Obtaining and Using Icons With Icon Services

10/20/99
Technical Publications

Contents

Figures and Listings

C H A P T E R 1

Using Icon Services

Contents

Icon Services Overview

This chapter starts with an overview of Icon Services and follows with a detailed description of how to use Icon Services. You should read this chapter if you are interested in using Icon Services to obtain and display icons for your application or extension. This document assumes that you are familiar with the basic concepts of icon creation and usage, as described in *Macintosh Human Interface Guidelines* and *Inside Macintosh*.

Icon Services provides icon data to multiple Mac OS clients, including the Finder, extensions and applications. Using Icon Services to obtain icon data means you can provide efficient icon caching and release memory when you don't need icon data any longer. Icon Services provides the appropriate icon for any **file object** (file, folder, or volume), as well as other commonly used icons such as caution, note, or help icons in alert boxes, for example. The icons provided by Icon Services support a much larger palette of colors: up to 24 bits per pixel and an eight-bit mask. Icons are Appearance-compliant and appropriate to the active theme.

The IconRef

The basic data type used by Icon Services is the `IconRef`, a 32-bit opaque value. You obtain an `IconRef` by calling one of the `GetIconRef` functions described in "Obtaining and Releasing IconRefs" (page 10). Two or more files that have the same file type and creator and do not provide custom icons will use the same `IconRef`. Files with custom icons have their own `IconRef`.

Reference Counting

`IconRef` values are reference counted, so that the icon data represented by a particular `IconRef` can be shared by several clients simultaneously. Each client that uses a particular `IconRef` increments that `IconRef`'s reference count. When there are no more clients using a particular `IconRef`, the icon data associated with it is disposed of.

The 'icns' Resource

The `'icns'` resource is a means of providing a single source for icon data, as opposed to the variety of icon resources represented by `‘ICN#’`, `‘icl8’` and other familiar resource types. Combining all icon data into a single resource type speeds up icon fetching and simplifies resource management.

32-bit Icon Data

The `'icns'` resource provides for 32-bit-deep icon data.

Deep Masks

Icons provided through the `'icns'` resource feature **deep masks**, meaning an icon mask can have 256 different levels of transparency.

Huge Icons

The `'icns'` resource adds “huge” icons, which are 48 pixels by 48 pixels, as well as providing the sizes contained in other icon resources. For more information, see `'icns'` (page 60).

Basic Tasks With Icon Services

Obtaining and Releasing IconRefs

In order to call Icon Services functions, your application must obtain an `IconRef` for the icon data you want to use. There are three functions you can use to accomplish this task; the one you choose depends on how much information you have about the icon you wish to use. If you need an icon from the desktop database or a **registered icon** (an icon that has been previously identified to Icon Services), the simplest and fastest way to obtain an `IconRef` is to use the function `GetIconRef` (page 23). Icon Services defines a number of constants for non-fileFinder icons, which makes it simple to use the `GetIconRef` function to obtain an `IconRef` for one of these icons. Listing 1-1 shows an example of how to obtain an `IconRef` for the standard Help icon:

Listing 1-1 Obtaining an IconRef for the standard help icon

```
err = GetIconRef(kOnSystemDisk, kSystemIconsCreator, kHelpIcon,
                &iconRef);
```

If you need an `IconRef` for a folder that you know has no custom icons associated with it, use the function `GetIconRefFromFolder` (page 25). If you need an `IconRef` for a file object for which you don't have any information, use the function `GetIconRefFromFile` (page 24).

Using IconRefs

Once you obtain a valid `IconRef`, you can use Icon Services functions to accomplish two major types of tasks. The first type of task is to draw an icon of the appropriate size and type in a specified area. The second task is checking to see whether the user has clicked on or selected an icon (also known as hit-testing). These functions are designed to be similar to familiar Icon Utilities functions.

Drawing Icons

Icon Services provides two basic drawing functions. For the task of drawing an icon directly to the screen, use the function `PlotIconRef` (page 33). If you need to convert icon data into a QuickDraw region, use the function `IconRefToRgn` (page 37).

Note
The introduction of deep masks means that you cannot simply draw over an icon and assume the previous icon will be erased. Call the function `IconRefToRgn` (page 37) to determine the area occupied by the current icon, erase that area, then draw the new icon. ◆

Hit-Testing

Icon Services provides several ways to determine whether a user has interacted with an icon. To determine whether a user has clicked on an icon, use the function `PtInIconRef` (page 34). If you need to determine whether an icon falls with a given rectangle (like a selection rectangle, for example), use the function

`RectInIconRef` (page 36). You can also use the function `IconRefToRgn` (page 37) to do hit-testing.

Advanced Tasks With Icon Services

Icon Services gives you several ways to modify the icon data used by Icon Services. You can add icon data to the Icon Services cache by registering icon data with the functions `RegisterIconRefFromIconFamily` (page 27) or `RegisterIconRefFromIconFile` (page 28). You can also release registered data by using the function `UnregisterIconRef` (page 30). You can override existing data in an `IconRef` and replace it with custom data by using the functions `OverrideIconRef` (page 41) or `OverrideIconRefFromResource` (page 40). You can also restore the original data in an `IconRef` by using the function `RemoveIconOverride` (page 41).

Registering Icon Data

You can speed up access to frequently-used icon data by registering icons. The preferred way of registering icons is to use the function `RegisterIconRefFromIconFile` (page 28). This will make the icon data purgeable if you need to free up memory later. If you can't use the `RegisterIconRefFromResource` function, you can use the function `RegisterIconRefFromIconFamily` (page 27). To unregister icon data, use the function `UnregisterIconRef` (page 30).

Updating IconRefs

If you need to refresh icon data without releasing an `IconRef`, you can use the function `UpdateIconRef` (page 40) to obtain current icon data. This might be useful after you have changed a file's custom icon, for example.

Overriding and Restoring Icon Data

You may wish to redraw icons on a temporary basis without going to the trouble of obtaining a new `IconRef`. One example of this is when a user has started to download a file and you want to use partial file icons to represent the various stages of the download process. Icon Services provides two functions to temporarily override the data in an `IconRef`; the one you choose depends on the source of the data you will use for the override. If you obtain the source data from another `IconRef`, use the function `OverrideIconRef` (page 41). If the source data is contained in a resource, use the function `OverrideIconRefFromResource` (page 40). You can restore the original icon data by using the function `RemoveIconOverride` (page 41).

Modifying Reference Counts

You may find it useful to modify the reference count of an `IconRef` in preference to obtaining multiple `IconRefs` to the same data. This might be useful when your application maintains multiple instances of the same icon data, such as multiple file-copying operations. Use the function `AcquireIconRef` (page 47) to increment the reference count for an `IconRef`. This allows you to keep the icon data in the cache without going through the trouble of obtaining additional `IconRefs`. If you modify the reference count directly, be sure to decrement the reference count as needed by using the function `ReleaseIconRef` (page 47).

Flushing IconRefs

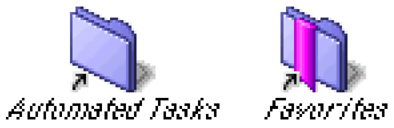
When you want to free up memory by releasing purgeable icon data from the cache, the preferred method is to use the function `FlushIconRefs` (page 48). You may also use the function `FlushIconRefsByVolume` (page 48), but this has two potential problems:

1. The additional scope of the `FlushIconRefsByVolume` function means it will take longer to complete.
2. Using the the `FlushIconRefsByVolume` function makes the icon data of all currently used `IconRefs` non-purgeable. This means any subsequent efforts to flush icon data will be much less effective.

Using Badges

A **badge** is an overlay or replacement for an icon. You can use a badge to signify that a folder contains special files, for example. Badges are described by a `'badg'` resource you store in a file’s resource fork or a folder’s invisible icon file. Figure 1-1 shows a folder alias icon displayed in standard form and with a badge.

Figure 1-1 Folder icons displayed in standard form and with a badge



There are two steps required to use a custom badge with a file object.

- Clear the `kExtendedFlagsAreInvalid` bit and set the `kExtendedFlagHasCustomBadge` bit in the `extendedFinderFlags` field of the `ExtendedFileInfo` structure (if the object is a file) or the `ExtendedFolderInfo` structure (if the object is a folder).
- Add a resource of type `kCustomBadgeResourceType` (`'badg'`) and ID `kCustomBadgeResourceID` to a file's resource fork or a folder's icon file.

There are three ways to use the data from a `'badg'` resource.

1. The `customBadgeType` and `customBadgeCreator` fields let you designate a custom badge to display on top of another icon, as shown in Figure 1-1.
2. The `windowBadgeType` and `windowBadgeCreator` fields let you designate which icon to display in Finder window header of the badged file or folder.
3. The `overrideType` and `overrideCreator` fields let you designate the badge as a replacement for the standard icon for this file or folder.

The type and creator codes specified in a`'badg'` resource must be registered with Icon Services before you can use the badge. For more information, see "Registering Icon Data" (page 12).

