following:

Value	Constant	Description
0x20	development	Prealpha file
0x40	alpha	Alpha file
0x60	beta	Beta file
0x80	release	Released file

- Prerelease revision level. This number specifies the version if the software is still prerelease.

- Region code. This identifies the script system for which this version of the software is intended. See the chapter "Script Manager" in *Inside Macintosh: Text* for information about the values represented by the various region codes that can be specified here.

- Version number. This Pascal string identifies the version number of the software. When the user opens the Views control panel, clicks the Show version box, and then chooses any command from the View menu other than by Icon or by Small Icon, the Finder window containing this application displays this string.

- Version message. This Pascal string identifies the version number and either a company copyright for a file or a product name for a superset of files. When the user selects this file and chooses the Get Info command, the Finder displays this string in the information window as follows:

  □ For a version resource with a resource ID number of 1, this string is displayed in the version field of the information window.

  □ For a version resource with a resource ID number of 2, this string is displayed beneath the file's name next to the file's icon at the top of the information window.

Summary of the Finder Interface

Pascal Summary

Constants

```
CONST {Gestalt selectors}
      gestaltFindFolderAttr        = 'fold';    {selector for FindFolder}

      {interpreting Gestalt selector responses}
      gestaltFindFolderPresent     = 0;            {if this bit is set, }
                                                   { FindFolder is present}
      {for custom icons}
      kCustomIconResource          = –16455;       {resource ID for }
                                                   { custom icon}
      {for Finder flags}
      fHasBundle                   = 8192;         {set if file has 'BNDL'}
      fInvisible                   = 16384;        {set if icon is invisible}
      kIsOnDesk                    = $1;           {unused and reserved in }
                                                   { System 7}
      kColor                       = $E;           {three bits of color coding}
      kIsShared                    = $40;          {file can be executed by }
                                                   { multiple users }
                                                   { simultaneously}
      kHasBeenInited               = $100;         {file info is in desktop }
                                                   { database}
      kHasCustomIcon               = $400;         {file or directory has a }
                                                   { customized icon}
      kIsStationery                = $800;         {file is a stationery pad}
      kNameLocked                  = $1000;        {file or directory can't }
                                                   { be renamed from Finder, }
                                                   { and icon can't be changed}
      kHasBundle                   = $2000;        {file has a bundle resource}
      kIsInvisible                 = $4000;        {file or directory is }
                                                   { invisible from Finder & }
                                                   { from Standard File }
                                                   { Package dialog boxes}
      kIsAlias                     = $8000;        {file is an alias file}
```

```
{for FindFolder}
kOnSystemDisk                     = $8000;     {use vRefNum for the }
                                              { boot disk}
kCreateFolder                     = TRUE;      {create folder if it }
                                              { doesn't exist}
kDontCreateFolder                 = FALSE;     {don't create folder}

{for special folder types}
kSystemFolderType                 = 'macs';    {System Folder}
kDesktopFolderType                = 'desk';    {Desktop Folder}
kTrashFolderType                  = 'trsh';    {single-user Trash}
kWhereToEmptyTrashFolderType      = 'empt';    {shared Trash on network}
kPrintMonitorDocsFolderType       = 'prnt';    {PrintMonitor Documents}
kStartupFolderType                = 'strt';    {Startup Items}
kFontsFolderType                  = 'font';    {Fonts}
kAppleMenuFolderType              = 'amnu';    {Apple Menu Items}
kControlPanelFolderType           = 'ctrl';    {Control Panels}
kExtensionFolderType              = 'extn';    {Extensions}
kPreferencesFolderType            = 'pref';    {Preferences}
kTemporaryFolderType              = 'temp';    {Temporary Items}
{alias types}
kContainerFolderAliasType         = 'fdrp';    {folder alias}
kContainerTrashAliasType          = 'trsh';    {Trash alias}
kContainerHardDiskAliasType       = 'hdsk';    {hard disk alias}
kContainerFloppyAliasType         = 'flpy';    {floppy disk alias}
kContainerServerAliasType         = 'srvr';    {server alias}
kApplicationAliasType             = 'adrp';    {application alias}
kContainerAliasType               = 'drop';    {all other containers}
kSystemFolderAliasType            = 'fasy';    {System Folder alias}
kAppleMenuFolderAliasType         = 'faam';    {Apple Menu Items folder }
                                              { alias}
kStartupFolderAliasType           = 'fast';    {Startup Items folder alias}
kPrintMonitorDocsFolderAliasType
                                  = 'fapn';    {PrintMonitor Documents }
                                              { folder alias}
kPreferencesFolderAliasType       = 'fapf';    {Preferences folder alias}
kControlPanelFolderAliasType      = 'fact';    {Control Panels folder alias}
kExtensionFolderAliasType         = 'faex';    {Extensions folder alias}
kExportedFolderAliasType          = 'faet';    {export folder alias}
kDropFolderAliasType              = 'fadr';    {drop folder alias}
kSharedFolderAliasType            = 'fash';    {shared folder alias}
kMountedFolderAliasType           = 'famn';    {mounted folder alias}
```

