CHAPTER 5

Icon Utilities

Contents

This chapter describes how your application can use the Icon Utilities to draw icons, including small, large, black-and-white, and color icons. The Finder draws and manages the icons that a user sees on the desktop, but if your application needs to display icons within its windows, it can use Icon Utilities routines to draw them.

For information on how to create icons and associate them with your application and its document, see the chapter "Finder Interface" in *Inside Macintosh: Macintosh Toolbox Essentials*. For information on how to design icons, see the chapter "Icons" in *Macintosh Human Interface Guidelines*.

This chapter begins with a brief overview of the various kinds of icons you can provide. The rest of the chapter describes how you can draw each kind of icon.

Introduction to the Icon Utilities

An **icon** on a Macintosh screen is an image that graphically represents some object, such as a file, a folder, or the Trash. On the desktop, the Finder displays icons representing your application and the documents it creates. The Finder also allows users to manipulate icons on the desktop and in folders.

If necessary, your application can also display icons in its menus, dialog boxes, or windows. You define an icon for a menu item by providing the icon's icon number in the `'MENU'` resource that describes the menu item. If you define an icon for a menu item in this manner, the Menu Manager automatically displays the icon whenever you display the menu using the `MenuSelect` function.

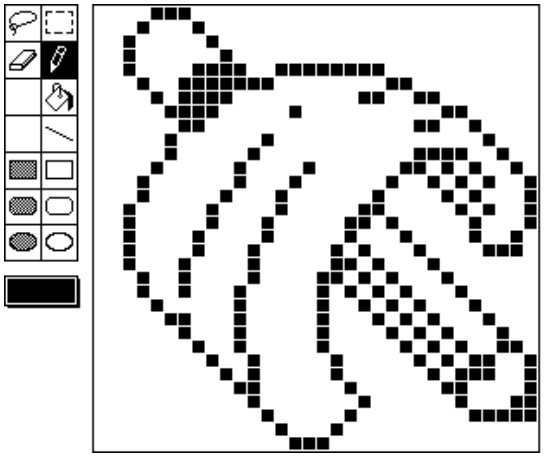
You usually define icons in dialog boxes by defining an item of type `Icon` and providing the resource ID of the icon in the item list (`'DITL'`) resource that describes the dialog. If you define an icon for a dialog item in this manner, the Dialog Manager automatically displays the icon whenever you display the dialog box using Dialog Manager routines.

Both the Menu Manager and Dialog Manager allow you to display icons of resource type `'ICON'` or `'cicn'`. The Menu Manager also allows you to display icons of resource type `'SICN'`. To display other types of icons in your menu items, you can write your own menu definition procedure and use the routines described in this chapter to draw the icons. To display other types of icons in your dialog items, define items of type `userItem` and use the routines in this chapter to draw your icons.

To display icons of any kind in your windows, use Icon Utilities routines. Icons in windows can be useful for representing files and folders in certain applications, such as archiving applications, groupware, and electronic mail applications. Other programs, such as games, might allow users to move or manipulate icons in windows for a variety of purposes.

Whenever you design an icon, you should generally begin by creating a black-and-white icon and then add color using the resource types that define color icons. Typically you use a high-level tool such as the ResEdit application to design icons. Figure 5-1 shows the ResEdit view of a black-and-white icon. When you are satisfied with the appearance of your icons, you can use the DeRez decompiler to convert them into Rez input.

Figure 5-1 The ResEdit view of an icon



For more information about designing and creating icons, see *Macintosh Human Interface Guidelines* and the chapter "Finder Interface" in *Inside Macintosh: Macintosh Toolbox Essentials*.

To display an icon most effectively at different sizes and on display devices with different bit depths, you should create an icon family for each icon you wish to use. An **icon family** is the set of icons that represent a single object. An entire icon family consists of large (32-by-32 pixel) and small (16-by-16 pixel) icons, each with a mask, and each available in three different versions of color: black and white, 4 bits of color data per pixel, and 8 bits of color data per pixel. Specifically, the following icons make up the icon family for a single icon:

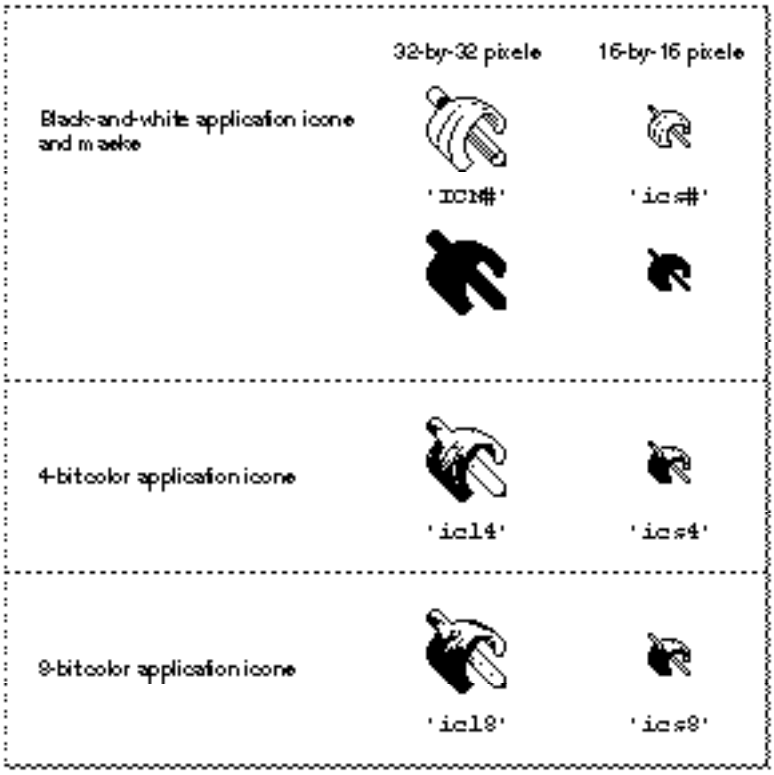
- a large (32-by-32 pixel) black-and-white icon and mask—both of which you define in an icon list (`'ICN#'`) resource
- a small (16-by-16 pixel) black-and-white icon and mask—both of which you define in a small icon list (`'ics#'`) resource
- a large (32-by-32 pixel) color icon with 4 bits of color data per pixel—which you define in a large 4-bit color icon (`'icl4'`) resource

- a small (16-by-16 pixel) color icon with 4 bits of color data per pixel—which you define in a small 4-bit color icon (`'ics4'`) resource
- a large (32-by-32 pixel) color icon with 8 bits of color data per pixel—which you define in a large 8-bit color icon (`'icl8'`) resource
- a small (16-by-16 pixel) color icon with 8 bits of color data per pixel—which you define in a small 8-bit color icon (`'ics8'`) resource

An icon family can contain only one icon of each resource type listed.

Figure 5-2 shows the icon family for the icon that represents the SurfWriter application. To see these icons in color, see Plate 3 in *Inside Macintosh: Macintosh Toolbox Essentials*.

Figure 5-2 An icon family



Somewhat related to these resources are the icon (`'ICON'`) resource and the color icon (`'cicn'`) resource. You can use either to describe a 32-by-32 pixel icon within some element of your application. As previously discussed, both the Menu Manager and Dialog Manager allow you to display icons with the resource type `'ICON'` or `'cicn'`, and the Menu Manager also allows you to display icons of resource type `'SICN'`. These are the only kinds of icons you can use in menu items and dialog boxes if you want the Menu Manager and Dialog Manager to display the icons automatically for you. If you provide a color icon (`'cicn'`) resource with the same resource ID as an icon (`'ICON'`) resource, the Menu Manager and the Dialog Manager display the color icon instead of the black-and-white icon.

The icon (`'ICON'`) resource contains a bitmap for a 32-by-32 pixel black-and-white icon. Because it is always displayed on a white background, and never in the Finder, it doesn't need a mask.

The color icon (`'cicn'`) resource has a special format that includes a pixel map, a bitmap, and a mask. You can use it to define a color icon of any size without a mask or a 32-by-32 pixel color icon with a mask. You can also define the bit depth for a color icon resource. For information about the format of a `'cicn'` resource, see *Inside Macintosh: Imaging With QuickDraw*.