If you supply a custom icon resource for a badge, Icon Services will use it in preference to other available data. For a complete description of the badge resource, see `'badg'` (page 62).

Guidelines for Designing Icons

Here are some guidelines to follow if you choose to design custom icons for use with Icon Services.

- You must provide at least one of the following icon types: `'ICN#'`, `'ics#'`, or `'icl#'`.

- If you provide a deep mask, all of the non-transparent pixels in the deep mask should correspond to a black pixel in the black-and-white mask. This is important for hit-testing and proper erasing and drawing of the icon.
- If you provide 32-bit icon data, you should also provide an 8-bit version of the icon. This ensures that the icon can be displayed on an 8-bit display without unwanted dithering.
- 8-bit icon data is no longer limited to the 34 colors of the classic icon color palette. All 256 colors from the System palette are available.
- 4-bit icons are still supported. However, the 4-bit display configuration is rarely used and often unsupported, so we recommend that you do not provide 4-bit icon data. If you don't provide 4-bit icon data, Icon Services will use black-and-white icon data instead.

Icon Services Reference

Contents

Icon Services Gestalt Selector

Before calling any Icon Services functions, your application should pass the selector `gestaltIconUtilitiesAttr` to the `Gestalt` function.

```
enum {
    gestaltIconUtilitiesAttr= 'icon'
};
```

Constant description

`gestaltIconUtilitiesAttr`
: The `Gestalt` selector passed to determine which features of Icon Services are present. The `Gestalt` function produces a 32-bit value whose bits you should test to determine which Icon Services features are available.

You can use the following bit mask to test for the presence of Icon Services.

```
enum{
    gestaltIconUtilitiesHasIconServices = 4
};
```

Constant description

`gestaltIconUtilitiesHasIconServices`
: If this bit is set, Icon Services is installed.

Icon Services Functions

Icon Services provides functions to perform the following types of tasks:

- "Enabling and Disabling Custom Icons" (page 22)
- "Obtaining IconRef Values" (page 23)
- "Registering and Unregistering IconRef Values" (page 26)
- "Obtaining Icon Data" (page 30)

- "Using IconRef Data" (page 33)
- "Modifying IconRef Data" (page 40)
- "Reading, Copying, and Converting Icon Data" (page 42)
- "IconRef Reference Counting" (page 46)
- "Flushing IconRef Data" (page 48)

Enabling and Disabling Custom Icons

Icon Services provides the following functions to enable and disable custom icons:

- `SetCustomIconsEnabled` (page 22)
- `GetCustomIconsEnabled` (page 23)

SetCustomIconsEnabled

Enables or disables custom icons on a specified volume.

```
pascal OSErr SetCustomIconsEnabled(
                SInt16     vRefNum,
                Boolean    enableCustomIcons);
```

`vRefNum`	A value of type `SInt16` specifying the volume where custom icons are to be enabled or disabled.
`enableCustomIcons`	A value of type `Boolean`. If you pass `true`, custom icons are enabled on the volume specified. Passing `false` disables custom icons on the volume specified.
function result	A result code. See "Icon Services Result Codes" (page 59) for a description of possible return values.

DISCUSSION

If you use the `SetCustomIconsEnabled` function to enable or disable custom icons, the setting remains in effect only as long as the specified volume remains mounted during the current session.

GetCustomIconsEnabled

Tells you whether custom icons are enabled or disabled on a specified volume.

```
pascal OSErr GetCustomIconsEnabled(
                  SInt16    vRefNum,
                  Boolean   *customIconsEnabled);
```

`vRefNum`	A value of type `SInt16` specifying the volume whose status you are querying.
`customIconsEnabled`	A pointer to a value of type `Boolean`. On return, Icon Services sets this value to `true` if custom icons are enabled on the volume specified or `false` if custom icons are disabled on the volume specified.
function result	A result code. See "Icon Services Result Codes" (page 59) for a description of possible return values.

Obtaining IconRef Values

Icon Services provides the following functions to obtain `IconRef` values:

- `GetIconRef` (page 23)
- `GetIconRefFromFile` (page 24)
- `GetIconRefFromFolder` (page 25)

GetIconRef

Provides an `IconRef` for an icon in the desktop database or for a registered icon.

```
pascal OSErr GetIconRef(
                  SInt16    vRefNum,
                  OSType    creator,
                  OSType    iconType,
                  IconRef   *iconRef);
```

`vRefNum`	A value of type `SInt16` specifying the volume where Icon Services should start to search for the desired icon. Pass the

	`kOnSystemDisk` constant if you are not sure which value to specify in this parameter.
`creator`	A value of type `OSType` specifying the creator code of the desired icon.
`iconType`	A value of type `OSType` specifying the type code of the desired icon.
`iconRef`	A pointer to a value of type `IconRef`. On return, this value contains a reference to the desired icon data.
function result	A result code. See "Icon Services Result Codes" (page 59) for a description of possible return values.

DISCUSSION

Icon Services defines constants for commonly-used system icons. You can pass one of these constants in the `iconType` parameter if you specify `kSystemIconCreator` in the `creator` parameter. See "Folder Icon Constants" (page 49) for a list of these constants.

Calling the `GetIconRef` function increments the reference count of the `IconRef`.

Remember to call the function `ReleaseIconRef` (page 47)
when you are done with an `IconRef`.

GetIconRefFromFile

Provides an `IconRef` for a file, folder or volume.

```
pascal OSErr GetIconRefFromFile (
                FSSpec  *theFile,
                IconRef *iconRef,
                SInt16  *theLabel);
```

`theFile`	A pointer to a structure of type `FSSpec` specifying the file, folder or volume for the `IconRef`. For more information on the `FSSpec` structure, see *Inside Macintosh: Files*.
`iconRef`	A pointer to a value of type `IconRef`. On return, this value contains a reference to the desired icon data.

`theLabel`	A pointer to a value of type `SInt16`. On return, this value specifies the file or folder's label.
function result	A result code. See "Icon Services Result Codes" (page 59) for a description of possible return values.

DISCUSSION

Use this function if you have no information about the file object passed in the `theFile` parameter. If you have already called the File System Manager function `PBGetCatInfo`, you can use the function `GetIconRefFromFolder` (page 25) if the object is a folder without custom icons or the function `GetIconRef` (page 23) if the object is a file without custom icons. For more information on the `PBGetCatInfo` function, see *Inside Macintosh: Files*. The `GetIconRefFromFile` function increments the reference count of the `IconRef`.

Remember to call the function `ReleaseIconRef` (page 47) when you're done with an `IconRef`.

GetIconRefFromFolder

Provides an `IconRef` for a folder with no custom icon.

```
pascal OSErr GetIconRefFromFolder (
                    SInt16    vRefNum,
                    SInt32    parentFolderID,
                    SInt32    folderID,
                    SInt8     attributes,
                    SInt8     accessPrivileges
                    IconRef   *iconRef);
```

`vRefNum`	A value of type `SInt16` specifying the volume where the folder is located.
`parentFolderID`	A value of type `SInt32` specifying the ID of the desired folder's parent folder.
`folderID`	A value of type `SInt32` specifying the ID of the desired folder.
`attributes`	A value of type `SInt8` specifying the attributes of the desired folder. You can obtain this data from the

	`CInfoPBRec.dirInfo.ioFlAttrib` field of the folder's catalog information record.
`accessPrivileges`	A value of type `SInt8` specifying the access privileges of the specified folder. You can obtain this data from the `CInfoPBRec.dirInfo.ioACUser` field of the folder's catalog information record.
`iconRef`	A pointer to a value of type `IconRef`. On return, this value contains a reference to the desired icon data.
function result	A result code. See "Icon Services Result Codes" (page 59) for a description of possible return values.

DISCUSSION

If you don't have the catalog information for a folder, use the function `GetIconRefFromFile` (page 24). For more information on the `PBGetCatInfo` function, see *Inside Macintosh: Files*.

Calling the `GetIconRefFromFolder` function increments the reference count of the `IconRef`.

Remember to call the function `ReleaseIconRef` (page 47)
when you're done with an `IconRef`.

Registering and Unregistering IconRef Values

Icon Services provides the following functions to register and unregister `IconRef` values:

- `RegisterIconRefFromIconFamily` (page 27)
- `RegisterIconRefFromIconFile` (page 28)
- `RegisterIconRefFromResource` (page 28)
- `UnregisterIconRef` (page 30)

RegisterIconRefFromIconFamily

Adds an `iconFamily`-derived `IconRef` to the Icon Services registry.

```
pascal OSErr RegisterIconRefFromIconFamily (
                   OSType           creator,
                   OSType           iconType,
                   IconFamilyHandle iconFamily,
                   IconRef          *iconRef);
```

`creator`	A value of type `OSType` specifying the creator code of the desired icon. You can use your application's creator code, for example. **Note** Lower-case creator codes are reserved for the System. ◆
`iconType`	A value of type `OSType` specifying the type code of the desired icon.
`iconFamily`	A handle specifying the `iconFamily` data structure to register.
`iconRef`	A pointer to a value of type `IconRef`. On return, this value contains a reference to the desired icon data.
function result	A result code. See "Icon Services Result Codes" (page 59) for a description of possible return values.