Data Types

```
TYPE   {Finder information records in the volume catalog file}
       FInfo =
       RECORD
          fdType:        OSType;        {file type}
          fdCreator:     OSType;        {file creator}
          fdFlags:       Integer;       {Finder flags}
          fdLocation:    Point;         {file's location in window}
          fdFldr:        Integer;       {directory that contains file}
       END;

       FXInfo =
       RECORD
          fdIconID:      Integer;       {icon ID}
          fdUnused:      ARRAY[1..3] OF Integer;
                                        {unused but reserved 6 bytes}
          fdScript:      SignedByte;    {script flag and code}
          fdXFlags:      SignedByte;    {reserved}
          fdComment:     Integer;       {comment ID}
          fdPutAway:     LongInt;       {home directory ID}
       END;

       DInfo =
       RECORD
          frRect:        Rect;          {folder's window rectangle}
          frFlags:       Integer;       {flags}
          frLocation:    Point;         {folder's location in window}
          frView:        Integer;       {folder's view}
       END;

       DXInfo =
       RECORD
          frScroll:      Point;         {scroll position}
          frOpenChain:   LongInt;       {dir ID chain of open folders}
          frScript:      SignedByte;    {script flag and code}
          frXFlags:      SignedByte;    {reserved}
          frComment:     Integer;       {comment ID}
          frPutAway:     LongInt;       {directory ID}
       END;
```

Routines

Resolving Alias Files

```
FUNCTION ResolveAliasFile    (VAR theSpec: FSSpec;
                              resolveAliasChains: Boolean;
                              VAR targetIsFolder: Boolean;
                              VAR wasAliased: Boolean): OSErr;
```

Finding Directories

```
FUNCTION FindFolder          (vRefNum: Integer; folderType: OSType;
                              createFolder: Boolean;
                              VAR foundVRefNum: Integer;
                              VAR foundDirID: LongInt): OSErr;
```

C Summary

Constants

```
enum {
      /*Gestalt selectors*/
      #define gestaltFindFolderAttr  'fold'     /*selector for FindFolder*/

      /*interpreting Gestalt selector responses*/
      gestaltFindFolderPresent      = 0          /*if this bit is set, */
                                                 /* FindFolder is present*/
};
      /*for custom icons*/
#define kCustomIconResource           -16455     /*resource ID for */
                                                 /* custom icon*/


      /*Finder flags*/
#define  kIsOnDesk                   0x1         /*unused and reserved in */
                                                 /* System 7*/
#define  kColor                      0xE         /*3 bits of color coding*/
#define  kIsShared                   0x40        /*file can be executed by */
                                                 /* multiple users */
                                                 /* simultaneously*/
#define  kHasBeenInited              0x100       /*file info is in desktop */
                                                 /* database*/
#define  kHasCustomIcon              0x400       /*file or directory has a */
                                                 /* customized icon*/
```

```
#define  kIsStationary              0x800      /*file is a stationery pad*/
#define  kNameLocked                0x1000     /*file or directory can't */
                                               /* be renamed from the */
                                               /* Finder, and icon can't */
                                               /* be changed*/
#define  kHasBundle                 0x2000     /*file has a bundle */
                                               /* resource*/
#define  kIsInvisible               0x4000     /*file or directory is */
                                               /* invisible from Finder */
                                               /* & from Standard File */
                                               /* Package dialog boxes*/
#define  kIsAlias                   0x8000     /*file is an alias file*/

enum {
     /*for Finder flags*/
     fHasBundle                 = 8192,     /*set if file has 'BNDL'*/
     fInvisible                 = 16384     /*set if icon is invisible*/
};
enum {
     /*for FindFolder*/
     kOnSystemDisk              = 0x8000    /*use vRefNum for the */
                                            /* boot disk*/
     #define kCreateFolder          true    /*create folder if it */
                                            /* doesn't exist*/
     #define kDontCreateFolder      false   /*don't create folder*/

     /*for special folder types*/
     #define kSystemFolderType      'macs'   /*System Folder*/
     #define kDesktopFolderType     'desk'   /*Desktop Folder*/
     #define kTrashFolderType       'trsh'   /*single-user Trash*/
     #define kWhereToEmptyTrashFolderType
                                     'empt'   /*shared Trash*/
     #define kPrintMonitorDocsFolderType
                                     'prnt'   /*PrintMonitor Documents*/
     #define kStartupFolderType     'strt'   /*Startup Items*/
     #define kFontsFolderType       'font'   /*Fonts*/
     #define kAppleMenuFolderType   'amnu'   /*Apple Menu Items*/
     #define kControlPanelFolderType 'ctrl'  /*Control Panels*/
     #define kExtensionFolderType   'extn'   /*Extensions*/
     #define kPreferencesFolderType 'pref'   /*Preferences*/
     #define kTemporaryFolderType   'temp'   /*Temporary Items*/
};
     /*for alias types*/
#define kContainerFolderAliasType    'fdrp'   /*folder alias*/
#define kContainerTrashAliasType     'trsh'   /*Trash alias*/
```

```
#define kContainerHardDiskAliasType    'hdsk'    /*hard disk alias*/
#define kContainerFloppyAliasType      'flpy'    /*floppy disk alias*/
#define kContainerServerAliasType      'srvr'    /*server alias*/
#define kApplicationAliasType          'adrp'    /*application alias*/
#define kContainerAliasType            'drop'    /*all other containers*/
#define kSystemFolderAliasType         'fasy'    /*System Folder alias*/
#define kAppleMenuFolderAliasType      'faam'    /*Apple Menu Items folder */
                                                 /* alias*/
#define kStartupFolderAliasType        'fast'    /*Startup Items folder */
                                                 /* alias*/
#define kPrintMonitorDocsFolderAliasType
                                       'fapn'    /*PrintMonitor Documents */
                                                 /* folder alias*/
#define kPreferencesFolderAliasType    'fapf'    /*Preferences folder alias*/
#define kControlPanelFolderAliasType   'fact'    /*Control Panels fldr alias*/
#define kExtensionFolderAliasType      'faex'    /*Extensions folder alias*/
#define kExportedFolderAliasType       'faet'    /*export folder alias*/
#define kDropFolderAliasType           'fadr'    /*drop folder alias*/
#define kSharedFolderAliasType         'fash'    /*shared folder alias*/
#define kMountedFolderAliasType        'famn'    /*mounted folder alias*/
```