Many of the icons in the System file are available in a small size; these icons are stored in `'SICN'` resources. The icons in an `'SICN'` resource are 12 by 16 pixels, even though they are stored in the resource as 16-by-16 pixel bitmaps. An `'SICN'` resource consists of a list of 16-by-16 pixel bitmaps for black-and-white icons; by convention, the list includes only two bitmaps, and the second bitmap is considered a mask. The Menu Manager lets you use an `'SICN'` resource as an icon in a menu item; however, you cannot use the Dialog Manager to display an `'SICN'` icon in a dialog box.

The Finder does *not* use or display any resources that you create of type `'ICON'`, `'cicn'`, or `'SICN'`. To create an icon for display by the Finder, create one or more of the icons in an icon family.

About the Icon Utilities

The Icon Utilities allow your application (and system software) to manipulate and draw icons of any standard resource type in windows and if necessary in menus or dialog boxes. You need to use these routines only if you wish to draw icons in your application's windows or to draw icons whose resource types are not recognized by the Menu Manager and Dialog Manager in menus and dialog boxes.

To display an icon most effectively at a variety of sizes and bit depths, you should provide an icon family. You can then draw the appropriate member of the family for a given size and bit depth either by passing the family's resource ID to an Icon Utilities routine or by reading the family's icon resources into memory as an icon suite and passing the suite's handle to Icon Utilities routines.

The next section, "Using the Icon Utilities," begins by describing how to draw icons in an icon family. After a brief overview of icon families, icon suites, icon caches, and related Icon Utilities routines, it describes in detail how to

- draw the most appropriate icon for a given destination rectangle and bit depth directly from an icon family member's resource
- get an icon suite and draw the most appropriate icon from that suite for a given destination rectangle and bit depth
- draw specific icons from an icon family or suite
- get a handle to an icon suite member's icon data so you can manipulate it
- draw icons that are not part of an icon family

You can use also Icon Utilities routines to

- perform operations on icons in an icon suite
- manipulate labels associated with specific icon suites
- dispose of icon suites and color icon records
- convert an icon mask to a region and perform hit-testing for an icon
- create an icon cache by associating an icon suite with an icon getter function and a pointer to data that you can use as a reference constant

For detailed descriptions of all Icon Utilities routines, including those used to perform these tasks, see "Icon Utilities Reference" beginning on page 5-17.

In addition to the resource types described earlier in this chapter, some Icon Utilities routines operate on icons of resource types `'icm#'`, `'icm4'`, and `'icm8'`. These **mini icons** are 12-by-16 pixel icons. Like the icons in an icon family, the three resource types for mini icons identify the icon list, 4-bit color icons, and 8-bit color icons, respectively.

Using the Icon Utilities

This section explains how you can use routines in the Icon Utilities to draw icons in your application's windows (or dialog boxes and menu items if needed).

Most of the Icon Utilities routines are available only in System 7 and later. To determine whether they are available, call the `Gestalt` function with the `gestaltIconUtilitiesAttr` selector and check the value of the `response` parameter. If the bit indicated by the constant `gestaltIconUtilitiesPresent` is set, then the Icon Utilities are available.

```
CONST
gestaltIconUtilitiesAttr      = 'icon';   {Icon Utils attributes}
gestaltIconUtilitiesPresent   = 0;        {check this bit in the }
                                          { response parameter}
```

The `GetIcon`, `PlotIcon`, `GetCIcon`, `PlotCIcon`, and `DisposeCIcon` routines are available in both System 6 and System 7.

Drawing Icons in an Icon Family

You can define different versions of an icon for specific sizes and bit depths as part of a single icon family whose members share the same resource ID. If you define all your application's icons in icon families, you can use Icon Utilities routines to draw the icon using the icon family member that is best suited for the destination rectangle and the current bit depth of the display device. When your application uses Icon Utilities routines like `PlotIconSuite` or `PlotIconID` to plot icons, it doesn't have to determine which icon in the icon family is best suited for a given destination rectangle and bit depth; instead, the routines automatically display the appropriate icon.

You can also define individual icons of resource type `'ICON'`, `'cicn'`, or `'SICN'` that are not part of an icon family and use Icon Utilities routines to draw them when necessary. For information about drawing these types of icons, see the section "Drawing Icons That Are Not Part of an Icon Family" beginning on page 5-13.

You can use the Icon Utilities to draw icons using modes or **transforms** that alter the icon's appearance in standard ways that are analogous to Finder states for icons. For example, the Finder draws a selected icon differently than it draws one that is not selected; to do so, the Finder specifies the transform constant `ttSelected` when it calls Icon Utilities routines to draw a selected icon. If you need to apply a particular transform to an icon, some Icon Utilities routines allow you to apply transforms for both standard Finder states and Finder label colors when you draw the icon.

Many of the Icon Utilities routines can also automatically align an icon within its destination rectangle. For example, the generic document icon that appears in the Finder is taller than it is wide. Some Icon Utilities routines allow you to draw such an icon without any special alignment, align it at the left or right of the destination rectangle, or use various other alignments.

Depending on the size of the rectangle, the Icon Utilities routines may stretch or shrink the icon to fit. To draw icons without stretching them, these routines require that the destination rectangle have the exact dimensions of a standard icon: that is, depending on the icon resource type, 32 by 32 pixels, 16 by 16 pixels, or 12 by 16 pixels. If you use destination rectangles of other sizes, these routines stretch or shrink the icons to fit the rectangles.

An icon family is a collection of icons representing a single object. Each icon in the family shares the same resource ID as other icons in the family but has its own resource type identifying the icon data it contains. The simplest way to draw an icon from an icon family is to pass the family's resource ID to the `PlotIconID` function, which draws the appropriate icon from the family for the specified destination rectangle and bit depth. The next section, "Drawing an Icon Directly From a Resource," describes how to use `PlotIconID`.

Alternatively, you can first use the `GetIconSuite` function to read the resource data for some or all icons in an icon family into memory. Given a resource ID and one or more resource types, the `GetIconSuite` function reads in resource data for each icon with the specified resource ID and resource types and collects handles to the resource data in an icon suite. An **icon suite** typically consists of one or more handles to icon resources from a single icon family that have been read into memory. The `GetIconSuite` function returns a handle for the requested icon suite; you can pass this handle to `PlotIconSuite` and other Icon Utilities routines. Like `PlotIconID`, `PlotIconSuite` draws the appropriate icon from an icon suite for the specified destination rectangle and bit depth. The section "Getting an Icon Suite and Drawing One of Its Icons" on page 5-11 describes how to use the `GetIconSuite` and `PlotIconSuite` routines.

An icon suite can in turn contain handles to each of the six icon resources that an icon family can contain, or it can contain handles to only a subset of the icon resources in an icon family. However, for best results, an icon suite should always include a black-and-white icon and icon mask for any icons you provide; that is, it should include a resource of type `'ICN#'` in addition to any other large icons you provide as well as a resource of type `'ics#'` in addition to any other small icons you provide. When you create an icon suite from icon family resources, the associated resource file should remain open while you use Icon Utilities routines.

Two types of handles exist in an icon suite: handles to icon data associated with a resource and handles to icon data that isn't associated with a resource. You typically use `GetIconSuite` to fill an icon suite with handles to icon resource data. You typically use `AddIconToSuite` to add to an icon suite handles to icon data. When you use `AddIconToSuite`, the handles that you add to the suite do not have to be associated with a resource fork. For example, your application might get icon data from the desktop database rather than reading it from a resource, or your application might read icon data from a resource and then detach it. In either case, you can provide a handle to the icon data and use `AddIconToSuite` to add the handle to the icon suite.

An **icon cache** is like an icon suite, except that an icon cache also contains a pointer to an application-defined **icon getter function** and a pointer to data that is associated with the icon suite. You can pass a handle to an icon cache to any of the Icon Utilities routines that accept a handle to an icon suite. An icon cache typically does not contain handles to the icon resources for all icon family members. Instead, if the icon cache does not contain an entry for a specific type of icon in an icon family, the Icon Utilities routines call your application's icon getter function to retrieve the data for that icon type. The icon getter function should return either a handle to the icon data or `NIL` to indicate that no icon data exists for the specified icon type.

Drawing an Icon Directly From a Resource

To draw an icon from an icon family without first creating an icon suite, use the `PlotIconID` function. Listing 5-1 shows an application-defined procedure that draws an icon from an icon family. Given a resource ID, the `PlotIconID` function determines which member of the icon family to draw and then draws the icon in the given rectangle with the specified transform and alignment.