DISCUSSION

Consider using the function `RegisterIconRefFromIconFile` (page 28), since the data registered using the `RegisterIconRefFromIconFamily` function cannot be purged. You are responsible for disposing of the `IconRef` by using the function `ReleaseIconRef` (page 47).

Calling this function increments the reference count of the `IconRef`.

RegisterIconRefFromIconFile

Adds a file-derived `IconRef` to the Icon Services registry.

```
pascal OSErr RegisterIconRefFromIconFile (
                    OSType         creator,
                    OSType         iconType,
                    const FSSpec   *iconFamily,
                    IconRef        *theIconRef);
```

`creator`	A value of type `OSType` specifying the creator code of the icon data you wish to register. You can use your application's creator code, for example. **Note** Lower-case creator codes are reserved for the system. ◆
`iconType`	A value of type `OSType` specifying the type code of the icon data you wish to register.
`iconFamily`	A pointer to a file system specification (`FSSpec`) structure. This structure specifies the file to use as the icon data source.
`iconRef`	A pointer to a value of type `IconRef`. On return, this value contains a reference to the specified icon data.
function result	A result code. See "Icon Services Result Codes" (page 59) for a description of possible return values.

RegisterIconRefFromResource

Adds a resource-derived `IconRef` to the Icon Services registry.

```
pascal OSErr RegisterIconRefFromResource (
                OSType    creator,
                OSType    iconType,
                FSSpec*   resourceFile,
                SInt16    resourceID,
                IconRef   *iconRef);
```

`creator`	A value of type `OSType` specifying the creator code of the icon data you wish to register. You can use your application's creator code, for example. **Note** Lower-case creator codes are reserved for the system. ◆
`iconType`	A value of type `OSType` specifying the type code of the icon data you wish to register.
`resourceFile`	A pointer to a structure of type `FSSpec` specifying the resource file from which to read the icon data. For more information on the `FSSpec` structure, see *Inside Macintosh: Files*.
`resourceID`	A value of type `SInt16` specifying the resource ID of the icon data to be registered. This value must be non-zero. You should provide a resource of type `'icns'` if possible. If an `'icns'` resource is not available, Icon Services uses standard icon suite resources, such as `'ICN#'`, instead.
`iconRef`	A pointer to a value of type `IconRef`. On return, this value contains a reference to the specified icon data.
function result	A result code. See "Icon Services Result Codes" (page 59) for a description of possible return values.

DISCUSSION

You can use the `RegisterIconRefFromResource` function to register icons from `'icns'` resources or "classic" custom icon resources (`'ics#'`, `'ICN#'`, etc.). Icon Services searches `'icns'` resources before searching other icon resources.

Calling this function increments the reference count of the `IconRef`.

Remember to call the function `ReleaseIconRef` (page 47)
when you're done with an `IconRef`.

UnregisterIconRef

Removes the specified icon data from the icon registry.

```
pascal OSErr UnregisterIconRef (
                  OSType  creator,
                  OSType  iconType);
```

`creator`	A value of type `OSType` specifying the creator code of the icon data to be unregistered.
`iconType`	A value of type `OSType` specifying the type code of the icon data to be unregistered.
function result	A result code. See "Icon Services Result Codes" (page 59) for a description of possible return values.

DISCUSSION

The specified icon data is not unregistered until all its users have called the function `ReleaseIconRef` (page 47).

Note
You should not unregister an icon that you have not registered. ◆

Obtaining Icon Data

Icon Services provides the following functions to obtain icon data:

- `IsValidIconRef` (page 31)
- `IsIconRefMaskEmpty` (page 31)
- `IsIconRefComposite` (page 31)
- `GetIconSizesFromIconRef` (page 32)

IsValidIconRef

Reports whether a specified `IconRef` is valid.

```
pascal Boolean IsValidIconRef (IconRef iconRef)
```

`iconRef`	A value of type `IconRef`.
function result	`true` if the specified `IconRef` is valid, `false` otherwise.

IsIconRefMaskEmpty

Reports whether a specified mask is empty.

```
pascal Boolean IsIconRefMaskEmpty (IconRef iconRef)
```

`iconRef`	A value of type `IconRef` whose mask you wish to test.
function result	`true` if the mask associated with the given `IconRef` is empty, `false` otherwise.

IsIconRefComposite

Reports whether a specified `IconRef` has been composited.

```
pascal OSErr IsIconRefComposite(
                 IconRef   compositeIconRef,
                 IconRef   *backgroundIconRef,
                 IconRef   *foregroundIconRef)
```

`compositeIconRef`
A value of type `IconRef` that you wish to test to determine whether it has been composited.

`backgroundIconRef`
A pointer to a value of type `IconRef`. On return, this points to the `IconRef` value that forms the background of the `IconRef`

specified in the `compositeIconRef` parameter. If the `IconRef` specified in the `compositeIconRef` parameter is not a composite, the return value is 0.

`foregroundIconRef`
A pointer to a value of type `IconRef`. On return, this points to the `IconRef` value that forms the foreground of the `IconRef` specified in the `compositeIconRef` parameter. If the `IconRef` specified in the `compositeIconRef` parameter is not a composite, the return value is 0.

function result A result code. See "Icon Services Result Codes" (page 59) for a description of possible return values.

DISCUSSION

The function `CompositeIconRef` (page 38) allows the creation of a composite `IconRef` from a given background `IconRef` and a given foreground `IconRef`. The `IsIconRefComposite` function checks a specified `IconRef` to determine whether it is a composite and, if so, provides the background and foreground `IconRef` values.

GetIconSizesFromIconRef

Provides an `IconSelectorValue` indicating the sizes and depths of icon data available for an `IconRef`.

```
pascal OSErr GetIconSizesFromIconRef(
              IconSelectorValue       iconSelectorInput,
              IconSelectorValue       *iconSelectorOutputPtr,
              IconServicesUsageFlags  iconServicesUsageFlags,
              IconRef                 iconRef);
```

`iconSelectorInput` A value of type `IconSelectorValue`. You pass a value specifying the icon sizes and depths you are requesting from the `IconRef`. For a description of the possible values, see "Icon Selector Constants" (page 51).

`iconSelectorOutputPtr`
A pointer to a value of type `IconSelectorValue`. On return, this value describes the icon sizes and depths available for

	the specified `IconRef`. For a description of the possible values, see "Icon Selector Constants" (page 51).
`iconServicesUsageFlags`	Reserved for future use. Pass the `kIconServicesDefaultUsageFlags` constant in this parameter.
`iconRef`	A pointer to a value of type `IconRef` specifying the icon family to query.
function result	A result code. See "Icon Services Result Codes" (page 59) for a description of possible return values.

DISCUSSION

Note that this function may be very time-consuming, as Icon Services may have to search disks or even the network to obtain the requested data.

Using IconRef Data

Icon Services provides the following functions to use icon data:

- `PlotIconRef` (page 33)
- `PtInIconRef` (page 34)
- `RectInIconRef` (page 36)
- `IconRefToRgn` (page 37)
- `CompositeIconRef` (page 38)
- `GetIconRefVariant` (page 39)

PlotIconRef

Draws an icon using appropriate size and depth data from an `IconRef`.

```
pascal OSErr PlotIconRef(
                Rect                    *theRect,
                IconAlignmentType       align,
                IconTransformType        transform,
                IconServicesUsageFlags   iconServicesUsageFlags,
                IconRef                  iconRef);
```

`theRect`	A pointer to a value of type `Rect` specifying the rectangle where the icon is to be drawn.
`align`	A value of type `IconAlignmentType` specifying how Icon Services should align the icon within the rectangle. for a description of possible return values, see "Icon Alignment Constants" (page 50)
`transform`	A value of type `IconTransformType` specifying how Icon Services should modify the appearance of the icon.
`iconServicesUsageFlags`	Reserved for future use. Pass the `kIconServicesDefaultUsageFlags` constant in this parameter.
`iconRef`	A pointer to a value of type `IconRef` specifying the icon to draw.
function result	A result code. See "Icon Services Result Codes" (page 59) for a description of possible return values.

DISCUSSION

This function is similar to the Icon Utilities function `PlotIconSuite`. For a description of Icon Utilities functions and data structures, see *Inside Macintosh: More Macintosh Toolbox*.

