9

THE FINDER AND THE APPLICATION

Introduction

The Finder

The **Finder** is an application that works with the system software to, amongst other things, mediate the user's access to volumes, folders and files in the file system, present a visual representation of those volumes, folders and files to the user on the desktop, and generally manage the **desktop**. The term "desktop" in this context simply means the work area on the screen.

The Finder requires that your application supply it with certain information. The Finder accesses this information and builds its own database of the resources it needs, such as the icons to use to display your application and the documents it creates.

The Finder uses certain high-level events known as **Apple events** to communicate certain instructions and information to your application. Accordingly, this chapter should be read in conjunction with Chapter 10.

Relevant Databases

The Desktop Database — Mac OS 8/9

When a file is created or installed, the File Manager extracts some of the information provided by resources described in this chapter and stores it in the volume's **catalog file**, a special file containing information about the hierarchical organisation of files and folders on that volume.

Although it is mostly used by the File Manager, the information in the catalog file is also used by the Mac OS 8/9 Finder. The Finder extracts the information provided by your application and uses it to build a **desktop database**. The purpose of the desktop database is to facilitate rapid access by the Finder to icons, file types, applications, version data, and comments. The Finder builds the desktop databases for each mounted volume at system start-up, and updates them as files and directories are added, moved, renamed or deleted.

The desktop database:

- Contains the definitions of all icons and their associated file types (see below).

- Lists all the file types that an application can open.

- Lists the location of each application on the disk.

- Contains any comments that the user has added to the information windows (the window opened when the user selects an icon and chooses Get Info from the Mac OS 8/9 Finder's File menu) for desktop objects.

Mac OS X maintains a number of private databases, one of which is the **application database**. Because of the multi-user nature of Mac OS X, the Finder maintains an application database for each user with an account on the system. This database contains information about all the applications the Mac OS X Finder has encountered for that user and includes information about the document types understood by each application.

If your application includes a 'plst' (property list) resource which itself includes a binary representation of an **information property list** containing the necessary information, the Mac OS X Finder extracts the information it requires from that list. If the 'plst' resource is empty, the Mac OS X Finder extracts the information provided by the signature, file reference, bundle, and version resources described in this chapter.

> ### Note
>
> As will be seen, an application's 'plst' resource must contain an information property list in order for it to specify large (128 by 128 pixel) thumbnail icons for itself and its documents. Since this will ordinarily be the requirement, the following assumes that the application includes an information property list containing all relevant information in its 'plst' resource.
>
> Note also that, in so-called "bundled" Mac OS X applications, information property lists are provided in files, not in 'plst' resources. However, the Mac OS X application packaging scheme known as the bundle is not addressed in this book.

The Mac OS 8/9 Finder ignores 'plst' resources.

An Application's Required Resources

The Mac OS 8/9 Finder requires that your application provide the following:

- A **signature resource**, which enables the Finder to identify and start up your application when a user double clicks a document created by your application.

- **Icon resources**, which represent your application and any documents it creates to the user.

- **File reference resources**, which link icons with the file types they represent.

- A **bundle resource**, which groups signature, icon and file reference resources together.

- A **'SIZE' resource** (see Chapter 2).

Other resources which should ideally be provided by your application for Mac OS 8/9 are:

- **Version resources**, which allow users to ascertain the version of your application.

- A **help resource**, which is used by the Finder to display your application's customised balloon help message.

Assuming that the application's 'plst' resource contains the necessary information, Mac OS X does not utilise the signature, file reference, bundle, or version resources. The help resource is irrelevant on Mac OS X.

Application Signature, File Creator, and File Types

Application Signature

Your application's **signature** is a unique four-character sequence, which must not conflict with that of any other application, and which the Finder uses to identify your application[1].

---

[1] To ensure uniqueness, developers are required to register their application's signature with Apple Computer, Inc, at Macintosh Developer Technical Support

To provide its signature, your application must include a special resource whose resource type is the application's signature and whose resource ID is 0.

Creating an Application Signature Resource Using Resorcerer

Resorcerer allows you to create an application signature resource within its bundle resource editing window (see Fig 5). Fig 1 shows an application signature resource being created using Resorcerer.
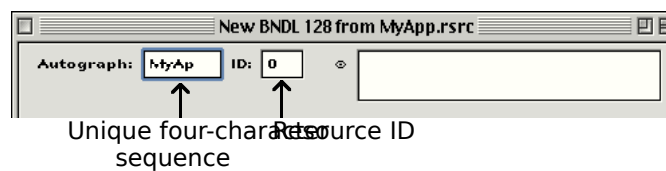

Unique four-character sequence

Resource ID

FIG 1 - CREATING A SIGNATURE RESOURCE USING

File Creator and File Type

When an application creates a document, it assigns that document a **creator** and a **file type**. Your application should set its own signature as the creator of the documents it creates. The creator field of the application file itself should contain the application's own signature.)

Use of the Creator by the Finder

The Finder reads the creator field of a document file when the user double clicks on the file's icon or selects it and chooses Open from the Finder's File menu (or, on Mac OS 8/9 only Print from the Finder's File menu). Having obtained the creator, the Finder then searches for an application with a signature of that name and, if it finds the application, calls the Process Manager to start it up. The Finder then passes to your application the information it needs to open or print the document via an Apple event.[2]

File Type

The file type can be a type especially defined for your application[3] or it can be one of the existing general types, some of which are as follows:

File Type	Description
'APPL'	Launchable application.
'DRVR'	Driver.
'FFIL'	File for storing fonts.
'PICT'	QuickDraw picture.
'PRER'	Printer driver.
'TEXT'	Stream of ASCII characters.
'appe'	Background only application.
'ffil'	Font.
'pref'	Preferences file.
'sfil'	Sound.
'ttro'	SimpleText read-only file.

Your application's file type must be 'APPL'.

---

[2]   Note that in this and other references to your application receiving requests to open and print documents, the assumption is made that your application supports the receipt and handling of required Apple events (see Chapter 10 — Apple Events).

[3]   Apple reserves the use of all signatures and file types whose names contain only lowercase and non-alphabetic characters. Your signature and file type created especially for your application must each contain at least one uppercase character. Like signatures, file types must be registered with Apple.

Use of the File Type by the Finder

Another way for a user to open a document created by your application, as well as a document not created by your application but which is of a file type *supported* by your application, is to select its icon and drag the selected icon to your application's icon. If the document's file type is supported by your application, the Finder calls the Process Manager to launch your application and then sends your application a required Apple event containing the information it needs to open the document.

Use of File Type by Your Application

Your application also relies on file types to determine which files to allow the user to open when your application is running. When your application calls Navigation Services 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 dialog then displays files of the specified types only. (See Chapter 18.)

Icons for the Finder

If you do not supply your own icons, the Finder uses its own default application and document icons for display. Fig 2 shows some of the Finder's default icons.

APPLICATION   DOCUMENT   PREFERENCE

FIG 2 - DEFAULT ICONS

Although default icons are available, you should ordinarily design and create your own icons for all the file types associated with your application, such as:

- The application file.

- Documents created by your application.

- Stationery pads created by users from your application's documents. (Stationery pads are files the user creates to serve as templates for other documents.)

The Icon Family

An **icon family** is a collection of icons intended to represent a single file type. The provision of a family of icon types, rather than just one icon type, enables the Finder to automatically select the appropriate family member to display depending on the icon size specified by the user and the bit depth of the display device.