Listing 5-1 Drawing the icon from an icon family that is best suited to the user's display

```
PROCEDURE MyDrawIconFromFamily (resID: Integer; destRect: Rect);
VAR
   align:         IconAlignmentType;
   transform:     IconTransformType;
   myErr:         OSErr;
BEGIN
   align := atAbsoluteCenter; {specify alignment (centered)}
   transform := ttNone;       {specify no special transforms}
   {draw the icon, using the icon type best suited for the }
   { destination rect and current bit depth of the display device}
   myErr := PlotIconID(destRect, align, transform, resID);
END;
```

The `PlotIconID` function determines, from the size of the specified destination rectangle and the current bit depth of the display device, which icon of a given size from an icon family to draw. For example, if the coordinates of the destination rectangle are (100,100,116,116) and the display device is set to 4-bit color, the `PlotIconID` function draws the icon of type `'ics4'` if that icon is available in the icon family.

If the width or height of a destination rectangle is greater than or equal to 32, `PlotIconID` uses the 32-by-32 pixel icon with the appropriate bit depth for the display device. If the destination rectangle is less than 32 by 32 pixels and greater than 16 pixels wide or 12 pixels high, `PlotIconID` uses the 16-by-16 pixel icon with the appropriate bit depth. If the destination rectangle's height is less than or equal to 12 pixels or its width is less than or equal to 16 pixels, `PlotIconID` uses the 12-by-16 pixel icon with the appropriate bit depth. (Typically only the Finder and the Standard File Package use 12-by-16 pixel icons.)

Depending on the size of the rectangle, the `PlotIconID` function may stretch or shrink the icon to fit. To draw icons without stretching them, `PlotIconID` requires that the destination rectangle have the exact dimensions of a standard icon: that is, depending on the icon resource type, 32 by 32 pixels, 16 by 16 pixels, or 12 by 16 pixels. If you use destination rectangles of other sizes, `PlotIconID` stretches or shrinks the icons to fit the rectangles.

Getting an Icon Suite and Drawing One of Its Icons

Listing 5-2 shows how you can use the `GetIconSuite` and `PlotIconSuite` functions to get an icon suite and then draw the icon from the suite that is best suited to the destination rectangle and the current bit depth of the display device.

Listing 5-2 Drawing the icon from an icon suite that is best suited to the display device

```
PROCEDURE MyDrawIconInSuite (resID: Integer; destRect: Rect;
                             VAR iconSuiteHdl: Handle);
VAR
   iconType:      IconSelectorValue;
   align:         IconAlignmentType;
   transform:     IconTransformType;
   myErr:         OSErr;
BEGIN
   iconType := svAllAvailable;   {get all icons in icon family}
   myErr := GetIconSuite(iconSuiteHdl, resID, iconType);
   IF iconSuiteHdl <> NIL THEN
   BEGIN
      align := atAbsoluteCenter; {specify alignment (centered)}
      transform := ttNone;       {specify no special transforms}
      {draw the icon, using the icon type best suited for the }
      { destination rect & current bit depth of display device}
      myErr := PlotIconSuite(destRect, align, transform,
                             iconSuiteHdl);
   END;
END;
```

The application-defined procedure `MyDrawIconInSuite` shown in Listing 5-2 first uses the `GetIconSuite` function, specifying the constant `svAllAvailable` in the third parameter, to get all icons from the icon family with the specified resource ID and to collect the handles to the data for each icon into an icon suite. (You can use other constants in the third parameter of `GetIconSuite` to request only certain members of an icon family for an icon suite.) The `MyDrawIconInSuite` procedure then draws an icon from this suite using the `PlotIconSuite` function.

Like the `PlotIconID` function described in the previous section, the `PlotIconSuite` function determines, from the size of the specified destination rectangle and the current bit depth of the display device, which icon from the icon suite to draw.

You can also specify various transforms and alignments to `PlotIconSuite`. For example, the code in Listing 5-2 specifies that `PlotIconSuite` should center the icon within the destination rectangle.

Drawing Specific Icons From an Icon Family

In most cases you should use `PlotIconID` or `PlotIconSuite` to draw an icon from an icon family, because these routines automatically select the best version of an icon to display for a given destination rectangle and bit depth. The preceding sections, "Drawing an Icon Directly From a Resource" and "Getting an Icon Suite and Drawing One of Its Icons," describe how to use these routines.

If you need to plot a specific icon from an icon family rather than using the Icon Utilities to select a family member, you must first create an icon suite that contains only the icon from the desired resource type and its corresponding mask. You can then use `PlotIconSuite` to plot the icon. In this case `PlotIconSuite` still attempts to use the best icon available for the given destination rectangle and bit depth; however, by limiting the icon resources available in the icon suite, you can force `PlotIconSuite` to plot either the black-and-white icon from the `'ICN#'` resource or just one of the other available resources. Listing 5-3 demonstrates how to do this.

Listing 5-3 Drawing a specific icon from an icon family or icon suite

```
PROCEDURE MyDrawThisIcon (destRect: Rect; resID: Integer;
                          VAR iconSuiteHdl: Handle);
VAR
   align:         IconAlignmentType;
   transform:     IconTransformType;
   myErr:         OSErr;
BEGIN
   {get only the 'ICN#' and 'icl4' icons and collect them in an }
   { icon suite}
   myErr := GetIconSuite(iconSuiteHdl, resID,
                         svLarge1Bit + svLarge4Bit);
   IF iconSuiteHdl <> NIL THEN
   BEGIN
      align := atAbsoluteCenter; {specify alignment (centered)}
      transform := ttNone;       {specify no special transforms}
      {draw the best icon from the suite referenced by the icon }
      { suite handle; since the suite contains only 'ICN#' and }
      { 'icl4' icons, PlotIconSuite draws the best of the two}
      myErr := PlotIconSuite(destRect, align, transform,
                             iconSuiteHdl);
   END;
END;
```

The application-defined procedure `MyDrawThisIcon` passes the constants `svLarge1Bit` and `svLarge4Bit` to `GetIconSuite`. In response, `GetIconSuite` reads only the `'ICN#'` and `'icl4'` resources into memory, storing handles to the icon

resource data in the icon suite. `MyDrawThisIcon` then uses `PlotIconSuite` to plot the best available icon from the suite.

If the bit depth of the display device is 1, the `PlotIconSuite` function in Listing 5-3 displays the black-and-white version of the icon from the `'ICN#'` resource, regardless of the size of the destination rectangle. If the bit depth of the display device is greater than 1, `PlotIconSuite` draws the icon from the `'icl4'` resource, regardless of the size of the destination rectangle.

Manipulating Icons

You can use the `GetIconFromSuite` function to get a handle to the pixel data for a specific icon from an icon suite. You can use the handle returned by the function `GetIconFromSuite` to manipulate the icon data—for example, to alter its color or add three-dimensional shading—but not to draw the icon with other Icon Utilities routines such as `PlotIconHandle`.

Listing 5-4 provides an example of an application-defined procedure, `MyGetIconData`, that calls `GetIconFromSuite` and manipulates the icon data.

Listing 5-4 Manipulating icon data in memory

```
PROCEDURE MyGetIconData (iconType: ResType; iconSuite: Handle;
                         VAR iconHandle: Handle);
VAR
   myErr: OSErr;
BEGIN
   {get the data for the icon with iconType from the suite}
   myErr := GetIconFromSuite(iconHandle, iconSuite, iconType);
   {do whatever with the data}
   myErr := MyManipulateIconData(iconHandle, iconType);
END;
```

The Icon Utilities also include routines that allow you to perform an action on one or more icons in an icon suite and to perform hit-testing on icons. For information about these routines, see “Performing Operations on Icons in an Icon Suite” and “Determining Whether a Point or Rectangle Is Within an Icon” beginning on page 5-38 and page 5-46, respectively.

Drawing Icons That Are Not Part of an Icon Family

To draw icons of resource type `'ICON'` or `'cicn'` in menus and dialog boxes, you can use the Menu Manager and Dialog Manager as described in *Inside Macintosh: Macintosh Toolbox Essentials*. You can also use Menu Manager routines to draw resources of type `'SICN'`.