PtInIconRef

Tests whether a specified point falls within an icon's mask.

```
pascal Boolean PtInIconRef(
                  Point                   *testPt,
                  Rect                    *iconRect,
                  IconAlignmentType       align,
                  IconServicesUsageFlags iconServicesUsageFlags,
                  IconRef                 iconRef);
```

`testPt`	A pointer to a value of type `Point`, specified in local coordinates of the current graphics port. This value specifies the location that Icon Services tests to see whether it falls within the mask of the indicated icon.
`iconRect`	A pointer to a value of type `Rect`. This value defines the area that Icon Services uses to determine which icon is hit-tested. Use the same `Rect` value as when the icon was last drawn.
`align`	A value of type `IconAlignmentType` that specifies how the indicated icon is aligned within the rectangle specified in the `iconRect` parameter. Use the same `IconAlignmentType` value as when the icon was last drawn. for a description of possible return values, see "Icon Alignment Constants" (page 50).
`iconServicesUsageFlags`	Reserved for future use. Pass the `kIconServicesDefaultUsageFlags` constant in this parameter.
`iconRef`	A pointer to a value of type `IconRef` specifying the icon to test.
function result	`true` if the point specified in the `testPt` parameter falls within the appropriate icon mask, `false` otherwise.

DISCUSSION

This function is similar to the Icon Utilities function `PtInIconSuite`. The function is useful when you want to determine whether a user has clicked on a particular icon, for example. For a description of Icon Utilities functions and data structures, see *Inside Macintosh: More Macintosh Toolbox*.

Note
Icon Services uses the icon's black-and-white mask for hit-testing, even if you provide a deep mask. ◆

RectInIconRef

Tests whether a specified rectangle falls within an icon's mask.

```
pascal Boolean RectInIconRef                 (
                     Rect                    *testRect,
                     Rect                    *iconRect,
                     IconAlignmentType       align,
                     IconServicesUsageFlags  iconServicesUsageFlags,
                     IconRef                 iconRef);
```

`testRect`	A pointer to a value of type `Rect`, specified in local coordinates of the current graphics port. This value specifies the rectangle that Icon Services tests to see whether it falls within the mask of the indicated icon.
`iconRect`	A pointer to a value of type `Rect`. This value defines the area that Icon Services uses to determine which icon is hit-tested. Use the same `Rect` value as when the icon was last drawn.
`align`	A value of type `IconAlignmentType` that specifies how the indicated icon is aligned within the rectangle specified in the `iconRect` parameter. Use the same `IconAlignmentType` value as when the icon was last drawn. for a description of possible return values, see "Icon Alignment Constants" (page 50).
`iconServicesUsageFlags`	Reserved for future use. Pass the `kIconServicesDefaultUsageFlags` constant in this parameter.
`iconRef`	A pointer to a value of type `IconRef` specifying the icon family to use for drawing the requested icon.
function result	`true` if the rectangle specified in the `testRect` parameter intersects the appropriate icon mask, `false` otherwise.

DISCUSSION

This function is similar to the Icon Utilities function `RectInIconSuite`. The function is useful when you want to determine whether a user selection intersects a particular icon, for example. For a description of Icon Utilities functions and data structures, see *Inside Macintosh: More Macintosh Toolbox*.

Note
Icon Services uses the icon's black-and-white mask for hit-testing, even if you provide a deep mask. ◆

IconRefToRgn

Converts an `IconRef`-derived icon into a QuickDraw region.

```
pascal OSErr IconRefToRgn(
                RgnHandle                theRgn,
                Rect                     *iconRect,
                IconAlignmentType        align,
                IconServicesUsageFlags   iconServicesUsageFlags,
                IconRef                  iconRef);
```

`theRgn`	A handle to the requested region. You must call the QuickDraw function `NewRegion` to allocate memory for the region handle before calling the `IconRefToRgn` function. For more information on the `NewRegion` function, see *Inside Macintosh: Imaging With Quickdraw*.
`iconRect`	A pointer to a value of type `Rect`. This value defines the area that Icon Services uses as the bounding box of the region.
`align`	A value of type `IconAlignmentType`. This value determines how Icon Services aligns the region within the rectangle. for a description of possible return values, see "Icon Alignment Constants" (page 50).
`iconServicesUsageFlags`	Reserved for future use. Pass the `kIconServicesDefaultUsageFlags` constant in this parameter.
`iconRef`	A pointer to a value of type `IconRef` specifying the icon family to use for drawing the requested region.
function result	A result code. See "Icon Services Result Codes" (page 59) for a description of possible return values.

DISCUSSION

Icon Services uses the rectangle and alignment values to automatically select the icon used to generate the region data.

This function is similar to the Icon Utilities function `IconSuiteToRegion`. For a description of Icon Utilities functions and data structures, see *Inside Macintosh: More Macintosh Toolbox*.

Note
Icon Services uses the icon's black-and-white mask to determine the region data, even if you provide a deep mask. ◆

CompositeIconRef

Superimposes one `IconRef` onto another.

```
pascal OSErr CompositeIconRef(
                   IconRef   backgroundIconRef,
                   IconRef   foregroundIconRef,
                   IconRef   *compositeIconRef)
```

`backgroundIconRef`	A value of type `IconRef` to use as the background for the composite `IconRef`.
`foregroundIconRef`	A value of type `IconRef` to use as the foregound for the composite `IconRef`.
`compositeIconRef`	A pointer to a value of type `IconRef`. On completion, this points to an `IconRef` that is a composite of the specified background and foreground `IconRefs`.
function result	A result code. See "Icon Services Result Codes" (page 59) for a description of possible return values.

DISCUSSION

This function provides an alternative to badging when you need to indicate a change of state.

GetIconRefVariant

Specifies a transformation for a given `IconRef`.

```
pascal IconRef GetIconRefVariant(
                    IconRef           inIconRef,
                    OSType            inVariant,
                    IconTransformType *outTransform)
```

`inIconRef`	A value of type `IconRef` to be tested.
`inVariant`	A four-character value of type `OSType`. You specify a variant by passing one of the constants defined by the enumeration described in "Icon Services Data Types" (page 58). These constants are as follows: `kTileIconVariant` specifies a tiled icon. `kRolloverIconVariant` specifies a rollover icon. `kDropIconVariant` specifies a drop target icon. `kOpenIconVariant` specifies an open icon. `kOpenDropIconVariant` specifies a open drop target icon.
`outTransform`	A pointer to a value of type `IconTransformType.` On completion, this points to a transformation type that you pass to the function `PlotIconRef` (page 33) for purposes of hit-testing.
function result	This function returns an `IconRef` value that that you pass to the function `PlotIconRef` (page 33) for purposes of hit-testing.

DISCUSSION

Icon variants give you a simple way to indicate a temporary change of state by changing an icon's appearance. For example, if you specify the `kDropIconVariant` value when the user drags over a valid drop target, the `GetIconVariant` function provides the appropriate data for you to plot the variant with the function `PlotIconRef` (page 33).

Modifying IconRef Data

Icon Services provides the following functions to modify `IconRef` data:

- `UpdateIconRef` (page 40)
- `OverrideIconRefFromResource` (page 40)
- `OverrideIconRef` (page 41)
- `RemoveIconOverride` (page 41)

UpdateIconRef

Forces an update of `IconRef` data.

```
pascal OSErr UpdateIconRef(IconRef iconRef);
```

`iconRef`	A pointer to a value of type `IconRef` to be updated.
function result	A result code. See "Icon Services Result Codes" (page 59) for a description of possible return values.

DISCUSSION

This function is useful after you have changed a file or folder's custom icon, for example. Don't call the `UpdateIconRef` function if you have not already obtained an `IconRef` for a particular icon; call the function `GetIconRefFromFile` (page 24) instead.

OverrideIconRefFromResource

Replaces the bitmaps in an `IconRef` with bitmaps from a specified resource file.

```
pascal OSErr OverrideIconRefFromResource(
                 IconRef iconRef,
                 FSSpec  *resourceFile,
                 SInt16  resourceID);
```

`iconRef`	An pointer to a value of type `IconRef` to be updated.

`resourceFile`	A pointer to an `FSSpec` data structure identifying the resource file containing the replacement bitmaps. For more information on the `FSSpec` structure, see *Inside Macintosh: Files*.
`resourceID`	A value of type `SInt16` specifying the resource ID containing the replacement bitmaps. This value must be non-zero. You should provide a resource of type `'icns'` if possible. If an `'icns'` resource is not available, Icon Services uses standard icon suite resources, such as `'ICN#'`, instead.
function result	A result code. See "Icon Services Result Codes" (page 59) for a description of possible return values.

OverrideIconRef

Replaces the bitmaps of one `IconRef` with those of another `IconRef`.

```
pascal OSErr OverrideIconRef(
                  IconRef   oldIconRef,
                  IconRef   newIconRef);
```

`oldIconRef`	A pointer to a value of type `IconRef` whose bitmaps are to be replaced.
`newIconRef`	A pointer to a value of type`IconRef` containing the replacement bitmaps.
function result	A result code. See "Icon Services Result Codes" (page 59) for a description of possible return values.

RemoveIconOverride

Restores the original bitmaps of an overridden `IconRef`.

```
pascal OSErr RemoveIconRef(IconRef iconRef);
```

`iconRef`	A pointer to a value of type`IconRef` whose bitmaps are to be restored.

function result	A result code. See "Icon Services Result Codes" (page 59) for a description of possible return values.