Prior to Mac OS 8.5, the following icons comprised the icon family for a single file:

Icon	Size (Pixels)	Resource in Which Defined
Large black-and-white icon (1-bit) and mask	32 by 32	Icon list ('ICN#').
Small black-and-white icon (1-bit) and mask	16 by 16	Small icon list ('ics#')
Mini black-and-white icon (1-bit) and mask	12 by 16	Mini icon list ('icm#')
Large colour icon with 4 bits of colour data per pixel	32 by 32	Large 4-bit colour icon ('icl4')
Small colour icon with 4 bits of colour data per pixel	16 by 16	Small 4-bit colour icon ('ics4')
Mini colour icon with 4 bits of colour data per pixel	12 by 16	Mini 4-bit colour icon ('icm4')
Large colour icon with 8 bits of colour data per pixel	32 by 32	Large 8-bit colour icon ('icl8')
Small colour icon with 8 bits of colour data per pixel	16 by 16	Small 8-bit colour icon ('ics8')
Mini colour icon with 8 bits of colour data per pixel	12 by 16	Mini 8-bit colour icon ('icm8')

All of these icons use 1-bit masks, which are stored with the 'ICN#', 'ics#', and 'icm#' resources. The Finder uses the mask to crop the icon's outline into the background and then draws the icon into this shape. For this reason, the mask must be exactly the same shape as the icon (see Fig 3).

Note that the term "icon list" applying to the names of the black-and-white icon resources is somewhat of a misnomer in that you can define only two images in these resources: the icon and its mask.

The mini icon is of doubtful utility in the Carbon era and will not be considered further.

Huge Icons, 32-Bit Icons, and Deep Masks

Mac OS 8.5 introduced the following:

- Huge (48 by 48 pixels) icons ('ich#', 'ich4', 'ich8', and 'ih32').

- 32-bit small, large, and huge icons ('is32', 'il32', and 'ih32').

- 8-bit masks for the 32-bit icons ('s8mk', 'l8mk', and 'h8mk'). These "deep" masks support 256 different levels of transparency.

The bundle resource, however, does not support huge icons, 32-bit icons or 8-bit masks. On Mac OS 8/9, therefore, you cannot specify these icons for your application or its documents. You can, however include them in an 'icns' resource (see below) for Mac OS X.

Thumbnail Icons, the 'icns' Resource, and Mac OS X

For Mac OS X, you should provide 128 by 128 pixel, 32-bit, **thumbnail** icons ('it32') icons. The associated mask ('t8mk') depth is 8-bits.

This icon should be included, along with other required icons, in an 'icns' resource. (The 'icns' resource was introduced with Mac OS 8.5 as a means of providing a single source for all icon data. Combining all icon data into a single resource type speeds up icon fetching and simplifies resource management.) The ID of the 'icns' resource must be included in the information property list in your application's 'plst' resource.

Mac OS X can generate icons of various sizes by scaling the thumbnail. Nevertheless, if you need to preserve fine details at smaller resolutions, you may provide huge, large, and small icons, in addition to thumbnail icons, in your 'icns' resources. In this case, the system will use the best available icon for the display size required.
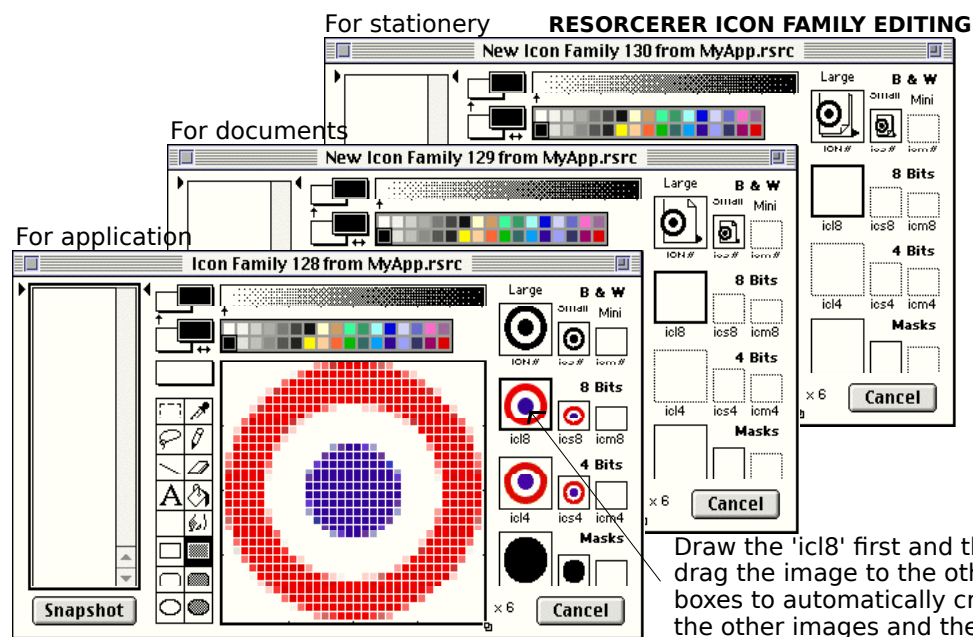
Creating Icon Resources

Unless you are using Resorcerer Version 2.4 or later, it will not be possible to use that application to create huge icons, any 32-bit icons, the thumbnail icons required for Mac OS X, or 'icns' resources. In these circumstances, a reasonable methodology is to:

- Use Resorcerer to create the icons that will be used when the application runs on Mac OS 8/9.

- Use the shareware program Iconographer to create the icons and 'icns' resource that will be used when the application runs on Mac OS X.

Creation Using Resorcerer

Fig 3 shows large and small 1-bit, 4-bit, and 8-bit icons being created in Resorcerer.

FIG 3 - THREE ICON FAMILIES BEING CREATED USING RES

Creation Using Iconographer

Fig 4 shows an application's icons being created using Iconographer. You can copy the 'icns' resource in the resource fork of the file created by Iconographer to your application's resource file.
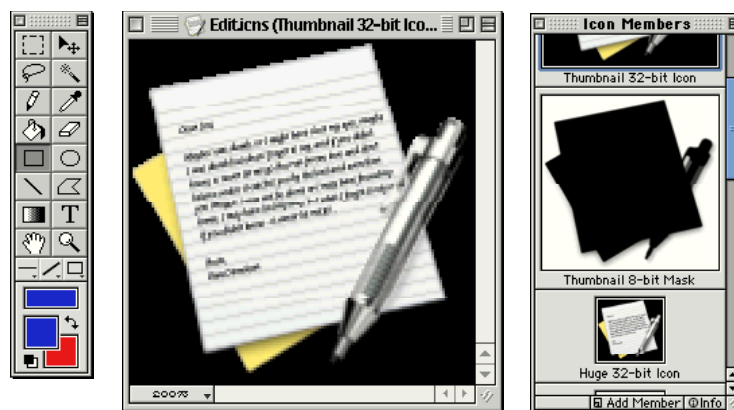
FIG 4 - AN APPLICATION'S ICON BEING CREATED USIN

Icon Design – Small and Large Icons

For the small and large icons used by Mac OS 8/9, good icon design requires that one graphic element be repeated in all icons created for the application. This allows the user to quickly identify the files associated with your product.

Icon Design — Thumbnail Icons

The design philosophy for thumbnail icons is significantly different from that applying to icons for Mac OS 8/9. The large size, bit-depth and transparency characteristics lend themselves to the creation of icons with photographic realism and perspective effects, making the icon appear as if it is an object on a desk as viewed from a chair.