To draw resources of resource type 'ICON', 'cicn', or 'SICN' in your application's windows, you can use these routines:

Resource type	Routines
'ICON'	PlotIconHandle PlotIcon
'cicn'	PlotCIconHandle PlotCIcon
'SICN'	PlotSICNHandle

The routines in this list that end in Handle allow you to specify alignment and transforms for the icons. You are responsible for disposing of the handle you pass to any of these routines.

Note

Unlike PlotCIcon, PlotCIconHandle doesn't honor the current foreground and background colors. ◆

The listings that follow provide examples of how to draw each of the three icon resource types that are not part of an icon family.

Listing 5-5 shows how to use PlotIcon to draw an icon of resource type 'ICON' without specifying alignment or transforms. The application-defined procedure MyPlotAnICON uses GetIcon to get a handle to the data for the desired icon and then passes the destination rectangle and the handle to PlotIcon.

Listing 5-5 Drawing an icon of resource type 'ICON'

```
PROCEDURE MyPlotAnICON (resID: Integer; destRect: Rect;
                        VAR myIcon: Handle);
BEGIN
   myIcon := GetIcon(resID);
   PlotIcon(destRect, myIcon);
END;
```

IMPORTANT

When you are finished using a handle obtained from GetIcon, use the ReleaseResource procedure to release the memory occupied by the icon resource data; for more information about ReleaseResource, see the chapter "Resource Manager" in this book. ▲

Listing 5-6 shows how to use `PlotIconHandle` to draw an icon of resource type `'ICON'` with a specific alignment and transform. The application-defined procedure `MyPlotAnICONWithAlignAndTransform` uses `GetIcon` to get a handle to the data for the desired icon and then passes the destination rectangle, alignment, transform, and handle to `PlotIconHandle`.

Listing 5-6 Drawing an icon of resource type `'ICON'` with a specific alignment and transform

```
PROCEDURE MyPlotAnICONWithAlignAndTransform
              (resID: Integer; destRect: Rect;
               align: IconAlignmentType;
               transform: IconTransformType; VAR myIcon: Handle);
VAR
   myErr: OSErr;
BEGIN
   myIcon := GetIcon(resID);
   myErr := PlotIconHandle(destRect, align, transform, myIcon);
END;
```

For the `PlotIconHandle` function in Listing 5-6 to draw the icon without stretching it, the destination rectangle passed in the `destRect` parameter of `MyPlotAnICONWithAlignAndTransform` must be exactly 32 by 32 pixels. If the destination rectangle is not 32 by 32 pixels, `PlotIconHandle` expands or shrinks the icon to fit.

Listing 5-7 shows how to use `PlotCIcon` to draw an icon of resource type `'cicn'` without specifying alignment or transform. The `MyPlotAcicn` procedure uses `GetCIcon` to get a handle to the color icon record of the desired icon and then passes the destination rectangle and handle to `PlotCIcon`.

Listing 5-7 Drawing an icon of resource type `'cicn'`

```
PROCEDURE MyPlotAcicn (resID: Integer; destRect: Rect;
                       VAR myCicnIcon: CIconHandle);
BEGIN
   myCicnIcon := GetCIcon(resID);
   PlotCIcon(destRect, myCicnIcon);
END;
```

Listing 5-8 shows how to use `PlotCIconHandle` to draw an icon of resource type `'cicn'` with a specific alignment and transform. Listing 5-8 uses `GetCIcon` to get a handle to the color icon record of the desired icon and then passes the destination rectangle, alignment, transform, and handle to `PlotCIconHandle`.

Listing 5-8 Drawing an icon of resource type `'cicn'` with a specific alignment and transform

```
PROCEDURE MyPlotAcicnWithAlignAndTransform
              (resID: Integer; destRect: Rect;
               align: IconAlignmentType;
               transform: IconTransformType;
               VAR myCicnIcon: CIconHandle);
VAR
   myErr: OSErr;
BEGIN
   myCicnIcon := GetCIcon(resID);
   myErr := PlotCIconHandle(destRect, align, transform,
                            myCicnIcon);
END;
```

Listing 5-9 shows how to use `PlotSICNHandle` to draw an icon of resource type `'SICN'` with a specific alignment and transform. The application-defined procedure `MyPlotAnSICNWithAlignAndTransform` uses `GetResource` to get a handle to the data for the desired icon and then passes the destination rectangle, alignment, transform, and handle to `PlotSICNHandle`.

Listing 5-9 Drawing an icon of resource type `'SICN'` with a specific alignment and transform

```
PROCEDURE MyPlotAnSICNWithAlignAndTransform
              (resID: Integer; destRect: Rect;
               align: IconAlignmentType;
               transform: IconTransformType; VAR myIcon: Handle);
VAR
   myErr: OSErr;
BEGIN
   myIcon := GetResource('SICN', resID);
   myErr := PlotSICNHandle(destRect, align, transform, myIcon);
END;
```

For the `PlotSICNHandle` function in Listing 5-9 to draw the icon without stretching it, the destination rectangle passed in the `destRect` parameter of `MyPlotAnSICNWithAlignAndTransform` must be exactly 16 by 16 pixels. If the destination rectangle is not this size, `PlotSICNHandle` expands or shrinks the icon to fit.

Icon Utilities Reference

The sections that follow describe the data structure and routines provided by the Icon Utilities.

The first section, "Data Structure," describes the color icon record. "Icon Utilities Routines" beginning on page 5-18 describes the routines for drawing and manipulating icons. "Application-Defined Routines" beginning on page 5-57 describes the syntax of the icon action and icon getter functions that your application can provide for use by Icon Utilities routines.

Data Structure

This section describes the color icon record. Note that you use the color icon record only for icons of resource type `'cicn'`; you do not need to use the color icon record for any of the color icons in an icon family.

The Color Icon Record

The `GetCIcon` function reads in a **color icon resource**—that is, an icon resource of type `'cicn'`—and returns a handle to a color icon record. A **color icon record** is defined by the `CIcon` data type.

```
TYPE
   CIcon =
   RECORD
      iconPMap:      PixMap;     {the icon's pixel map}
      iconMask:      BitMap;     {the icon's mask}
      iconBMap:      BitMap;     {the icon's bitmap}
      iconData:      Handle;     {handle to the icon's data}
      iconMaskData:              {the data for the icon's mask}
                     ARRAY[0..0] OF Integer;
   END;
   CIconPtr    = ^CIcon;         {pointer to color icon record}
   CIconHandle = ^CIconPtr;      {handle to color icon record}
```

Field descriptions

`iconPMap`	The pixel map describing the icon. Note that this is a pixel map record, not a handle to a pixel map record.
`iconMask`	A bitmap of the icon's mask.
`iconBMap`	A bitmap of the icon.
`iconData`	A handle to the icon's pixel image.
`iconMaskData`	An array containing the icon's mask data followed by the icon's bitmap data. This is used only when the icon is stored as a resource.

Your application can load a color icon resource into memory using the `GetCIcon` function. All color icon resources should be marked purgeable. To draw a color icon, you can use the `PlotCIcon` or `PlotCIconHandle` function. When your application has finished using a color icon, it can dispose of the color icon record by calling the `DisposeCIcon` function.

You can use icons of resource type `'cicn'` in menus the same way that you use resources of type `'ICON'`. If a menu item specifies an icon number, the menu definition procedure first tries to load in a `'cicn'` resource with the specified resource ID. If it doesn't find one, the menu definition procedure tries to load in an `'ICON'` resource with the same ID. The Dialog Manager also uses a `'cicn'` resource instead of an `'ICON'` resource if it finds one with the same resource ID. For more information, see *Inside Macintosh: Macintosh Toolbox Essentials*.

For information about the format of a color icon resource, see *Inside Macintosh: Imaging With QuickDraw*.

Icon Utilities Routines

This section describes the Icon Utilities routines. You can use these routines to draw icons in windows and, if necessary, in menus and dialog boxes. You can also use Icon Utilities routines to perform operations on icons in an icon suite, get and set labels associated with specific icon suites, dispose of icon suites and color icon records, convert an icon mask to a region, perform hit-testing, and create and manipulate icon caches. Note that you can pass a handle to an icon cache to any of the Icon Utilities routines that accept a handle to an icon suite.