Reading, Copying, and Converting Icon Data

Icon Services provides the following functions to read, copy, and convert icon data:

- `IconRefToIconFamily` (page 42)
- `GetIconFamily` (page 43)
- `SetIconFamily` (page 43)
- `IconFamilyToIconSuite` (page 44)
- `IconSuiteToIconFamily` (page 45)
- `ReadIconFile` (page 45)
- `WriteIconFile` (page 46)

IconRefToIconFamily

Provides icon family data for a given `IconRef`.

```
pascal OSErr IconRefToIconFamily(
                        IconRef          iconRef,
                        IconSelectorValue whichIcons,
                        IconFamilyHandle  *iconFamily);
```

`iconRef`	A pointer to a value of type `IconRef` to use as a source for icon data.
`whichIcons`	A value of type `IconSelectorValue` specifying the depths and sizes of icons in the `iconFamily` data structure. For a description of the possible values, see "Icon Selector Constants" (page 51).
`iconFamily`	A handle to an `IconFamily` data structure. On return, this data structure contains icon data as specified in the `IconRef` and `whichIcons` parameters. Icon Services returns `NULL` if no

appropriate icon data is found. For more information on the `IconFamily` data structure, see Figure 2-1 (page 61).

function result	A result code. See "Icon Services Result Codes" (page 59) for a description of possible return values.

GetIconFamily

Obtains a copy of the data for a specified icon family.

```
pascal OSErr GetIconFamilyData(
                IconFamilyHandle  iconFamily,
                OSType            iconType,
                Handle            h)
```

`iconFamily`	A handle to an `iconFamily` data structure to use as a source for icon data.
`iconType`	A value of type `OSType` specifying the format of the icon data passed to you.You may specify one of the icon types (such as `'icns'`) or `'PICT'` in this parameter.
`h`	A handle to the icon data being returned. Icon Services resizes this handle as needed. If no data is available for the specified icon family, Icon Services sets the handle to 0.
function result	A result code. See "Icon Services Result Codes" (page 59) for a description of possible return values.

SetIconFamily

Provides new data for a specified icon family.

```
pascal OSErr SetIconFamilyData     (
                IconFamilyHandle  iconFamily,
                OSType            iconType,
                Handle            h)
```

`iconFamily`	A handle to an `iconFamily` data structure to be used as the target.

`iconType`	A value of type `OSType` specifying the format of the icon data you provide.You may specify one of the icon types (such as `'icns'`) or `'PICT'` in this parameter.
`h`	A handle to the icon data you provide.
function result	A result code. See "Icon Services Result Codes" (page 59) for a description of possible return values.

IconFamilyToIconSuite

Provides `iconSuite` data for a given icon family.

```
pascal OSErr IconFamilyToIconSuite(
                        IconFamilyHandle    iconFamily,
                        IconSelectorValue   whichIcons,
                        IconSuiteRef        *iconSuite);
```

`iconFamily`	A handle to an `iconFamily` data structure to use as a source for icon data. For more information on the `IconFamily` data structure, see Figure 2-1 (page 61).
`whichIcons`	A value of type `IconSelectorValue` specifying the depths and sizes of icons to extract from the `IconFamily` data structure. For a description of the possible values, see "Icon Selector Constants" (page 51).
`iconSuite`	A pointer to an `IconSuite` data structure. On return, this structure contains icon data as specified in the `iconFamily` and `whichIcons` parameters. Icon Services returns `NULL` if no appropriate icon data is found.
function result	A result code. See "Icon Services Result Codes" (page 59) for a description of possible return values.

IconSuiteToIconFamily

Provides `IconFamily` data for a specified `IconSuite`.

```
pascal OSErr IconSuiteToIconFamily(
                    IconSuiteRef        iconSuite,
                    IconSelectorValue   whichIcons,
                    IconFamilyHandle    *iconFamily);
```

`iconSuite`	A value of type `IconSuiteRef` to use as a source for icon data.
`whichIcons`	A value of type `IconSelectorValue` specifying the depths and sizes of icons to extract from the `iconFamily` data structure. For a description of the possible values, see "Icon Selector Constants" (page 51).
`iconFamily`	A handle to an `iconFamily` data structure. On return, this data structure contains icon data as specified in the `iconSuite` and `whichIcons` parameters. Icon Services returns `NULL` if no appropriate icon data is found. For more information on the `IconFamily` data structure, see Figure 2-1 (page 61).
function result	A result code. See "Icon Services Result Codes" (page 59) for a description of possible return values.

ReadIconFile

Copies data from a given file into an icon family.

```
pascal OSErr ReadIconFile(
                  const FSSpec      *iconFile
                  IconFamilyHandle  iconFamily)
```

`iconFile`	A pointer to a file specification structure (`FSSpec`) specifying a source file for icon data.
`iconFamily`	A handle to an `iconFamily` data structure to be used as the target. Icon Services resizes the handle as needed. For more information on the `IconFamily` data structure, see Figure 2-1 (page 61).

function result	A result code. See "Icon Services Result Codes" (page 59) for a description of possible return values.

WriteIconFile

Copies data from a given icon family into a file.

```
pascal OSErr WriteIconFile(
                 IconFamilyHandle  iconFamily
                 const FSSpec      *iconFile)
```

`iconFamily`	A handle to an `iconFamily` data structure to be used as the source for icon data. For more information on the `IconFamily` data structure, see Figure 2-1 (page 61).
`iconFile`	A pointer to a file specification structure (`FSSpec`) specifying a file to use as a target for icon data.
function result	A result code. See "Icon Services Result Codes" (page 59) for a description of possible return values.

IconRef Reference Counting

Icon Services provides the following functions to support `IconRef` reference counting:

- `GetIconRefOwners` (page 46)
- `AcquireIconRef` (page 47)
- `ReleaseIconRef` (page 47)

GetIconRefOwners

Provides the current reference count for an `IconRef`.

```
pascal OSErr GetIconRefOwners(
                  IconRef iconRef,
                  UInt16  *owners);
```

`iconRef`	A pointer to a value of type`IconRef` whose reference count you wish to obtain.
`owners`	A pointer to a value of type `UInt16`. On return, this value represents the current reference count.
function result	A result code. See "Icon Services Result Codes" (page 59) for a description of possible return values.

DISCUSSION

When an `IconRef`'s reference count reaches 0, all memory allocated for the `IconRef` is marked as disposable. Any subsequent attempt to use the `IconRef` returns a result code of `-2580` (`invalidIconRefErr`).

AcquireIconRef

Increments the reference count for an `IconRef`.

```
pascal OSErr AcquireIconRef (IconRef iconRef);
```

`iconRef`	A pointer to a value of type `IconRef` whose reference count you wish to increment.
function result	A result code. See "Icon Services Result Codes" (page 59) for a description of possible return values.

ReleaseIconRef

Decrements the reference count for an `IconRef`.

```
pascal OSErr ReleaseIconRef (IconRef iconRef);
```

`iconRef`	A pointer to a value of type `IconRef` whose reference count you wish to decrement.
function result	A result code. See "Icon Services Result Codes" (page 59) for a description of possible return values.

DISCUSSION

When an `IconRef`'s reference count reaches 0, all memory allocated for the `IconRef` is marked as disposable. Any subsequent attempt to use the `IconRef` returns a result code of -`2580` (`invalidIconRefErr`).

Flushing IconRef Data

Icon Services provides the following functions to flush `IconRef` data:

- `FlushIconRefs` (page 48)
- `FlushIconRefsByVolume` (page 48)

FlushIconRefs

Reclaims memory used by the specified icon if the memory is purgeable.

```
pascal OSErr FlushIconRefs(
                    OSType  creator,
                    OSType  iconType);
```

`creator`	A value of type `OSType` specifying the creator code of the file whose icon data is to be flushed.
`iconType`	A value of type `OSType` specifying the type code of the file whose icon data is to be flushed.
function result	A result code. See "Icon Services Result Codes" (page 59) for a description of possible return values.

FlushIconRefsByVolume

On a given volume, reclaims memory used by purgeable icons.

```
pascal OSErr FlushIconRefsByVolume(SInt16 vRefNum);
```

`vRefNum`	A value of type `SInt16` specifying the volume whose icon cache is to be flushed.

function result A result code. See "Icon Services Result Codes" (page 59) for a description of possible return values.

DISCUSSION

Calling this function locks the bitmap data of all `IconRefs` with non-zero reference counts (that is, all `IconRefs` that are in use) on the volume. The Finder normally maintains a number of `IconRefs` with non-zero reference counts, so you should use the function `FlushIconRefs` (page 48) instead of the `FlushIconRefsByVolume` function whenever feasible.