An icon for an application should contain a base object (such as, for example, a sheet of water colour paper containing a half-finished painting) and a supportive "tool" (such as a paint brush) which communicates the type of task that the application allows the user to accomplish. The effect should be such the icon is perceived by the user as a familiar object viewed in a familiar way (that is, with perspective).

File Reference ('FREF') Resources

File reference ('FREF') resources associate your application's icons with file types created and supported by your application, allowing users to open documents supported by your application by dragging their icons to your application's icon.

You create separate 'FREF' resource for your application file and for each file type that your application can open.

Structure of the 'FREF' Resource and Creating a 'FREF' Resource Using Resorcerer

Fig 5 shows the structure of a 'FREF' resource and a 'FREF' resource being created using Resorcerer. The following describes the main fields of a compiled 'FREF' resource:

Field	Description
FILE TYPE	A four-character code identifying the type of file represented by this resource.
LOCAL ID FOR 'ICN#' RESOURCE	Used by the Finder to map the file type specified in this resource to an icon list ('ICN#') resource with the same local ID in the bundle ('BNDL') resource.
EMPTY STRING	(Not implemented.)

STRUCTURE OF A COMPILED 'FREF' RESOURCE

FILE TYPE

LOCAL ID FOR 'ICN#' RESOURCE

EMPTY STRING

File type - stationery.
Maps to 'ICN#' resource with local ID 2 in bundle resource (see Fig 7).

FREF 210 from MyApp.rsrc

File Type: sEXT   Local ICN# ID: 2

Description:

File type - document.
Maps to 'ICN#' resource with local ID 1 in bundle resource (see Fig 7).

FREF 209 from MyApp.rsrc

File Type: TEXT   Local ICN# ID: 1   ncel

Description:

File type - application.
Maps to 'ICN#' resource with local ID 0 in bundle resource (see Fig 7).

FREF 208 from MyApp.rsrc

File Type: APPL   Local ICN# ID: 0   ncel

Description: Not implemented. Leave empty.

For the stationery file type, use the file type of the document upon which the stationery is based but replace the first letter of the file type with a lowercase s, as shown in this example.

Cancel

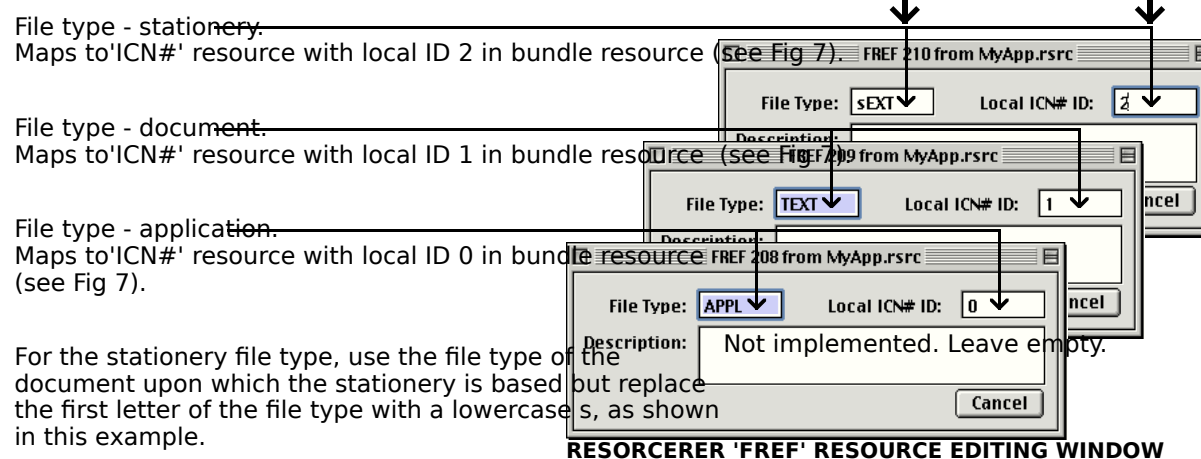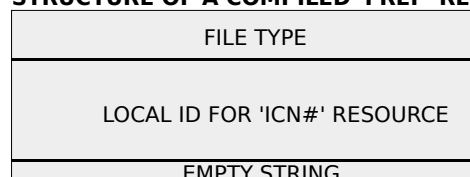**RESORCERER 'FREF' RESOURCE EDITING WINDOW**

FIG 5 - STRUCTURE OF A COMPILED 'FREF' RESOURCE AND CREATING A 'FREF' RESO

Use of 'FREF' Resources by the Finder

The Finder maintains a list of your 'FREF' resources. When, for example, the user drags a document icon to your application icon, the Finder checks the list and, if the document's file type appears in this list, launches your application with a request, passed via a required Apple event, to open that document.

You can define 'FREF' resources for file types your application supports but for which it does not provide icons. This allows users to launch your application by dragging the icons representing documents of this type to your application icon. For example, you could create a 'FREF' resource for the document file type 'ttro', which is used by documents which have SimpleText as their creator. (The Finder uses SimpleText's icon family resources for these documents.) The user can then open these documents in your application by dragging the document icon to your application's icon.

The Bundle ('BNDL') Resource

A bundle ('BNDL') resource associates all of the resources (signature, icons, and file reference) used by the Finder for your application.

When the Finder displays your application on the desktop for the first time, it determines whether the application has a 'BNDL' resource by checking the catalog file. If the Finder finds a 'BNDL' resource, it installs the information from the 'BNDL' resource and its associated resources into the Mac OS 8/9 desktop database and uses that information to display icons for your application's file types. If the Finder does not find a 'BNDL' resource, it displays your application using the default icons.