Data Types

```
struct FInfo {          /*Finder information records in the catalog file*/
      OSType            fdType;        /*file type*/
      OSType            fdCreator;     /*file creator*/
      unsigned short    fdFlags;       /*Finder flags*/
      Point             fdLocation;    /*file's location in window*/
      short             fdFldr;        /*directory that contains file*/
};

struct FXInfo {
      short             fdIconID;      /*icon ID*/
      short             fdUnused[3];   /*unused but reserved 6 bytes*/
      char              fdScript;      /*script flag and code*/
      char              fdXFlags;      /*reserved*/
      short             fdComment;     /*comment ID*/
      long              fdPutAway;     /*home directory ID*/
};
```

```
struct DInfo {
    Rect              frRect;        /*folder's window rectangle*/
    unsigned short    frFlags;       /*flags*/
    Point             frLocation;    /*folder's location in window*/
    short             frView;        /*folder's view*/
};

struct DXInfo {
    Point     frScroll;        /*scroll position*/
    long      frOpenChain;     /*directory ID chain of open folders*/
    char      frScript;        /*script flag and code*/
    char      frXFlags;        /*reserved*/
    short     frComment;       /*comment ID*/
    long      frPutAway;       /*directory ID*/
};
```

Routines

Resolving Alias Files

```
pascal OSErr ResolveAliasFile
                        (FSSpec *theSpec, Boolean resolveAliasChains,
                         Boolean *targetIsFolder, Boolean *wasAliased);
```

Finding Directories

```
pascal OSErr FindFolder      (short vRefNum, OSType folderType,
                              Boolean createFolder, short *foundVRefNum,
                              long *foundDirID);
```

Assembly-Language Summary

Data Structures

FInfo Data Structure

0	fdType	long	file type
4	fdCreator	long	file creator
8	fdFlags	word	Finder flags
10	fdLocation	long	file's location in window
14	fdFldr	word	directory that contains file

FXInfo Data Structure

0	fdIconID	word	icon ID
2	fdUnused	6 bytes	reserved
8	fdScript	1 byte	script flag and code
9	fdXFlags	1 byte	reserved
10	fdComment	word	comment ID
12	fdPutAway	long	home directory ID

DInfo Data Structure

0	frRect	8 bytes	folder's window rectangle
8	frFlags	word	flags
10	frLocation	long	folder's location in window
14	frView	word	folder's view

DXInfo Data Structure

0	frScroll	long	scroll position
4	frOpenChain	long	directory ID chain of open folders
8	frScript	1 byte	script flag and code
9	frXFlags	1 byte	reserved
10	frComment	word	comment ID
12	frPutAway	long	directory ID

Result Codes

noErr	0	No error
nsvErr	−35	Volume not found
fnfErr	−43	For FindFolder: Type not found in 'fld#' resource, or disk doesn't have System Folder support or System Folder in volume header, or disk does not have desktop database support for Desktop Folder—in all cases, folder not found
		For ResolveAliasFile: Target not found, but volume and parent directory found and theSpec parameter contains a valid file system specification record
dupFNErr	−48	File found instead of folder
dirNFErr	−120	Parent directory not found