Icon Services Constants

Alert Icon Constants

Icon Services defines constants for a number of standard alert icons. You can pass one of these constants in the `iconType` parameter of the function `GetIconRef` (page 23), for example.

```
enum {
    kAlertNoteIcon                = FOUR_CHAR_CODE('note'),
    kAlertCautionIcon             = FOUR_CHAR_CODE('caut'),
    kAlertStopIcon                = FOUR_CHAR_CODE('stop')
};
```

Folder Icon Constants

Icon Services defines constants for a number of standard folder icons. You can pass one of these constants in the `iconType` parameter of the function `GetIconRef` (page 23), for example.

```
enum {
    kGenericFolderIcon                = FOUR_CHAR_CODE('fldr'),
    kDropFolderIcon                   = FOUR_CHAR_CODE('dbox'),
    kMountedFolderIcon                = FOUR_CHAR_CODE('mntd'),
    kOpenFolderIcon                   = FOUR_CHAR_CODE('ofld'),
    kOwnedFolderIcon                     = FOUR_CHAR_CODE('ownd'),
    kPrivateFolderIcon                   = FOUR_CHAR_CODE('prvf'),
    kSharedFolderIcon                    = FOUR_CHAR_CODE('shfl')
};
```

Filesharing Privilege Icon Constants

Icon Services defines constants for a number of standard filesharing privilege icons. You can pass one of these constants in the `iconType` parameter of the function `GetIconRef` (page 23), for example.

```
enum {
    kSharingPrivsNotApplicableIcon       = FOUR_CHAR_CODE('shna'),
    kSharingPrivsReadOnlyIcon            = FOUR_CHAR_CODE('shro'),
    kSharingPrivsReadWriteIcon           = FOUR_CHAR_CODE('shrw'),
    kSharingPrivsUnknownIcon             = FOUR_CHAR_CODE('shuk'),
    kSharingPrivsWritableIcon            = FOUR_CHAR_CODE('writ')
};
```

Icon Alignment Constants

The `IconAlignmentType` enumeration defines constants that allow you to specify how to align an icon within its rectangle.

```
enum {
    kAlignNone                = 0x00,
    kAlignVerticalCenter      = 0x01,
    kAlignTop                 = 0x02,
    kAlignBottom              = 0x03,
    kAlignHorizontalCenter    = 0x04,
    kAlignAbsoluteCenter      = kAlignVerticalCenter |
                                 kAlignHorizontalCenter,
    kAlignCenterTop           = kAlignTop | kAlignHorizontalCenter,
    kAlignCenterBottom        = kAlignBottom | kAlignHorizontalCenter,
    kAlignLeft                = 0x08,
    kAlignCenterLeft          = kAlignVerticalCenter | kAlignLeft,
    kAlignTopLeft             = kAlignTop | kAlignLeft,
    kAlignBottomLeft          = kAlignBottom | kAlignLeft,
    kAlignRight               = 0x0C,
    kAlignCenterRight        = kAlignVerticalCenter | kAlignRight,
    kAlignTopRight           = kAlignTop | kAlignRight,
    kAlignBottomRight        = kAlignBottom | kAlignRight
};
```

Icon Transformation Constants

The `IconTransformType` enumeration defines values that Icon Services uses to report how an icon has been transformed after you call the function `GetIconRefVariant` (page 39).

```
enum {
    kTransformNone              = 0x00,
    kTransformDisabled          = 0x01,
    kTransformOffline           = 0x02,
    kTransformOpen              = 0x03,
    kTransformLabel1            = 0x0100,
    kTransformLabel2            = 0x0200,
    kTransformLabel3            = 0x0300,
    kTransformLabel4            = 0x0400,
    kTransformLabel5            = 0x0500,
    kTransformLabel6            = 0x0600,
    kTransformLabel7            = 0x0700,
    kTransformSelected          = 0x4000,
    kTransformSelectedDisabled  = kTransformSelected |
                                   kTransformDisabled,
    kTransformSelectedOffline   = kTransformSelected | kTransformOffline,
    kTransformSelectedOpen      = kTransformSelected | kTransformOpen
};
```

Icon Selector Constants

The `IconSelectorValue` enumeration describes values that you can use to obtain information about the sizes and depths of icons available in a given icon family.

```
enum {
    kSelectorLarge1Bit          = 0x00000001,
    kSelectorLarge4Bit          = 0x00000002,
    kSelectorLarge8Bit          = 0x00000004,
    kSelectorLarge32Bit          = 0x00000008,
    kSelectorLarge8BitMask       = 0x00000010,
    kSelectorSmall1Bit           = 0x00000100,
    kSelectorSmall4Bit           = 0x00000200,
    kSelectorSmall8Bit           = 0x00000400,
    kSelectorSmall32Bit          = 0x00000800,
    kSelectorSmall8BitMask       = 0x00001000,
    kSelectorMini1Bit            = 0x00010000,
    kSelectorMini4Bit            = 0x00020000,
    kSelectorMini8Bit            = 0x00040000,
    kSelectorHuge1Bit            = 0x01000000,
    kSelectorHuge4Bit            = 0x02000000,
    kSelectorHuge8Bit            = 0x04000000,
    kSelectorHuge32Bit           = 0x08000000,
    kSelectorHuge8BitMask        = 0x10000000,
    kSelectorAllLargeData        = 0x000000FF,
    kSelectorAllSmallData        = 0x0000FF00,
    kSelectorAllMiniData         = 0x00FF0000,
    kSelectorAllHugeData         = (long)0xFF000000,
    kSelectorAll1BitData         = kSelectorLarge1Bit | kSelectorSmall1Bit
                                 | kSelectorMini1Bit | kSelectorHuge1Bit,
    kSelectorAll4BitData         = kSelectorLarge4Bit | kSelectorSmall4Bit
                                 | kSelectorMini4Bit | kSelectorHuge4Bit,
    kSelectorAll8BitData         = kSelectorLarge8Bit | kSelectorSmall8Bit
                                 | kSelectorMini8Bit | kSelectorHuge8Bit,
    kSelectorAll32BitData        = kSelectorLarge32Bit
                                 | kSelectorSmall32Bit|kSelectorHuge32Bit,
    kSelectorAllAvailableData    = (long)0xFFFFFFFF
};
```

Icon Variant Constants

Icon variants are transformations of standard icons. You can use variants as elements of an icon family or pass them in the `inVariant` parameter of the function `GetIconRefVariant` (page 39).

```
enum {
    kTileIconVariant                = FOUR_CHAR_CODE('tile'),
    kRolloverIconVariant            = FOUR_CHAR_CODE('over'),
    kDropIconVariant                = FOUR_CHAR_CODE('drop'),
    kOpenIconVariant                    = FOUR_CHAR_CODE('open'),
    kOpenDropIconVariant                = FOUR_CHAR_CODE('odrp')
};
```

Internet Icon Constants

Icon Services defines constants for a number of standard Internet icons. You can pass one of these constants in the `iconType` parameter of the function `GetIconRef` (page 23), for example.

```
enum {
    kInternetLocationHTTPIcon           = FOUR_CHAR_CODE('ilht'),
    kInternetLocationFTPIcon            = FOUR_CHAR_CODE('ilft'),
    kInternetLocationAppleShareIcon     = FOUR_CHAR_CODE('ilaf'),
    kInternetLocationAppleTalkZoneIcon  = FOUR_CHAR_CODE('ilat'),
    kInternetLocationFileIcon           = FOUR_CHAR_CODE('ilfi'),
    kInternetLocationMailIcon           = FOUR_CHAR_CODE('ilma'),
    kInternetLocationNewsIcon           = FOUR_CHAR_CODE('ilnw'),
    kInternetLocationGenericIcon        = FOUR_CHAR_CODE('ilge')
};
```

Miscellaneous Icon Constants

Icon Services defines constants for a number of miscellaneous icons. You can pass one of these constants in the `iconType` parameter of the function `GetIconRef` (page 23), for example.

```
enum {
    kAppleLogoIcon                  = FOUR_CHAR_CODE('capl'),
    kAppleMenuIcon                  = FOUR_CHAR_CODE('sapl'),
    kBackwardArrowIcon              = FOUR_CHAR_CODE('baro'),
    kFavoriteItemsIcon              = FOUR_CHAR_CODE('favr'),
    kForwardArrowIcon               = FOUR_CHAR_CODE('faro'),
    kGridIcon                       = FOUR_CHAR_CODE('grid'),
    kHelpIcon                       = FOUR_CHAR_CODE('help'),
    kKeepArrangedIcon               = FOUR_CHAR_CODE('arng'),
    kLockedIcon                     = FOUR_CHAR_CODE('lock'),
    kNoFilesIcon                    = FOUR_CHAR_CODE('nfil'),
    kNoFolderIcon                   = FOUR_CHAR_CODE('nfld'),
    kNoWriteIcon                    = FOUR_CHAR_CODE('nwrt'),
    kProtectedApplicationFolderIcon = FOUR_CHAR_CODE('papp'),
    kProtectedSystemFolderIcon      = FOUR_CHAR_CODE('psys'),
    kRecentItemsIcon                = FOUR_CHAR_CODE('rcnt'),
    kShortcutIcon                   = FOUR_CHAR_CODE('shrt'),
    kSortAscendingIcon              = FOUR_CHAR_CODE('asnd'),
    kSortDescendingIcon             = FOUR_CHAR_CODE('dsnd'),
    kUnlockedIcon                   = FOUR_CHAR_CODE('ulck'),
    kConnectToIcon                  = FOUR_CHAR_CODE('cnct')
};
```