Structure of the 'BNDL' Resource and Creating a 'BNDL' Resource Using Resorcerer

Fig 6 shows the structure of a 'BNDL' resource and a 'BNDL' resource being created using Resorcerer.
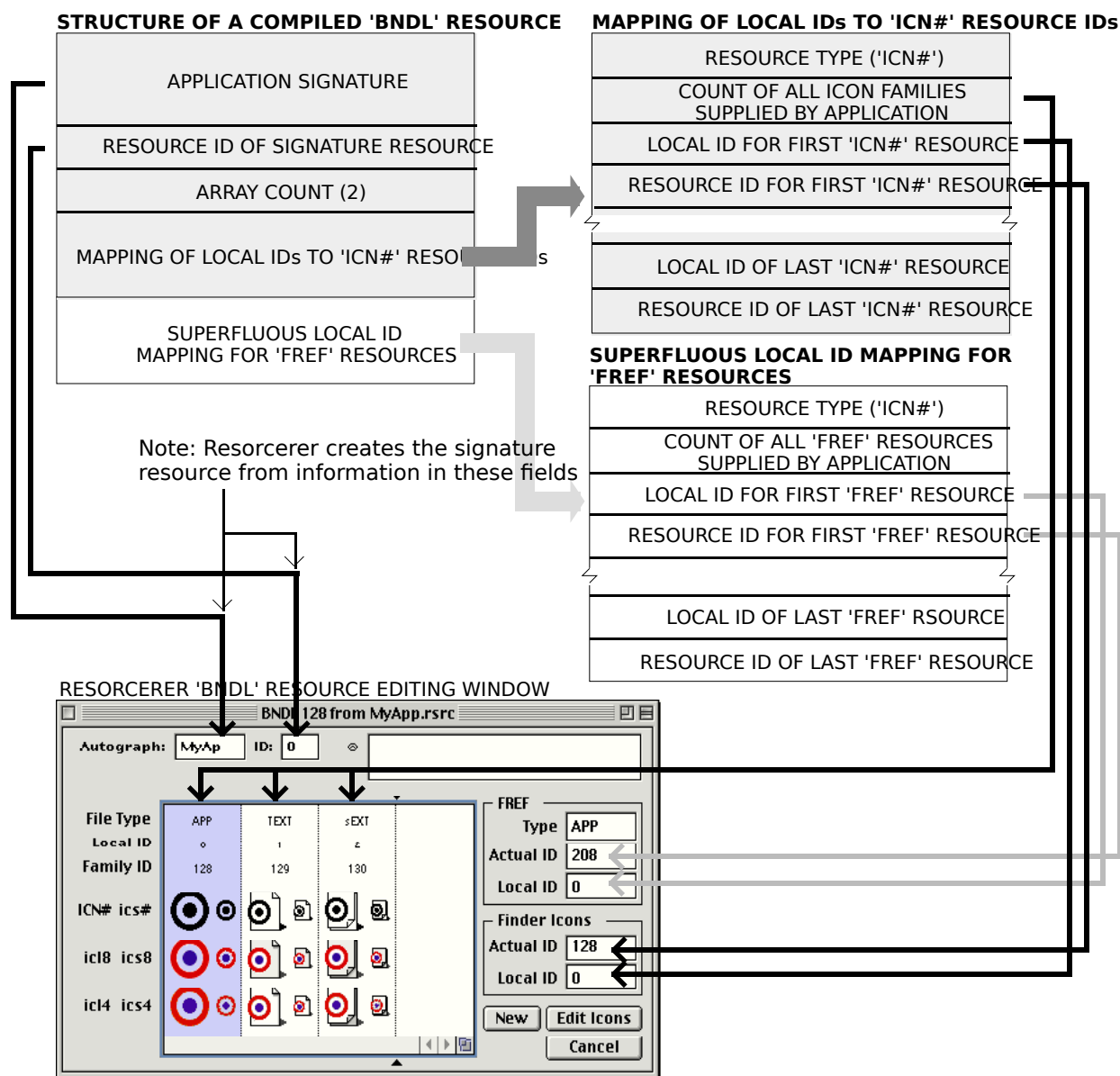

FIG 6 - STRUCTURE OF A COMPILED 'BNDL' RESOURCE AND CREATING A 'BNDL' RES

The following describes the main fields of a compiled 'BNDL' resource:

Field	Description
APPLICATION SIGNATURE	Identifies the application to the Finder.
RESOURCE ID OF THE SIGNATURE RESOURCE	Signature resource ID. (Always 0.)
LOCAL ID OF FIRST 'ICN#' RESOURCE	Must match the local ID assigned to the 'ICN#' resource within a 'FREF' resource.
RESOURCE ID OF FIRST 'ICN#'	To visually represent files of the type described in the 'FREF' resource that contains the

RESOURCE	local ID in the preceding field, the Finder uses the appropriate member of the icon family with the same resource ID as this 'ICN#' resource.
LOCAL ID OF FIRST 'FREF' RESOURCE	A local ID for the 'FREF' resource relating to this file type. (Superfluous.)
RESOURCE ID OF FIRST 'FREF' RESOURCE	The resource ID of the 'FREF' resource relating to this file type. (Superfluous.)

Note that you also assign local IDs to 'FREF' resources within the 'BNDL' resource. This assignment is, in fact, superfluous because the Finder does not map these local IDs to any other resources. It was implemented for the earliest versions of the system software, and it remains this way today for backward compatibility.

Resource IDs, Local IDs and the Desktop Database

As shown at Fig 6, you must assign local IDs to your 'ICN#' resources within your 'BNDL' resources. You must ensure that, for all your file types for which you provide icons, these local IDs match the local IDs you assigned inside their corresponding 'FREF' resources.

In the desktop database, the Finder renumbers the resource IDs that you have assigned to your resources to avoid conflicts with the resources of other applications. Therefore, the 'BNDL' resource has to rely on these local IDs to map 'ICN#' resources to their 'FREF' resources, that is, the 'BNDL' resource uses the local ID you assign to an 'ICN#' resource to map it to the 'FREF' resource that has specified the same local ID. For example, the 'FREF' resource with resource ID 208 at Fig 5 shows that the file type 'APPL' is assigned a local ID of 0. At Fig 6, you will see that local ID 0 is assigned to the 'ICN#' resource with resource ID 128. This maps the icon defined by this resource to the application file.

Fig 7 illustrates how the 'BNDL' resource uses local IDs to map 'ICN#' resources to 'FREF' resources.
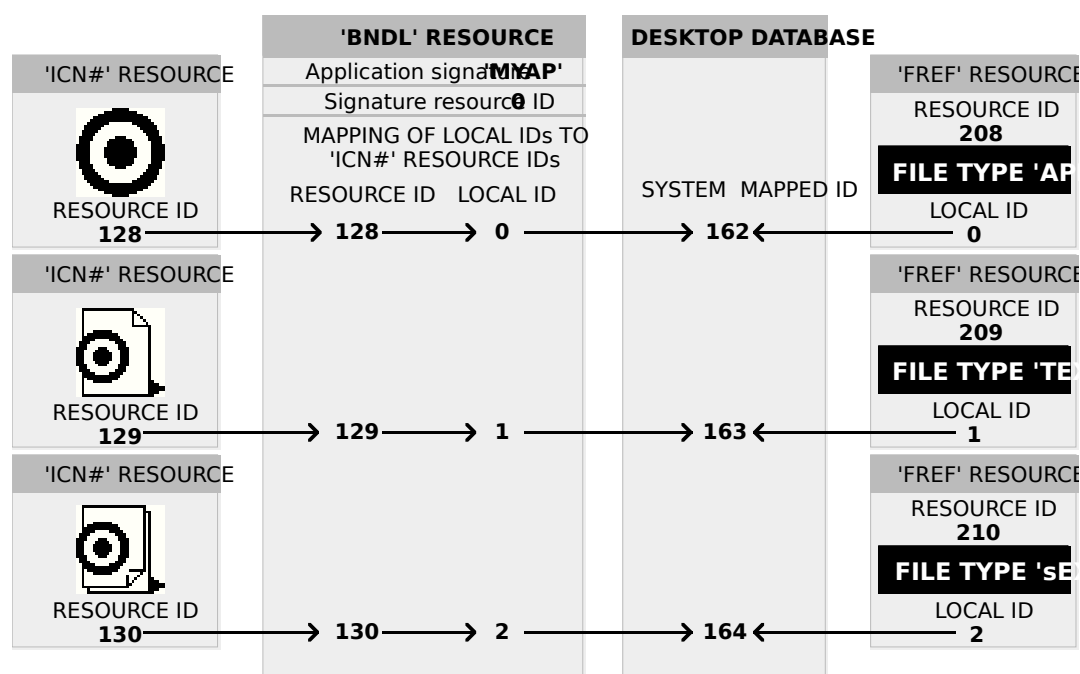

FIG 7 - HOW THE BUNDLE RESOURCE LINKS 'ICN#' RESOURCES AND

In Fig 7, and for illustrative purposes, the application file's 'ICN#' resource has a resource ID of 128 while its 'FREF' resource has a resource ID of 208. In an actual application, it is best if you assign the same resource ID to a file's 'FREF' resource that you assign to its 'ICN#' resource. This makes code maintenance easier.