Most of the Icon Utilities routines are available only in System 7 and later. To determine whether they are available, call the `Gestalt` function with the `gestaltIconUtilitiesAttr` selector and check the value of the `response` parameter. If the bit indicated by the constant `gestaltIconUtilitiesPresent` is set, then the Icon Utilities are available. The `GetIcon`, `PlotIcon`, `GetCIcon`, `PlotCIcon`, and `DisposeCIcon` routines are available in both System 6 and System 7.

Note
The Icon Utilities routines do not place any restrictions on whether icon resources are purgeable or nonpurgeable; however, in general, you should specify your icon resources as purgeable. ◆

This section first describes the routines you can use to draw icons from an icon family and then describes the routines that work with icon suites and icon caches.

IMPORTANT
All of the Icon Utilities routines may move or purge memory blocks in the application heap or for some other reason should not be called from within an interrupt. Your application should not call Icon Utilities routines at interrupt time. ▲

Assembly-Language Note
You can invoke Icon Utilities routines by using the trap `_IconDispatch` with the appropriate routine selector. The routine selectors are listed in "Assembly-Language Summary" beginning on page 5-71. ◆

Drawing Icons From Resources

The routines described in this section allow you to plot an icon directly from a resource without first creating an icon suite.

To draw an icon from an icon family (that is, those resources of type `'ICN#'`, `'ics#'`, `'icl4'`, `'icl8'`, `'ics4'`, or `'ics8'` that share the same resource ID), use the `PlotIconID` function. This function gets the icon's data from its resource and also allows you to specify transforms and alignment. The `PlotIconID` function also determines, from the destination rectangle in which the icon is to be drawn and the current bit depth of the display device, which resource type to get from the icon family.

To draw an icon obtained with the aid of an icon getter function, use the `PlotIconMethod` function. For information about icon getter functions, see "Icon Getter Functions" beginning on page 5-58.

To plot an icon of resource types `'ICON'` and `'cicn'` from an icon handle previously obtained from the `GetIcon` or `GetCIcon` function, use the `PlotIconHandle` and `PlotCIconHandle` functions, respectively. These functions allow you to specify transforms and alignment.

You can also plot an icon of resource types `'ICON'` and `'cicn'` using the `PlotIcon` and `PlotCIcon` procedures, respectively. However, neither of these procedures allow you to specify transforms and alignment.

To plot an icon of resource type `'SICN'`, use the `PlotSICNHandle` function. This function allows you to specify transforms and alignment.

PlotIconID

You can use the `PlotIconID` function to draw the icon described by an icon family. From the icon family, `PlotIconID` selects the most appropriate icon resource for the current bit depth of the display device and the rectangle in which the icon is to be drawn.

```
FUNCTION PlotIconID (theRect: Rect; align: IconAlignmentType;
                     transform: IconTransformType;
                     theResID: Integer): OSErr;
```

`theRect` The rectangle, specified in local coordinates of the current graphics port, in which to draw the icon. The `PlotIconID` function determines, from the size of the specified destination rectangle and the current bit depth of the display device, which icon of a given size to draw from an icon family.

`align` A value that specifies how `PlotIconID` should align the icon within the rectangle. For example, you can specify that `PlotIconID` center the icon within the rectangle or align it at one side or the other. See the description that follows for a list of constants you can use in this parameter.

`transform` A value that specifies how `PlotIconID` should modify the appearance of the icon. See the description that follows for a list of constants you can use in this parameter.

`theResID` The resource ID of the icon to draw. The icon resource must be of resource type `'ICN#'`, `'ics#'`, `'icl4'`, `'icl8'`, `'ics4'`, or `'ics8'`.

DESCRIPTION

The `PlotIconID` function plots a single icon from the icon family specified by `theResID`. You cannot determine which icon from the family it will draw; `PlotIconID` bases this decision on the size of the specified destination rectangle and the current bit depth of the display device. For example, if the destination rectangle has the coordinates (100,100,116,116) and the display device is set to 4-bit color, the `PlotIconID` function draws the icon of type `'ics4'` if that icon is available in the icon family.

If the width or height of a destination rectangle is greater than or equal to 32, `PlotIconID` uses the 32-by-32 pixel icon with the appropriate bit depth for the display device. If the destination rectangle is less than 32 by 32 pixels and greater than 16 pixels wide or 12 pixels high, `PlotIconID` uses the 16-by-16 pixel icon with the appropriate bit depth. If the destination rectangle's height is less than or equal to 12 pixels or its width is less than or equal to 16 pixels, `PlotIconID` uses the 12-by-16 pixel icon with the appropriate bit depth. (Typically only the Finder and Standard File Package use 12-by-16 pixel icons.)

You can use these constants in the `align` parameter to specify the alignment of the icon within the rectangle specified by the `theRect` parameter:

```
CONST
   atNone               = $0; {no special alignment}
   atVerticalCenter     = $1; {centered vertically}
   atTop                = $2; {top aligned}
   atBottom             = $3; {bottom aligned}
   atHorizontalCenter   = $4; {centered horizontally}
   atLeft               = $8; {left aligned}
   atRight              = $C; {right aligned}
   atAbsoluteCenter     = (atVerticalCenter + atHorizontalCenter);
   atCenterTop          = (atTop + atHorizontalCenter);
   atCenterBottom       = (atBottom + atHorizontalCenter);
   atCenterLeft         = (atVerticalCenter + atLeft);
   atTopLeft            = (atTop + atLeft);
   atBottomLeft         = (atBottom + atLeft);
   atCenterRight        = (atVerticalCenter + atRight);
   atTopRight           = (atTop + atRight);
   atBottomRight        = (atBottom + atRight);
```

The destination rectangle passed in the `theRect` parameter of `PlotIconID` must be exactly 32 by 32 pixels, 16 by 16 pixels, or 12 by 16 pixels for `PlotIconID` to draw the icon without stretching it. If the destination rectangle is not one of these standard sizes, `PlotIconID` expands or shrinks the icon to fit. After stretching or shrinking the icon, the `PlotIconID` function aligns the icon according to the value specified in the `align` parameter, moving the icon so that the edges of its mask align with the specified side or direction.

You can pass constants in the `transform` parameter to specify how you want the icon modified, if at all, when plotted by `PlotIconID`. If you don't want to specify any transform constants, specify `ttNone` in the `transform` parameter.

```
CONST ttNone         =    $0;
```

You can use these constants in the `transform` parameter to transform the icon in a manner analogous to certain Finder states for icons:

```
CONST
   ttDisabled           = $1;
   ttOffline            = $2;
   ttOpen               = $3;
   ttSelected           = $4000;
   ttSelectedDisabled   = (ttSelected + ttDisabled);
   ttSelectedOffline    = (ttSelected + ttOffline);
   ttSelectedOpen       = (ttSelected + ttOpen);
```

You can use another group of constants to color the icon using the Finder label colors. To determine the appropriate label for a file's icon, you can check bits 1–3 of the `fdFlags` field in the file's file information record. These bits contain a number from 0 to 7 indicating the label setting (0 indicates no label). Simply add the corresponding constant from this list to the `transform` parameter when you call `PlotIconID`:

```
CONST
   ttLabel1             = $0100;
   ttLabel2             = $0200;
   ttLabel3             = $0300;
   ttLabel4             = $0400;
   ttLabel5             = $0500;
   ttLabel6             = $0600;
   ttLabel7             = $0700;
```

RESULT CODES

`noErr`	0	No error
`resNotFound`	–192	Resource not found
`noMaskFoundErr`	–1000	No mask found

SEE ALSO

For an example of the use of the `PlotIconID` function, see Listing 5-1 on page 5-10.

To restrict the icons from an icon family that are available for use by the Icon Utilities, see "Drawing Specific Icons From an Icon Family" on page 5-12.

For information about the file information record, see the chapter "Finder Interface" in *Inside Macintosh: Macintosh Toolbox Essentials*.

PlotIconMethod

You can use the `PlotIconMethod` function to plot an icon obtained with the aid of an icon getter function for a specified destination rectangle and alignment.

```
FUNCTION PlotIconMethod (theRect: Rect; align: IconAlignmentType;
                         transform: IconTransformType;
                         theMethod: IconGetter;
                         yourDataPtr: UNIV Ptr): OSErr;
```

`theRect` The rectangle in which to draw the icon, specified in local coordinates of the current graphics port.