Networking Icon Constants

Icon Services defines constants for a number of standard networking icons. You can pass one of these constants in the `iconType` parameter of the function `GetIconRef` (page 23), for example.

```
enum {
    kAppleTalkIcon                      = FOUR_CHAR_CODE('atlk'),
    kAppleTalkZoneIcon                  = FOUR_CHAR_CODE('atzn'),
    kAFPServerIcon                      = FOUR_CHAR_CODE('afps'),
    kFTPServerIcon                      = FOUR_CHAR_CODE('ftps'),
    kGenericNetworkIcon                 = FOUR_CHAR_CODE('gnet'),
    kIPFileServerIcon                   = FOUR_CHAR_CODE('isrv')
};
```

Special Folder Icon Constants

Icon Services defines constants for a number of special folder icons. You can pass one of these constants in the `iconType` parameter of the function `GetIconRef` (page 23), for example.

```
enum {
    kAppleExtrasFolderIcon              = FOUR_CHAR_CODE('aexƒ'),
    kAppleMenuFolderIcon                = FOUR_CHAR_CODE('amnu'),
    kApplicationsFolderIcon             = FOUR_CHAR_CODE('apps'),
    kApplicationSupportFolderIcon       = FOUR_CHAR_CODE('asup'),
    kAssistantsFolderIcon               = FOUR_CHAR_CODE('astƒ'),
    kContextualMenuItemsFolderIcon      = FOUR_CHAR_CODE('cmnu'),
    kControlPanelDisabledFolderIcon     = FOUR_CHAR_CODE('ctrD'),
    kControlPanelFolderIcon             = FOUR_CHAR_CODE('ctrl'),
    kControlStripModulesFolderIcon      = FOUR_CHAR_CODE('sdvƒ'),
    kDocumentsFolderIcon                = FOUR_CHAR_CODE('docs'),
    kExtensionsDisabledFolderIcon       = FOUR_CHAR_CODE('extD'),
    kExtensionsFolderIcon               = FOUR_CHAR_CODE('extn'),
    kFavoritesFolderIcon                = FOUR_CHAR_CODE('favs'),
    kFontsFolderIcon                    = FOUR_CHAR_CODE('font'),
    kHelpFolderIcon                     = FOUR_CHAR_CODE('ƒhlp'),
    kInternetFolderIcon                 = FOUR_CHAR_CODE('intƒ'),
    kInternetPlugInFolderIcon           = FOUR_CHAR_CODE('ƒnet'),
    kLocalesFolderIcon                  = FOUR_CHAR_CODE('ƒloc'),
    kMacOSReadMeFolderIcon              = FOUR_CHAR_CODE('morƒ'),
    kPreferencesFolderIcon              = FOUR_CHAR_CODE('prfƒ'),
    kPrinterDescriptionFolderIcon       = FOUR_CHAR_CODE('ppdf'),
    kPrinterDriverFolderIcon            = FOUR_CHAR_CODE('ƒprd'),
    kPrintMonitorFolderIcon             = FOUR_CHAR_CODE('prnt'),
    kRecentApplicationsFolderIcon       = FOUR_CHAR_CODE('rapp'),
    kRecentDocumentsFolderIcon          = FOUR_CHAR_CODE('rdoc'),
    kRecentServersFolderIcon            = FOUR_CHAR_CODE('rsrv'),
    kScriptingAdditionsFolderIcon       = FOUR_CHAR_CODE('ƒscr'),
    kSharedLibrariesFolderIcon          = FOUR_CHAR_CODE('ƒlib'),
    kScriptsFolderIcon                  = FOUR_CHAR_CODE('scrƒ'),
    kShutdownItemsDisabledFolderIcon    = FOUR_CHAR_CODE('shdD'),
    kShutdownItemsFolderIcon            = FOUR_CHAR_CODE('shdf'),
    kSpeakableItemsFolder               = FOUR_CHAR_CODE('spki'),
    kStartupItemsDisabledFolderIcon     = FOUR_CHAR_CODE('strD'),
    kStartupItemsFolderIcon             = FOUR_CHAR_CODE('strt'),
    kSystemExtensionDisabledFolderIcon  = FOUR_CHAR_CODE('macD'),
    kSystemFolderIcon                   = FOUR_CHAR_CODE('macs'),
    kTextEncodingsFolderIcon            = FOUR_CHAR_CODE('ƒtex'),
    kAppearanceFolderIcon               = FOUR_CHAR_CODE('appr'),
    kUtilitiesFolderIcon                = FOUR_CHAR_CODE('utiƒ'),
    kVoicesFolderIcon                   = FOUR_CHAR_CODE('fvoc'),
    kColorSyncFolderIcon                = FOUR_CHAR_CODE('prof'),
    kInternetSearchSitesFolderIcon      = FOUR_CHAR_CODE('issf'),
    kUsersFolderIcon                    = FOUR_CHAR_CODE('usrƒ')
};
```

Standard Finder Icon Constants

Icon Services defines constants for a number of standard Finder icons. You can pass one of these constants in the `iconType` parameter of the function `GetIconRef` (page 23), for example.

```
enum {
    kClipboardIcon                      = FOUR_CHAR_CODE('CLIP'),
    kClippingUnknownTypeIcon            = FOUR_CHAR_CODE('clpu'),
    kClippingPictureTypeIcon            = FOUR_CHAR_CODE('clpp'),
    kClippingTextTypeIcon               = FOUR_CHAR_CODE('clpt'),
    kClippingSoundTypeIcon              = FOUR_CHAR_CODE('clps'),
    kDesktopIcon                        = FOUR_CHAR_CODE('desk'),
    kFinderIcon                         = FOUR_CHAR_CODE('FNDR'),
    kFontSuitcaseIcon                   = FOUR_CHAR_CODE('FFIL'),
    kFullTrashIcon                      = FOUR_CHAR_CODE('ftrh'),
    kGenericApplicationIcon             = FOUR_CHAR_CODE('APPL'),
    kGenericCDROMIcon                   = FOUR_CHAR_CODE('cddr'),
    kGenericControlPanelIcon            = FOUR_CHAR_CODE('APPC'),
    kGenericControlStripModuleIcon      = FOUR_CHAR_CODE('sdev'),
    kGenericComponentIcon               = FOUR_CHAR_CODE('thng'),
    kGenericDeskAccessoryIcon           = FOUR_CHAR_CODE('APPD'),
    kGenericDocumentIcon                = FOUR_CHAR_CODE('docu'),
    kGenericEditionFileIcon             = FOUR_CHAR_CODE('edtf'),
    kGenericExtensionIcon               = FOUR_CHAR_CODE('INIT'),
    kGenericFileServerIcon              = FOUR_CHAR_CODE('srvr'),
    kGenericFontIcon                    = FOUR_CHAR_CODE('ffil'),
    kGenericFontScalerIcon              = FOUR_CHAR_CODE('sclr'),
    kGenericFloppyIcon                  = FOUR_CHAR_CODE('flpy'),
    kGenericHardDiskIcon                = FOUR_CHAR_CODE('hdsk'),
    kGenericRemovableMediaIcon          = FOUR_CHAR_CODE('rmov'),
    kGenericMoverObjectIcon             = FOUR_CHAR_CODE('movr'),
    kGenericPCCardIcon                  = FOUR_CHAR_CODE('pcmc'),
    kGenericPreferencesIcon             = FOUR_CHAR_CODE('pref'),
    kGenericQueryDocumentIcon           = FOUR_CHAR_CODE('qery'),
    kGenericRAMDiskIcon                 = FOUR_CHAR_CODE('ramd'),
    kGenericSharedLibaryIcon            = FOUR_CHAR_CODE('shlb'),
    kGenericStationeryIcon              = FOUR_CHAR_CODE('sdoc'),
    kGenericSuitcaseIcon                = FOUR_CHAR_CODE('suit'),
    kGenericWORMIcon                    = FOUR_CHAR_CODE('worm'),
    kInternationResourcesIcon           = FOUR_CHAR_CODE('ifil'),
    kKeyboardLayoutIcon                  = FOUR_CHAR_CODE('kfil'),
    kSoundFileIcon                       = FOUR_CHAR_CODE('sfil'),
    kSystemSuitcaseIcon                  = FOUR_CHAR_CODE('zsys'),
    kTrashIcon                           = FOUR_CHAR_CODE('trsh'),
    kTrueTypeFontIcon                    = FOUR_CHAR_CODE('tfil'),
    kTrueTypeFlatFontIcon                = FOUR_CHAR_CODE('sfnt'),
    kTrueTypeMultiFlatFontIcon           = FOUR_CHAR_CODE('ttcf')
};
```