Informing the Finder that Your Application has a 'BNDL' Resource

You alert the Finder that your application has a 'BNDL' resource by setting a bit in the file's Finder flags field (see below).
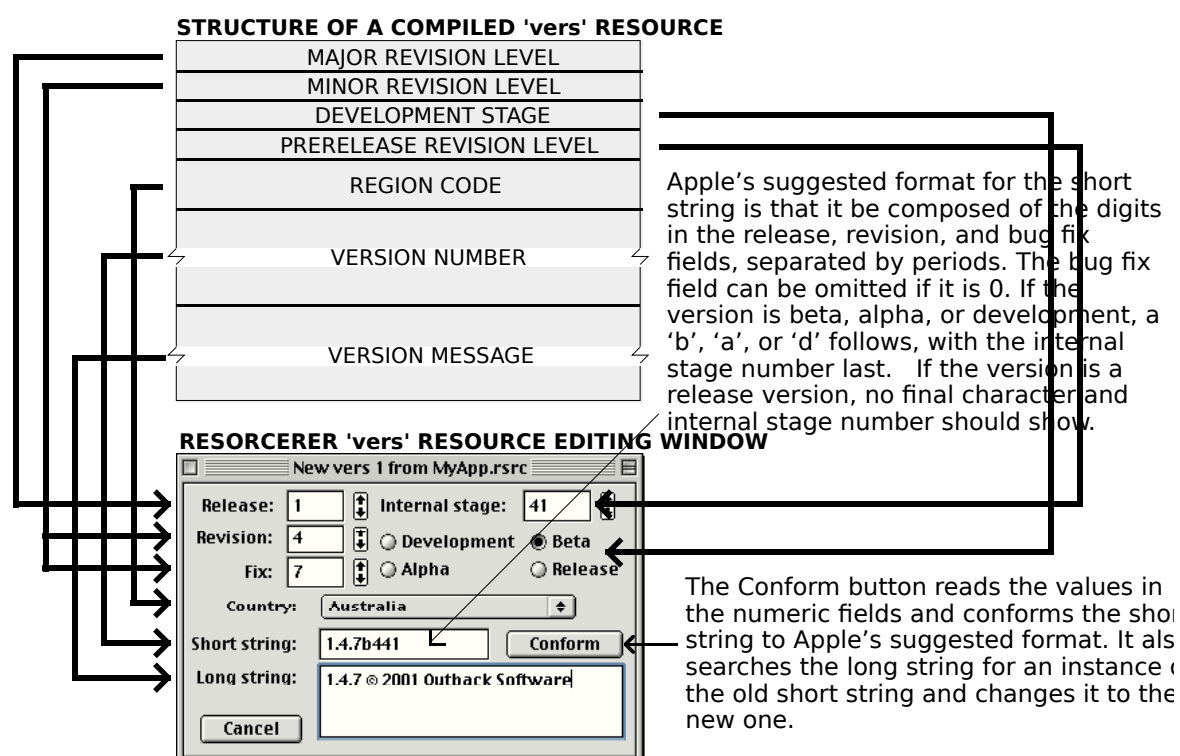
The Version ('vers') Resource

You use version ('vers') resources to supply version information to the Finder. There are two ways that the Finder displays this information:

- In Finder list views, the version number (short string) is displayed in the Version column.

- The version message (long string) is displayed in the Finder's Get Info window on Mac OS 8/9 and in the Finder's Show Info window on Mac OS X.

You can use 'vers' 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 'vers' resource with ID 1 specifies the version of the file; the 'vers' resource with ID 2 specifies the version of the set of files.

Structure of the 'vers' Resource and Creating a 'vers' Resource Using Resorcerer

Fig 8 shows the structure of a 'vers' resource and a 'vers' resource being created using Resorcerer.

STRUCTURE OF A COMPILED 'vers' RESOURCE

MAJOR REVISION LEVEL
MINOR REVISION LEVEL
DEVELOPMENT STAGE
PRERELEASE REVISION LEVEL
REGION CODE
VERSION NUMBER
VERSION MESSAGE

Apple's suggested format for the short string is that it be composed of the digits in the release, revision, and bug fix fields, separated by periods. The bug fix field can be omitted if it is 0. If the version is beta, alpha, or development, a 'b', 'a', or 'd' follows, with the internal stage number last. If the version is a release version, no final character and internal stage number should show.

RESORCERER 'vers' RESOURCE EDITING WINDOW

New vers 1 from MyApp.rsrc

Release: 1   Internal stage: 41
Revision: 4   ○ Development ● Beta
Fix: 7   ○ Alpha   ○ Release
Country: Australia
Short string: 1.4.7b441   [Conform]
Long string: 1.4.7 © 2001 Outback Software
[Cancel]

The Conform button reads the values in the numeric fields and conforms the short string to Apple's suggested format. It also searches the long string for an instance of the old short string and changes it to the new one.

FIG 8 - STRUCTURE OF A COMPILED 'vers' RESOURCE AND CREATING A 'vers' RESOURCE

The following describes the main fields of a compiled 'vers' resource:

Field	Description
MAJOR REVISION LEVEL	Major revision level in binary coded decimal format. Although the Finder does not display it anywhere, you can store this information here.
MINOR REVISION LEVEL	Major revision level in binary coded decimal format. Although the Finder does not display it anywhere, you can store this information here.
DEVELOPMENT STAGE	The values in this field indicate one of four development stages: development, alpha, beta, or released.
PRERELEASE REVISION LEVEL	The version if the software is still in prerelease.
REGION CODE	The script system for which this version of the software is intended.
VERSION NUMBER	The version number. This string appears in the Version column in list view windows when the Version checkbox in the View Options dialog is checked.
VERSION MESSAGE	The version number and a company copyright. This string appears in Get Info/General Information windows.

Finder Icon Help ('hfdr') Resource — Mac OS 8/9

The Mac OS 8/9 Finder provides a default help balloon for your application icon. However, you can provide a customised help balloon by adding a finder icon help override ('hfdr') resource with resource ID -5696 to the resource fork of your application.

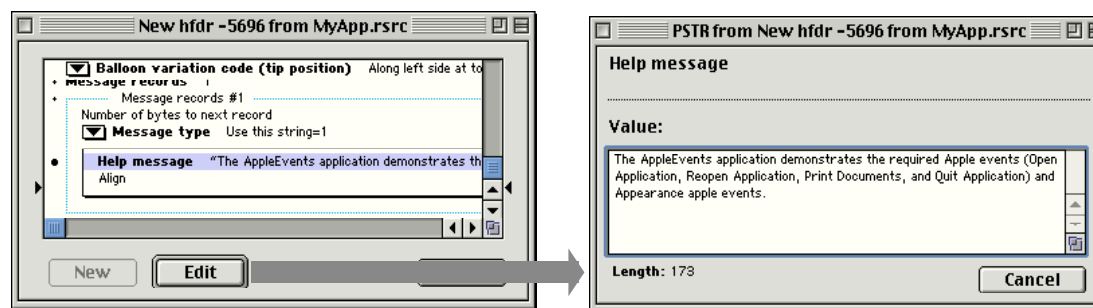Fig 9 shows a 'hfdr' resource being created using Resorcerer.

FIG 9 - CREATING AN 'hfdr' RESOURCE USING RESORCEREI

Missing Application and Application-Missing Resources — Mac OS 8/9

Missing application and application missing string resources are not supported on Mac OS X. Accordingly, the following is applicable to Mac OS 8/9 only.