`align` A value that specifies how to align the icon within the rectangle specified by `theRect`. See the description of `PlotIconID` on page 5-20 for a list of constants you can use in this parameter.

`transform` A value that specifies how `PlotIconMethod` should modify the appearance of the icon. See the description of `PlotIconID` beginning on page 5-20 for a list of constants you can use in this parameter.

`theMethod` A pointer to an icon getter function.

`yourDataPtr` A pointer to data that is passed to your icon getter function.

DESCRIPTION

The `PlotIconMethod` function uses your icon getter function to obtain the icon to draw. Then `PlotIconMethod` draws this icon in the specified destination rectangle, with the specified transform and alignment.

`PlotIconMethod` passes to your icon getter function the type of the icon to draw and the value specified in the `yourDataPtr` parameter. The `PlotIconMethod` function examines the current bit depth of the display devices and calls your icon getter function once for each display device that intersects the rectangle specified in the parameter `theRect`. Your icon getter function should return a handle to the requested icon's data. Your icon getter function can get the icon data using whatever method is appropriate to your application. For example, your application might maintain its own cache of icons or use its icon getter function to get an icon from the desktop database.

RESULT CODES

`noErr`	0	No error
`noMaskFoundErr`	–1000	No mask found

SEE ALSO

For more information about icon getter functions, see page 5-58.

PlotIcon

You can use the `PlotIcon` procedure to plot an icon of resource type `'ICON'`. You must have previously obtained a handle to the icon using `GetIcon` (or `GetResource` or other Resource Manager routines).

```
PROCEDURE PlotIcon (theRect: Rect; theIcon: Handle);
```

`theRect` The rectangle in which to draw the icon, specified in local coordinates of the current graphics port.

`theIcon` A handle to the icon to draw.

DESCRIPTION

The `PlotIcon` procedure draws the icon specified by the given handle. Unlike `PlotIconHandle`, `PlotIcon` does not allow you to specify any transforms or alignment. The `PlotIcon` procedure uses the QuickDraw procedure `CopyBits` with the `srcCopy` transfer mode.

If the destination rectangle is not 32 by 32 pixels, `PlotIcon` stretches or shrinks the icon to fit.

To plot an icon of resource type `'ICON'` with a specified transform and alignment, use `PlotIconHandle` (described next).

SEE ALSO

For an example of the use of the `PlotIcon` procedure, see Listing 5-5 on page 5-14. For information on `GetIcon`, see page 5-28. For information on the QuickDraw procedure `CopyBits`, see *Inside Macintosh: Imaging With QuickDraw*.

PlotIconHandle

You can use the `PlotIconHandle` function to plot an icon of resource type `'ICON'` or `'ICN#'`. You must have previously obtained a handle to the icon using `GetIcon` (or `GetResource` or other Resource Manager routines).

```
FUNCTION PlotIconHandle (theRect: Rect; align: IconAlignmentType;
                         transform: IconTransformType;
                         theIcon: Handle): OSErr;
```

`theRect` The rectangle in which to draw the icon, specified in local coordinates of the current graphics port.

`align` A value that specifies how `PlotIconHandle` should align the icon within the rectangle. See the description of `PlotIconID` on page 5-20 for a list of constants you can use in this parameter.

`transform` A value that specifies how `PlotIconHandle` should modify the appearance of the icon. See the description of `PlotIconID` beginning on page 5-20 for a list of constants you can use in this parameter.

`theIcon` A handle to the icon to draw.

DESCRIPTION

The `PlotIconHandle` function draws the icon specified by the `theIcon` parameter with the transform and alignment specified by the `transform` and `align` parameters.

IMPORTANT

To plot an icon from an icon suite, you should normally use `PlotIconSuite`. The `PlotIconHandle` function may not draw the icon correctly if you pass it the handle returned in the `theIconData` parameter of `GetIconFromSuite`. ▲

RESULT CODES

`noErr`	0	No error
`noMaskFoundErr`	–1000	No mask found

SEE ALSO

For an example of the use of the `PlotIconHandle` function, see Listing 5-6 on page 5-15. For information on `GetIcon`, see page 5-28.

PlotCIcon

You can plot a color icon of resource type `'cicn'` using the `PlotCIcon` procedure. You must have previously obtained a handle to the icon using `GetCIcon` (or `GetResource` or other Resource Manager routines).

```
PROCEDURE PlotCIcon (theRect: Rect; theIcon: CIconHandle);
```

`theRect`	The rectangle in which to draw the icon, specified in local coordinates of the current graphics port.
`theIcon`	A handle to the color icon record of the color icon to draw.

DESCRIPTION

The `PlotCIcon` procedure draws the color icon specified by the given handle. The `iconMask` field of the color icon record determines which pixels in the `iconPMap` field are drawn and which are not. Only pixels with 1s in corresponding positions in the `iconMask` field are drawn. If the screen depth is 1 or 2 bits per pixel, `PlotCIcon` uses the `iconBMap` field instead of the `iconPMap` field (unless the `rowBytes` field of `IconBMap` contains 0, indicating that there is no bitmap for the icon).

When PlotCIcon draws the icon, it uses the bounds field of iconPMap as the source rectangle of the image. If the destination rectangle is not the same size as the icon or its mask, PlotCIcon stretches or shrinks the icon to fit. The icon's pixels are remapped to the current depth and color table, if necessary. The bounds fields of iconPMap, iconBMap, and iconMask are expected to be equal in size.

Unlike PlotIconHandle, PlotCIcon does not allow you to specify any transforms or alignment. The PlotCIcon procedure uses the QuickDraw procedure CopyMask and doesn't send any of its drawing commands through QuickDraw bottleneck routines. Therefore, calls to PlotCIcon are not recorded as pictures.

RESULT CODE

noErr	0	No error

SEE ALSO

For a description of the color icon record, see "The Color Icon Record" on page 5-17. For information on GetCIcon, see page 5-29. For information on the QuickDraw procedure CopyMask, see *Inside Macintosh: Imaging With QuickDraw*.

For an example of the use of the PlotCIcon procedure, see Listing 5-7 on page 5-15.

PlotCIconHandle

You can use the PlotCIconHandle function to plot an icon of resource type 'cicn'. You must have previously obtained a handle to the icon using GetCIcon (or GetResource or other Resource Manager routines).

```
FUNCTION PlotCIconHandle (theRect: Rect; align: IconAlignmentType;
                          transform: IconTransformType;
                          theCIcon: CIconHandle): OSErr;
```

theRect	The rectangle in which to draw the icon, specified in local coordinates of the current graphics port.
align	A value that specifies how PlotCIconHandle should align the icon within the rectangle. See the description of PlotIconID on page 5-20 for a list of constants you can use in this parameter.
transform	A value that specifies how PlotCIconHandle should modify the appearance of the icon. See the description of PlotIconID beginning on page 5-20 for a list of constants you can use in this parameter.
theCIcon	A handle to the color icon record of the icon to draw.

DESCRIPTION

The `PlotCIconHandle` function draws the specified color icon with the transform and alignment specified by the `transform` and `align` parameters. Unlike `PlotCIcon`, `PlotCIconHandle` doesn't honor the current foreground and background colors.

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Error in parameter list

SEE ALSO

For an example of the use of `PlotCIconHandle`, see Listing 5-8 on page 5-16. For information on `GetCIcon`, see page 5-29. For a description of the color icon record, see page 5-17.

PlotSICNHandle

You can use the `PlotSICNHandle` function to plot a small icon of resource type `'SICN'` with a specified transform and alignment. You must have previously obtained a handle to the icon using `GetResource` (or other Resource Manager routines).

```
FUNCTION PlotSICNHandle (theRect: Rect; align: IconAlignmentType;
                         transform: IconTransformType;
                         theSICN: Handle): OSErr;
```

`theRect` The rectangle in which to draw the icon, specified in local coordinates of the current graphics port.

`align` A value that specifies how `PlotSICNHandle` should align the icon within the rectangle. See the description of `PlotIconID` on page 5-20 for a list of constants you can use in this parameter.

`transform` A value that specifies how `PlotSICNHandle` should modify the appearance of the icon. See the description of `PlotIconID` beginning on page 5-20 for a list of constants you can use in this parameter.

`theSICN` A handle to the icon to draw.

DESCRIPTION

The `PlotSICNHandle` function draws the specified small icon with the transform and alignment specified by the `transform` and `align` parameters. Only `'SICN'` resources with a single member—or with two members, the second of which is a mask for the first—plot correctly.