Standard Icon Badge Constants

Icon Services defines constants for a number of standard badges. You can pass one of these constants in the `iconType` parameter of the function `GetIconRef` (page 23), for example. For a description of badging, see "Using Badges" (page 13).

```
enum {
    kAppleScriptBadgeIcon                = FOUR_CHAR_CODE('scrp'),
    kLockedBadgeIcon                     = FOUR_CHAR_CODE('lbdg'),
    kMountedBadgeIcon                    = FOUR_CHAR_CODE('mbdg'),
    kSharedBadgeIcon                     = FOUR_CHAR_CODE('sbdg'),
    kAliasBadgeIcon                      = FOUR_CHAR_CODE('abdg')
};
```

System Icon Constant

You can use the `kSystemIconsCreator` constant to obtain System icons that are not associated with a file, such as the help icon. For an example of using this constant, see "Obtaining an IconRef for the standard help icon" (page 11).

```
enum {
    kSystemIconsCreator                  = FOUR_CHAR_CODE('macs')
};
```

Users and Groups Icon Constants

Icon Services defines constants for a number of icons used in the Users and Groups control panel. You can pass one of these constants in the `iconType` parameter of the function `GetIconRef` (page 23), for example.

```
enum {
    kUserFolderIcon                         = FOUR_CHAR_CODE('ufld'),
    kWorkgroupFolderIcon                    = FOUR_CHAR_CODE('wfld'),
    kGuestUserIcon                          = FOUR_CHAR_CODE('gusr'),
    kUserIcon                               = FOUR_CHAR_CODE('user'),
    kOwnerIcon                              = FOUR_CHAR_CODE('susr'),
    kGroupIcon                              = FOUR_CHAR_CODE('grup')
};
```

Icon Services Data Types

IconFamilyElement

Contains data describing individual icon types obtained from an `'icns'` resource.

```
struct IconFamilyElement {
    OSType          elementType;    /* 'ICN#', 'icl8', etc...*/
    Size            elementSize;    /* Size of this element*/
    unsigned char   elementData[1];
};

typedef struct IconFamilyElement  IconFamilyElement;
```

Field descriptions

`elementType`	A value of type `OSType`. This four-character code specifies which type of icon resource (`icl8`, for example) is described by this particular element.

`elementSize`	A value of type `Size`. This value specifies the size of the data contained in the `elementData` field plus 8 bytes; that is, the total size of the element.
`elementData`	An array of values of type `char`. These values define the icon family element specified in this structure.

IconFamilyResource

Contains data obtained from an `'icns'` resource.

```
struct IconFamilyResource {
OSType              resourceType;    /* Always 'icns'*/
Size                resourceSize;    /* Total size of this resource*/
IconFamilyElement   elements[1];
};
```

Field descriptions

`resourceType`	A value of type `OSType`. This is always `'icns'`.
`resourceSize`	A value of type `Size`. This value specifies the total size of this resource.
`elements`	An array of values of type `IconFamilyElement`. These values define the icon family described by this structure.

Icon Services Result Codes

All Icon Services functions return result codes, which are listed here.

`noErr`	0	No error has occurred.
`invalidIconRefErr`	-2580	An invalid `IconRef` was specified.
`noSuchIconErr`	-2581	The requested icon could not be found.
`noIconDataAvailableErr`	-2582	No data is available for the requested icon.

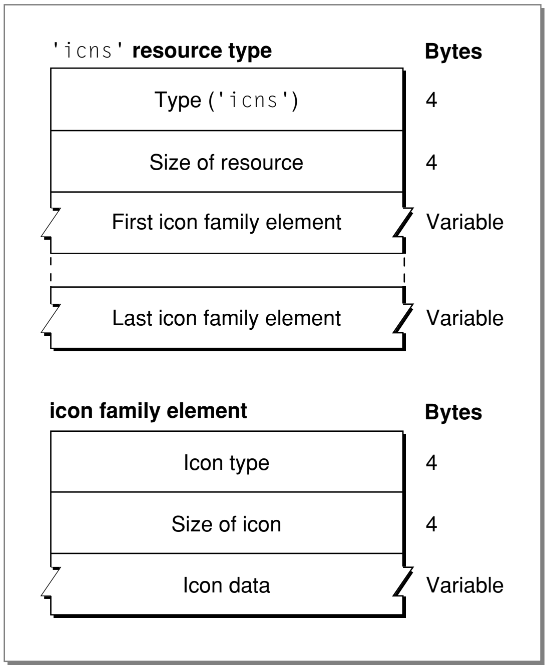
Icon Services Resources

'icns'

The `'icns'` resource type contains data describing an entire icon family (all sizes and depths). If you specify a custom icon, Icon Services checks for an appropriate `'icns'` resource before it checks individual custom icon resources (`'ics#'`, for example). If Icon Services finds an appropriate `'icns'` resource, it obtains all icon data exclusively from that resource. In order to avoid incompatibilities with older versions of Finder, new icon features such as 32-bit deep icons are only obtained from the `'icns'` resource.

To determine the driver icon for a particular device, Icon Services calls `DriverGestalt` with the `kdgMediaIconSuite` constant after calling the File System Manager. The `DriverGestalt` call returns a pointer to an icon family.

Figure 2-1 Structure of an icon family ('icns') resource



A compiled version of an `'icns'` resource contains the following elements:

- Type. A four-character code that identifies the resource. This value is always `'icns'`.
- Size of resource. A long integer that specifies the size of the resource in bytes.
- One or more icon family elements. These elements contain the following sub-elements:
  - Type. A four-character code that identifies the icon type, such as `'icl8'`.
  - Size of icon. A long integer that specifies the size of the element in bytes.
  - The icon data.

'badg'

The `'badg'` resource contains information about badges that your application can specify to overlay or replace other icons.

```
struct CustomBadgeResource
{
UInt16 version;                 // This is version 0

SInt16 customBadgeResourceID;   // If not 0, ID of resource to use
                                // on top of icon for this file or folder

OSType customBadgeType;         // If not 0, type and creator of icon
OSType customBadgeCreator;      // to use on top of existing icon

OSType windowBadgeType;         // If not 0, type and creator of icon
OSType windowBadgeCreator;      // to display in window header
                                // for this file or folder

OSType overrideType;            // If not 0, type and creator of icon to
OSType overrideCreator;         // use INSTEAD of the icon for this file
                                // or folder
};
```

Field descriptions

`version`	An unsigned integer value. This is currently set to 0, but may change in the future.
`customBadgeResourceID`	A signed integer value. This value specifies a custom icon resource to use as an overlay for the icon associated with this file or folder. The icon resource should be in the same file as the `'badg'` resource.
`customBadgeType`	A value of type `OSType`. This four-character value specifies the type code of a registered icon to use as an overlay for the regular icon for this file or folder. If you do not wish to use an overlay, set the value of the `customBadgeType` field to 0.
`customBadgeCreator`	A value of type `OSType`. This four-character value specifies the creator code of a registered icon to use as an overlay for the regular icon for this file or folder. If you do not wish to

	use an overlay, set the value of the `customBadgeCreator` field to 0.
`windowBadgeType`	A value of type `OSType`. This four-character value specifies the type code for a registered icon to display in the window header of a Finder window displaying this object. If you do not wish to specify a window header icon, set the value of the `windowBadgeType` field to 0.
`windowBadgeCreator`	A value of type `OSType`. This four-character value specifies the creator code for a registered icon to display in the window header of a Finder window displaying this object. If you do not wish to specify a window header icon, set the value of the `windowBadgeCreator` field to 0.
`overrideType`	A value of type `OSType`. This four-character value specifies the type code for a registered icon to replace the standard icon for this object. If you do not wish to specify a replacement icon, set the value of the `overrideType` field to 0.
`overrideCreator`	A value of type `OSType`. This four-character value specifies the creator code for a registered icon to replace the standard icon for this object. If you do not wish to specify a replacement icon, set the value of the `overrideCreator` field to 0.

Glossary

Appearance Manager The part of Mac OS system software that provides additional user interface elements and options.

Appearance-compliant Software that is written to work with the Appearance Manager.

badge An overlay for an icon. Badges indicate a change of state for an object without a change in type; for example, a folder could be badged to indicate that it contains Applescripts.

deep mask An icon mask with 256 levels of transparency.

desktop A location composed of the startup volume's desktop folder plus the icons of all other mounted volumes.

file object A file, folder or volume.

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

icon variants A set of optional transformations for standard icons.

IconRef An opaque value representing a particular set of icon data.

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

opaque value An object that references a particular type and instance of data. An `IconRef` is an example of an opaque value.

override The process of temporarily changing the data for an `IconRef`.

reference counting The process of tracking how many clients are using a particular object (such as an `IconRef`.)

registered icon An icon whose data is stored in the icon cache and therefore is readily available.

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

Index

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

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

WRITER
Otto Schlosser

DEVELOPMENTAL EDITOR
Donna S. Lee

ILLUSTRATOR
David Arrigoni

PRODUCTION EDITOR
Lorraine Findlay

ACKNOWLEDGMENTS
Tony Francis, Pete Gontier, Arno Goudol, Deborah Grits, Eric Simenel