Missing Application and Application Missing

On Mac OS 8/9, if the user tries to open a file created by an application that is missing in circumstances where the user has selected Translate documents automatically off in the File Translation tab of the File Exchange control panel, or if the user attempts to open a file which he/she should not be able to open (such as a Preferences file), the Finder responds by displaying a movable modal alert. The contents of that alert depend on the following factors:

- If the file is a document file created by an application that is missing, whether the resource fork of that file contains a **missing application name string resource**, that is, a 'STR ' resource with a resource ID of -16396.

- If the file is a file which is not meant to be opened, whether the file's resource fork contains an **application-missing string resource**, that is, a 'STR ' resource with ID -16397.

If the Finder cannot find the document's creator on any mounted volume, it looks first for the application-missing string resource. If it cannot find an application-missing string resource, it then searches for a missing-application name string resource.

Missing Application

Assuming that the document's name is "Document A", the alert at Fig 10 is displayed if the file does not contain a missing application name string resource and either the document is not of type 'TEXT' or 'PICT' or the document is of type 'TEXT' or 'PICT' and SimpleText is not available. If the document is of type 'TEXT' or 'PICT' and SimpleText is available, the alert at Fig 11 is displayed.
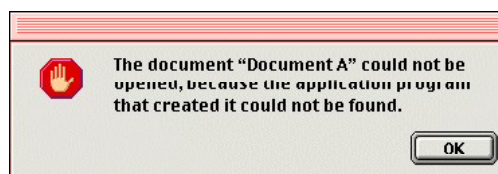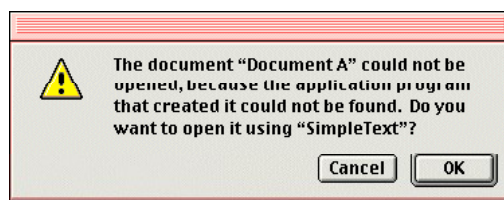

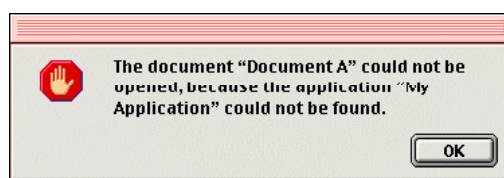**FIG 10 - THE DEFAULT MISSING APPLICATION M**

FIG 11 - THE MISSING APPLICATION MODAL ALERT
IS A 'TEXT' OR 'PICT' FILE AND SIMPLETEXT IS AVAILAB

Missing Application Name String Resource

The purpose of the missing application name string resource is to provide more specific information to the user. That information is the name of the application that created the program. This is achieved by:

- Including a 'STR ' resource containing your application's name in your application's resource fork.

- When your application saves a document for the first time, copying the resource from your application's resource fork to the resource fork of the newly-created document, ensuring that its resource ID is -16396. If this resource is present in the document's resource fork, and the user attempts to open the document when your application is not present, a movable modal alert similar to that at Fig 12 is displayed.

FIG 12 - THE MISSING APPLICATION MODAL ALERT
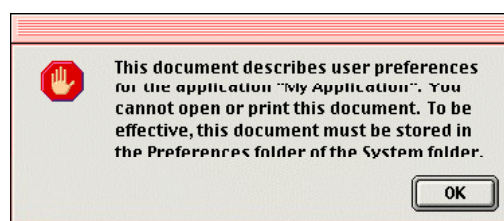**HAS A MISSING APPLICATION NAME STRING RESOU**

Application-Missing

If the user attempts to open a file that is not meant to be opened, and if the file does not contain an application-missing string resource, the Finder displays a modal alert similar to that at Fig 10.

Application-Missing String Resource

The purpose of the application-missing resource is to provide more specific information to the user. That information is an explanation of why the file cannot be opened. This is achieved by:

- Including a 'STR ' resource containing the explanation in your application's resource fork.

- When your application creates the file, copying the resource from your application's resource fork to the resource fork of the file, ensuring that its resource ID is -16397. If this resource is present in the document's resource fork, and the user attempts to open the file, a movable modal alert similar to that at Fig 13 is displayed.

FIG 13 - AN APPLICATION MISSING MODAL ALERT BO
TYPICAL APPLICATION-MISSING STRING RESOURC

The file must have a unique signature that no application known to the finder is likely to have. This ensures that the Finder will display your message, instead of attempting to launch the application with that signature, when the user double clicks on the file's icon.

One 'STR ' Resource Only

You supply *either* the missing-application name string resource *or* the application-missing string resource, but never both. For example, you would supply an application-missing string resource for your Preferences file (which the user should not be able to open) and a missing-application name string resource for documents which users should be able to open with your application.

The 'plst' Resource and Information Property Lists

As previously stated:

- If your application includes a 'plst' (property list) resource which itself includes a binary representation of an information property list containing the necessary information, the Mac OS X Finder extracts the information it requires from that list. In essence, the information property list is the interface through which your application communicates essential data to the Finder.

- If the 'plst' resource is empty, the Mac OS X Finder extracts the information provided by the signature, file reference, bundle, and version resources described in this chapter.

An information property list is a textual method of representing data which uses the Extensible Markup Language (XML) as the structuring medium, and which uses predefined keys for specifying information of interest to the Finder. This information is specified as key-value pairs (that is, as a dictionary). Standard keys are defined by Mac OS X for information such as the version string to be displayed in the Finder's Show Info dialog and the Finder defines keys for information such as the application's signature and type definitions for document types the application understands.

Example Information Property List

The following is an example of an information property list for a simple non-bundled application which edits text files:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist SYSTEM "file://localhost/System/Library/DTDs/PropertyList.dtd">
<plist version="0.9">
<dict>
  <key>CFBundleInfoDictionaryVersion</key>
  <string>6.0</string>
  <key>CFBundleIdentifier</key>
  <string>com.Smith Software.MyApp</string>
  <key>CFBundleShortVersionString</key>
  <string>1.0.0</string>
  <key>CFBundleDevelopmentRegion</key>
  <string>English</string>
  <key>NSHumanReadableCopyright</key>
  <string>Copyright © 2001 Smith Software</string>
  <key>NSAppleScriptEnabled</key>
  <string>NO</string>
  <key>CFBundleName</key>
  <string>MyApp</string>
  <key>CFBundlePackageType</key>
  <string>APPL</string>
  <key<CFBundleSignature></key>
  <string>myaP</string>
  <key>CFBundleIconFile</key>
  <string>128</string>
  <key>CFBundleShortVersionString</key>
  <string>Version 1.0.0 May 2001</string>
  <key>CFBundleDocumentTypes</key>
  <array>
   <dict>
     <key>CFBundleTypeIconFile</key>
     <string>129</string>
     <key>CFBundleTypeName</key>
     <string>MyApp TEXT Document</string>
     <key>CFBundleTypeOSTypes</key>
     <array>
      <string>TEXT</string>
     </array>
     <key>CFBundleTypeRole</key>
     <string>Editor</string>
```