RESULT CODES

noErr	0	No error
noMaskFoundErr	–1000	No mask found

SEE ALSO

For an example of the use of the `PlotSICNHandle` function, see Listing 5-9 on page 5-16.

Getting Icons From Resources That Don't Belong to an Icon Family

You can get a handle to an `'ICON'` or `'cicn'` resource using the `GetIcon` and `GetCIcon` functions. You can then draw these icons using the routines `PlotIcon`, `PlotCIcon`, `PlotIconHandle`, or `PlotCIconHandle` (see "Drawing Icons From Resources" beginning on page 5-19).

To get a handle to an icon suite for a given icon family, use the routines described in "Creating an Icon Suite" beginning on page 5-30.

GetIcon

You can use the `GetIcon` function to get a handle to an icon resource of type `'ICON'`.

```
FUNCTION GetIcon (iconID: Integer): Handle;
```

`iconID` The resource ID for an icon of resource type `'ICON'`.

DESCRIPTION

The `GetIcon` function reads in the `'ICON'` resource with the specified resource ID and returns a handle to it. The `GetIcon` function searches the current resource chain for the resource. If `GetIcon` finds the resource, it reads the resource and returns a handle to the icon as its function result. If `GetIcon` can't find the resource, it returns `NIL` as its function result.

To draw an icon obtained from `GetIcon` in a specified rectangle, you can use either `PlotIcon` or `PlotIconHandle`. Unlike `PlotIcon`, `PlotIconHandle` allows you to specify transforms and alignments.

When you are finished using a handle obtained from `GetIcon`, use the `ReleaseResource` procedure to release the memory occupied by the icon resource data.

RESULT CODES

noErr	0	No error
resNotFound	–192	Resource not found

SEE ALSO

For a description of the `PlotIcon` procedure and `PlotIconHandle` function, see page 5-23 and page 5-24, respectively. For information about `ReleaseResource`, see the chapter "Resource Manager" in this book.

GetCIcon

You can use `GetCIcon` to get a handle to a color icon of resource type `'cicn'`.

```
FUNCTION GetCIcon (iconID: Integer): CIconHandle;
```

`iconID` The resource ID for an icon of resource type `'cicn'`.

DESCRIPTION

The `GetCIcon` function reads in the `'cicn'` resource with the specified resource ID and returns a handle to it. The `GetCIcon` function searches the current resource chain for the resource. If `GetCIcon` finds the resource, it reads the resource, creates a color icon record for the icon, and initializes the fields of the record according to the information contained in the `'cicn'` resource. `GetCIcon` returns a handle to the color icon record as its function result. If `GetCIcon` can't find the resource, it returns `NIL` as its function result.

To draw an icon obtained from `GetCIcon` in a specified rectangle, you can use either the `PlotCIcon` or `PlotCIconHandle` routine. Unlike `PlotCIcon`, `PlotCIconHandle` allows you to specify transforms and alignments.

When you are finished with a handle obtained from `GetCIcon`, use the `DisposeCIcon` procedure to release the memory occupied by the color icon record.

RESULT CODES

noErr	0	No error
resNotFound	–192	Resource not found

SEE ALSO

For information about the color icon record, see "The Color Icon Record" on page 5-17. For information about the format of the `'cicn'` resource, see *Inside Macintosh: Imaging With QuickDraw*.
For descriptions of the `PlotCIcon` procedure and `PlotCIconHandle` function, see page 5-25 and page 5-26, respectively. The `DisposeCIcon` procedure is described next.

Disposing of Icons

When you are finished with a handle obtained from `GetCIcon`, use the `DisposeCIcon` procedure to release the memory occupied by the color icon record.

When you are finished using a handle obtained from `GetIcon` or `GetResource`, use the `ReleaseResource` procedure to release the memory occupied by the icon resource data; for more information about `GetResource` and `ReleaseResource`, see the chapter "Resource Manager" in this book.

To dispose of icons in an icon suite, use the `DisposeIconSuite` function described on page 5-42.

DisposeCIcon

You can use the `DisposeCIcon` procedure to release the memory occupied by an icon color record obtained from the `GetCIcon` function. The `DisposeCIcon` procedure is also available as the `DisposCIcon` procedure.

```
PROCEDURE DisposeCIcon (theIcon: CIconHandle);
```

`theIcon` A handle to the color icon record to dispose of.

DESCRIPTION

The `DisposeCIcon` procedure disposes of any structure allocated by `GetCIcon`.

Creating an Icon Suite

You typically create an icon suite by reading all resources from a specific icon family into memory and storing handles to the icon resource data in a new icon suite. You can do this using the `GetIconSuite` function. Alternatively, you can create an empty icon suite using the `NewIconSuite` function and then add icons to it one at a time using the `AddIconToSuite` function.

Although you typically create an icon suite using `GetIconSuite` (which fills the suite with handles to icon resource data), you can also create an icon suite and then add handles to icon data. The handles that you add to the suite do not have to be associated with a resource fork. For example, your application might get icon data from the desktop database rather than reading it from a resource, or your application might read icon data from a resource and then detach it. In either case, you can provide a handle to the icon data and use `AddIconToSuite` to add the handle to the icon suite. You need to release the memory occupied by the icon suite when you're finished using it. The `DisposeIconSuite` function releases this memory but does not release the memory of any resource handles. You can request `DisposeIconSuite` to release the memory of any other handles to icon data in the suite.

Note
When you create an icon suite from icon family resources, the associated resource file should remain open while you use Icon Utilities routines. ◆

GetIconSuite

You can use the `GetIconSuite` function to create an icon suite in memory that contains handles to a specified icon family's resources and to return a handle to the icon suite.

```
FUNCTION GetIconSuite (VAR theIconSuite: Handle;
                       theResID: Integer;
                       selector: IconSelectorValue): OSErr;
```

`theIconSuite`
`GetIconSuite` allocates the memory for and returns, in this parameter, a handle to an icon suite for the requested icon family. To release the memory occupied by an icon suite, you must use the `DisposeIconSuite` function.

`theResID` The resource ID of the icons in the icon family to be read into memory.

`selector` A value that indicates which icons from the icon family to include in the icon suite. See the description that follows for a list of constants you can use in this parameter.

DESCRIPTION

The `GetIconSuite` function returns a handle to a suite of icons for the icon family whose resource ID is specified in the `theResID` parameter. Use one or more of these constants in the `selector` parameter to specify which members of the family to include in the icon suite:

```
CONST
   svLarge1Bit       = $00000001; {'ICN#' resource}
   svLarge4Bit       = $00000002; {'icl4' resource}
   svLarge8Bit       = $00000004; {'icl8' resource}
   svSmall1Bit       = $00000100; {'ics#' resource}
   svSmall4Bit       = $00000200; {'ics4' resource}
   svSmall8Bit       = $00000400; {'ics8' resource}
   svMini1Bit        = $00010000; {'icm#' resource}
   svMini4Bit        = $00020000; {'icm4' resource}
   svMini8Bit        = $00040000; {'icm8' resource}
   svAllLargeData    = $000000FF; {'ICN#', 'icl4', and 'icl8' }
                                  { resources}
   svAllSmallData    = $0000FF00; {'ics#', 'ics4', and 'ics8' }
                                  { resources}
```

```
svAllMiniData     = $00FF0000; {'icm#', 'icm4', and 'icm8' }
                               { resources}
svAll1BitData     = (svLarge1Bit + svSmall1Bit + svMini1Bit);
svAll4BitData     = (svLarge4Bit + svSmall4Bit + svMini4Bit);
svAll8BitData     = (svLarge8Bit + svSmall8Bit + svMini8Bit);
svAllAvailableData= $FFFFFFFF; {all resources of given ID}
```

These constants are additive; that is, you can add several constants to include the corresponding family members in the icon suite.

When you create an icon suite using `GetIconSuite`, it sets the default label for the suite to none. To set a new default label for an icon suite, use the `SetSuiteLabel` function.

If you call `SetResLoad` with the `load` parameter set to `FALSE` before you call `GetIconSuite`, the suite is filled with unloaded resource handles.

To perform operations on one or more icons in an icon suite, use the `ForEachIconDo` function.