```
        </dict>
      </array>
    </dict>
  </plst>
```

The following describes the standard and Finder keys and associated values in this information property list:

Standard Keys	Value
CFBundleInfoDictionaryVersion	A string used to support future versioning of the format.
CFBundleIdentifier	A string, used by the preferences system to uniquely identify the application, in the form of a Java-style package name.
CFBundleVersion	A string containing a 'vers' resource-style (see Fig 8) version number.
CFBundleDevelopmentRegion	A string specifying the application's "native" region. In 'plst' resources, this serves the same purpose as the Region Code in a 'vers' resource.
NSHumanReadableCopyright	A string containing copyright information to be displayed in the Finder's Show Inspector dialog.
NSAppleScriptEnabled	A string specifying whether the application is AppleScript enabled.

Finder Keys	Value
CFBundleName	A string containing the name of the application.
CFBundlePackageType	A string containing, for applications, the file type APPL.
CFBundleSignature	A string containing the application's four-letter creator code.
CFBundleIconFile	A string containing the resource ID of the 'icns' resource containing the application's icon.
CFBundleShortVersionString	A string containing human-readable description of the version. This should be more than just the string that can be generated from the CFBundleVersion key.
CFBundleDocumentTypes	An array of type definitions for document types the application understands.
CFBundleTypeName	A string containing a name for this document type.
CFBundleTypeIconFile	A string containing the resource ID of the 'icns' resource containing this file type's icon.
CFBundleTypeRole	A string defining the application's role in relation to this document type. The role can be Editor, Viewer, or None.
CFBundleTypeOSTypes	An array of strings containing the four-letter file type codes that map to this type.

Creating the 'plst' Resource

You can create the complete 'plst' resource by simply copying the XML listing and pasting it into an empty 'plst' resource using Resorcerer.

Application Launching by the Finder

The following describes how the Finder launches an application.

User Double-Clicks the Application's Icon, selects the Application Icon and Chooses Open From the Finder's File Menu, or Chooses the Application From the Mac OS 8/9 Apple Menu

When the user double clicks your application icon, selects it and chooses Open from the Finder's File menu, or chooses it from the Mac OS 8/9 Apple menu, the Finder calls the Process Manager to launch your application. The Finder then sends your application a required Apple event (called an Open Application event) before relinquishing control to your application. In response to the Open Application event, your application should then perform its usual startup actions, such as opening an untitled document window.

User Double-Clicks a Document Icon or Selects One or More Document Icons and Chooses Open from the Finder's File Menu or Chooses Print from the Mac OS 8/9 File Menu

When the user double-clicks on the icon for a document created by your application, or selects one or more document icons and chooses Open from the Finder's File menu or chooses Print from the Mac OS 8/9 File menu, the Finder reads the creator field of each selected file to find the document's creator. If a document's creator matches your application's signature, the Finder calls the Process Manager to launch your application, and then sends your application a required Apple event (called an Open Documents or (Mac OS 8/9 only) Print Documents event) before relinquishing control to your application. These events contain the name, or names, of the document, or documents, to open or (on Mac OS 8/9 only) print. Your application should then open the documents or print them, as appropriate.

User Drags One or More Document Icons to the Application Icon

When the user drags one or more document icons to your application's icon, the Finder determines whether to launch your application by comparing the document's file type (stored in the catalog file) with the list of the file types supported by your application. If the document's type appears in your application's 'FREF' resource or, where your application's 'plst' resource contains an information property list, in that information property list (Mac OS X only), the Finder calls the Process Manager to launch your application, and passes the name/s of the document/s to your application in a required Apple event (Open Documents event). Your application should then open the document/s.

User Double Clicks a Document Icon — Application Already Running

If the user double clicks a document item while your application is already running, the Finder sends your application an Open Documents event.

The Finder and the Catalog File

The catalog file, which maintains relationships between the files and directories on a volume, is of prime importance to the File Manager; however, information in the catalog file is also used by the Finder.

In the catalog file, information for files is stored in **file information** (FInfo) and **extended file information** (FXInfo) **structures** and information for directories is stored in **directory information** (DInfo) and **extended directory information** (DXInfo) structures. The Finder can manipulate fields in file information, directory information, and extended directory information structures.

When your application creates a new file, it assigns file type and creator information to the relevant fields of the file's file information structure[4]. The Finder manipulates the other fields of the file information structure.

The file information structure is as follows:

```
struct FInfo
{
  OSType fdType;     // Type of file.
  OSType fdCreator;  // File's creator.
  UInt16 fdFlags;    // Finder flags (invisible, locked, stationery, etc).
  Point  fdLocation; // File's location in folder.  ({0,0} places item automatically.)
  SInt16 fdFldr;     // Folder containing file.
};
typedef struct FInfo FInfo;
```

For a file that has already been created, you can set the type, creator and flags fields using the File Manager function FspSetFInfo, having first called FSpGetInfo to return the file information structure.

---

[4]   See Chapter 18.

Finder Flags

Individual bits of the `fdFlags` field of the file information structure may be examined and set using constants defined in the header file Finder.h. The bits of most interest to an application, and the associated constants, are as follows:

Bit	Constant	Meaning When Set
10	kHasCustomIcon	The file has a customised icon.
11	kIsStationery	The file is a stationery pad.
		To support stationery pads, your application should check this bit for every document passed to it by the Finder and Navigation Services.
12	kNameLocked	The file cannot be renamed from the Finder (and, also, cannot have customised icons assigned to it).
13	kHasBundle	The file has a `'BNDL'` resource.
		If the `hasBundle` bit is set, the Finder checks the `hasBeenInited` flag. If the `hasBeenInited` bit is set, the Finder uses database information to display the file's icon. If the `hasBeenInited` bit is not set, the Finder copies the information from the bundle resource to the database and sets the `hasBeenInited` bit.
		(If the `hasBundle` bit is not set, the Finder displays the default icon for that file type.)
14	kIsInvisible	The file is invisible from the Finder and from Navigation Services.
15	kIsAlias	The file an alias.

Preferences, Temporary Items, and Trash Folders — Using FindFolder

The only system-related folders you are ever likely to need are the Preferences folder, the Temporary Items folder, and the Trash folder.

You can use the `FindFolder` function to get the volume reference number and directory ID of these folders. The following **folder type constants** are passed in the `folderType` parameter of `FindFolder` to specify the folder type:

Constant	Value	Folder
kPreferencesFolderType	'pref'	Preferences.
kTemporaryFolderType	'temp'	Temporary items.
kTrashFolderType	'trsh'	Trash

Stationery Pads

Stationery Pads and the Finder

If the user opens a stationery pad from the Finder, the Finder first checks the `isStationeryAware` bit in your application's `'SIZE'` resource. If the `isStationeryAware` bit is set, the Finder informs your application that the user has opened a document and passes your application the name of the stationery pad. On the other hand, if the `isStationeryAware` bit is not set, the Finder *creates a new document* from the template and starts up your application, passing it the name of the *new* document.

Stationery Pads and Navigation Services

Unlike the Finder, Navigation Services (see Chapter 18) always passes your application the stationery pad itself, not a copy of it, regardless of the setting of the `isStationeryAware` bit in your application's `'SIZE'` resource. This means that the user can mistakenly over-write the stationery pad by saving it without assigning it a new name, a danger that can be avoided by making your application stationery-aware.

The key step in making your application stationery-aware is to always checking the kIsStationery bit of a document before opening it. The following is an example function which takes a file system specification structure and returns true or false according to whether the file is a stationery document or not:

```
Boolean  isStationeryDoc(FSSpec fileSSpec)
{
  OSErr   osErr;
  FInfo   fInfo;
  Boolean result;

  osErr = FSpGetFInfo(&fileSSpec,&fInfo);
  if(osErr == noErr)
    result = ((fInfo.fdFlags & kIsStationary) == kIsStationary);
  else
    result = false;

  return(result);
}
```

Using Aliases

An **alias** is an object that represents another file, directory or volume. Both the Finder and Navigation Services resolve aliases before passing them to your application. However, if your application opens a file or directory without going through the Finder or Navigation Services, your application should always call ResolveAlias just before opening the file.

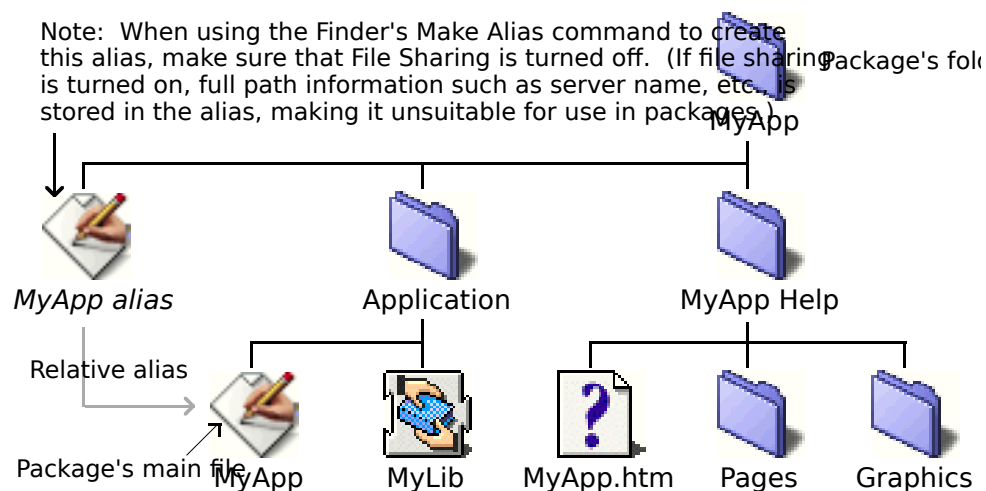# Mac OS 9 Packages and Mac OS X Bundles

Mac OS 9 Packages

Mac OS 9 introduced **packages**. Basically, a package is a folder with the "package bit" set. The package bit is bit 13 of the frFlags field of the directory information structure. (This bit is the equivalent, for folders, of the bundle bit for files (bit 13 of the fdFlags field of the file information structure). Recall that the bundle bit for files determines how the file's icon is displayed.)

Any folder which has the package bit set, and which contains exactly one alias at its topmost level, will be treated by the Finder as a package.

Normally, a package comprises an application and a number of support files organised into sub-directories. There are no rules governing the arrangement of files and sub-directories other than that there must be one alias at the topmost level. This alias points to some other file in the package's directory hierarchy. The file pointed to is called the "package's main file", and is ordinarily, but need not be, an application. An example of an application package is shown at Fig 14.

Note: When using the Finder's Make Alias command to create this alias, make sure that File Sharing is turned off. (If file sharing is turned on, full path information such as server name, etc., is stored in the alias, making it unsuitable for use in packages.)

FIG 14 - A PACKAGE (EXAMPLE)

The Mac OS 8/9 Finder treats this package as if it were a file containing the 'BNDL', 'FREF', icon resources, etc., found in the package's main file. The package's folder is visually presented to the user as if it were a

single double-clickable file, specifically, the package's main file. If, for example, a file is dragged over an application package's icon, the Finder will track the drag command as if the file was being dragged over the package's main file.

The major advantages of application packages are as follows:

- Since they cause the Finder to hide the contents of a package's folder from the user, they prevent accidental (or intentional) tampering with an application's support files.

- They simplify installation and uninstallation. All the user has to do is drag the application package onto a volume or, for uninstall, drag it to the trash.

Note also that the package's name and the package's main file name may differ. This allows the user to edit the name of the package without disturbing the name of the package's main file.

Creating Packages

The Apple application PackageTool may be used to convert correctly formatted folders into packages and vice versa.

Mac OS X Bundles

On Mac OS X, the option exists to package an application as a **bundle**. A bundle is a directory that contains, in a hierarchical organization, the application's executable and the resources required to support that code. One of the required contents of a bundle directory is a file containing the same XML code as would be contained in the 'plst' resource in a non-bundled version of the application.

A bundle can be presented to the user as a directory. However, as is the case with Mac OS 9 packages, if bit 13 of the frFlags field of the directory information structure is set, the Finder presents the bundle directory to the user as if it were a single file.

Packaging applications as bundles is not addressed in this book.

Relevant Constants, Data Types, and Functions

Constants

Finder Flags

```
kIsOnDesk          = 0x1
kColor             = 0xE
kIsShared          = 0x40
kHasBeenInited     = 0x100
kHasCustomIcon     = 0x400
kIsStationary      = 0x800
kNameLocked        = 0x1000
kHasBundle         = 0x2000
kIsInvisible       = 0x4000
kIsAlias           = 0x8000
```

Folder Type Constants

```
kTrashFolderType       = FOUR_CHAR_CODE('trsh')
kPreferencesFolderType = FOUR_CHAR_CODE('pref')
kTemporaryFolderType   = FOUR_CHAR_CODE('temp')
```

Data Types

File Information Structure

```
struct FInfo
{
  OSType fdType;     // Type of file.
  OSType fdCreator;  // File's creator.
  UInt16 fdFlags;    // Finder flags (invisible, locked, stationery, etc).
  Point  fdLocation; // File's location in folder.  ({0,0} places item automatically.)
  SInt16 fdFldr;     // Folder containing file.
};
typedef struct FInfo FInfo;
```

Directory Information Structure

```
struct DInfo
{
  Rect   frRect;     // Folder's window bounds.
  Uint16 frFlags;    // Finder flags (invisible, locked, etc).
  Point  frLocation; // Folder location. ({0,0} places item automatically.)
  SInt16 frView;     // Reserved.  (Set to 0.)
};
typedef struct DInfo DInfo;
```

Functions

```
OSErr  FSpGetFInfo(const FSSpec *spec,FInfo *fndrInfo);
OSErr  FSpSetFInfo(const FSSpec *spec,const FInfo *fndrInfo);
OSErr  ResolveAliasFile(FSSpec *theSpec,Boolean resolveAliasChains,Boolean *targetIsFolder,
       Boolean *wasAliased);
OSErr  FindFolder(short vRefNum,OSType folderType,Boolean createFolder,short *foundVRefNum,
       long *foundDirID);
```

Demonstration Programs

As previously stated, this chapter should be read in conjunction with Chapter 10. The resources for the demonstration program at that chapter include a signature resource, icon resources, 'FREF' resources, a 'BNDL' resource, a 'vers' resource, a 'hfdr' resource, and a 'plst' resource containing an information property list.

Missing Application Name String and Application-Missing String ('STR ') Resources

The demonstration program at Chapter 18 shows how to copy a missing application name string resource to the resource fork of a document file.

The demonstration program at Chapter 19 shows how to copy an application-missing string resource to the resource fork of a Preferences file.

Preferences, Temporary Items, and Trash Folders

The demonstration program at Chapter 19 demonstrates creating and accessing a preferences file in the Preferences folder.

The demonstration program at Chapter 18 demonstrates creating a temporary file in, and deleting it from, the Temporary Items folder.

The demonstration program at Chapter 18 demonstrates determining whether a file is in the Trash folder, or in a folder in the Trash folder.