To draw the icon described by the icon suite using the icon family member that is most suitable for the current bit depth of the display device, use the `PlotIconSuite` function.

RESULT CODES

`noErr`	0	No error
`memFullErr`	–108	Not enough memory in heap zone

SEE ALSO

For examples of the use of the `GetIconSuite` function, see Listing 5-2 and Listing 5-3 on page 5-11 and page 5-12, respectively.

For a description of the `PlotIconSuite` and `ForEachIconDo` functions, see page 5-35 and page 5-38, respectively. For information on the `DisposeIconSuite` function, see page 5-42.

NewIconSuite

You can use the `NewIconSuite` function to get a handle to an empty icon suite. Then you can use `AddIconToSuite` to add handles to icon data.

```
FUNCTION NewIconSuite (VAR theIconSuite: Handle): OSErr;
```

`theIconSuite`
`NewIconSuite` allocates the memory for a new icon suite and returns, in this parameter, a handle to an empty icon suite.

DESCRIPTION

The `NewIconSuite` function returns a handle to an empty icon suite in the parameter `theIconSuite`. When you create an icon suite using `NewIconSuite`, it sets the default label for the suite to none. To set a new default label for an icon suite, use the `SetSuiteLabel` function. `NewIconSuite` allocates the memory for the icon suite handle. To release the memory occupied by an icon suite, you must use the `DisposeIconSuite` function.

RESULT CODES

`noErr`	0	No error
`memFullErr`	–108	Not enough memory in heap zone

SEE ALSO

For information on the `DisposeIconSuite` function, see page 5-42.

AddIconToSuite

You can use the `AddIconToSuite` function to add icons to an icon suite. This function is most often used to read icons into an empty icon suite created with `NewIconSuite`.

```
FUNCTION AddIconToSuite (theIconData: Handle; theSuite: Handle;
                         theType: ResType): OSErr;
```

`theIconData`
: A handle to the data for the new icon to be added to the icon suite. You can obtain a handle to icon data using various routines, such as `GetIcon` or `GetResource`.

`theSuite`
: A handle to the icon suite to which to add the icon.

`theType`
: The resource type of the new icon. The resource type should be that of an icon family member.

DESCRIPTION

The `AddIconToSuite` function adds the handle to the icon data to the specified icon suite at the location reserved for icon data of type `theType`. If the icon suite already includes a handle to icon data for that type, `AddIconToSuite` replaces the handle to the old data without disposing of it. In this case you may want to call `GetIconFromSuite` (described next) first to obtain the old handle so that you can dispose of it.

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	No such type in icon family

Getting Icons From an Icon Suite

The `GetIconFromSuite` function returns a handle to the specified icon in an icon suite.

GetIconFromSuite

You can use the `GetIconFromSuite` function to get an icon from an icon suite.

```
FUNCTION GetIconFromSuite (VAR theIconData: Handle;
                           theSuite: Handle;
                           theType: ResType): OSErr;
```

`theIconData`
`GetIconFromSuite` returns a handle to the data for the requested icon in this parameter. If an icon of the specified type does not exist in the given icon suite, `GetIconFromSuite` returns `NIL` in this parameter.

`theSuite` A handle to the icon suite from which to get the icon.

`theType` The resource type of the desired icon.

DESCRIPTION

The `GetIconFromSuite` function returns a handle to the data for the icon of type `theType` in the icon suite specified by `theSuite`. If you intend to dispose of the handle, pass a `NIL` handle to the `AddIconToSuite` function to delete the corresponding entry in the suite.

You can use the handle returned by `GetIconFromSuite` to manipulate the icon data, for example, to alter its color or add three-dimensional shading. However, you should not use the returned handle to draw the icon with other Icon Utilities routines.

IMPORTANT

To plot an icon from an icon suite, you should normally use `PlotIconSuite`. The `PlotIconHandle` function may not draw the icon correctly if you pass it the handle returned in the `theIconData` parameter of `GetIconFromSuite`. ▲

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	Requested type not present in suite

SEE ALSO

For an example of the use of the `GetIconFromSuite` function, see Listing 5-4 on page 5-13.

For a description of the `AddIconToSuite` function, see page 5-33. The `PlotIconSuite` function is described next.

Drawing Icons From an Icon Suite

To draw an icon from an icon suite using the icon that that is most appropriate for a specified rectangle and the current bit depth of the display device, use the `PlotIconSuite` function.

To draw an icon from a resource, use the routines described in "Drawing Icons From Resources" beginning on page 5-19. For example, to draw an icon from an icon family, use the `PlotIconID` function.

PlotIconSuite

You can use the `PlotIconSuite` function to draw the icon described by an icon suite using the most appropriate icon in the suite for the current bit depth of the display device and the rectangle in which the icon is to be drawn.

```
FUNCTION PlotIconSuite (theRect: Rect;
                        align: IconAlignmentType;
                        transform: IconTransformType;
                        theIconSuite: Handle): OSErr;
```

`theRect` The rectangle in which to draw the icon. The `PlotIconSuite` function uses the size of the specified destination rectangle and the current bit depth of the display device to determine which icon from an icon suite to draw.

`align` A value that specifies how `PlotIconSuite` should align the icon within the rectangle. For example, you can specify that `PlotIconSuite` center the icon within the rectangle or align it at one side or the other. See the description that follows for a list of constants you can use in this parameter.

`transform` A value that specifies how `PlotIconSuite` should modify the appearance of the icon. See the description that follows for a list of constants you can use in this parameter.

`theIconSuite`
A handle to the icon suite from which `PlotIconSuite` gets the icon to draw. You can get a handle to an icon suite using the `GetIconSuite` or `NewIconSuite` function.

DESCRIPTION

The `PlotIconSuite` function plots a single icon from an icon suite in the current graphics port. You cannot determine which icon from a given suite it will draw; `PlotIconSuite` bases this decision on the size of the specified destination rectangle and the current bit depth of the display device. For example, if the destination rectangle has the coordinates (100,100,116,116) and the display device is set to 4-bit color, the `PlotIconSuite` function draws the icon of type `'ics4'` if that icon is available in the icon suite.

If the width or height of a destination rectangle is greater than or equal to 32 pixels, `PlotIconSuite` uses the 32-by-32 pixel icon with the appropriate bit depth for the display device. If the destination rectangle is less than 32 by 32 pixels and greater than 16 pixels wide or 12 pixels high, `PlotIconSuite` uses the 16-by-16 pixel icon with the appropriate bit depth. If the destination rectangle's height is less than or equal to 12 pixels or its width is less than or equal to 16 pixels, `PlotIconSuite` uses the 12-by-16 pixel icon with the appropriate bit depth. (Typically, only the Finder and Standard File Package use 12-by-16 pixel icons.)

You can use these constants in the `align` parameter to specify the alignment of the icon within the rectangle specified by the parameter `theRect`:

```
CONST
   atNone               = $0; {no special alignment}
   atVerticalCenter     = $1; {centered vertically}
   atTop                = $2; {top aligned}
   atBottom             = $3; {bottom aligned}
   atHorizontalCenter   = $4; {centered horizontally}
   atLeft               = $8; {left aligned}
   atRight              = $C; {right aligned}
   atAbsoluteCenter     = (atVerticalCenter + atHorizontalCenter);
   atCenterTop          = (atTop + atHorizontalCenter);
   atCenterBottom       = (atBottom + atHorizontalCenter);
   atCenterLeft         = (atVerticalCenter + atLeft);
   atTopLeft            = (atTop + atLeft);
   atBottomLeft         = (atBottom + atLeft);
   atCenterRight        = (atVerticalCenter + atRight);
   atTopRight           = (atTop + atRight);
   atBottomRight        = (atBottom + atRight);
```

The destination rectangle passed in the `theRect` parameter of `PlotIconSuite` must be exactly 32 by 32 pixels, 16 by 16 pixels, or 12 by 16 pixels for `PlotIconSuite` to draw the icon without stretching it. If the destination rectangle is not one of these standard sizes, `PlotIconSuite` expands or shrinks the icon to fit. After stretching or shrinking the icon, the `PlotIconSuite` function aligns the icon according to the value specified in the `align` parameter, moving the icon so that the edges of its mask align with the specified side or direction.

You can pass constants in the `transform` parameter to specify how you want the icon modified, if at all, when plotted by `PlotIconSuite`. If you don't want to specify any transform constants, specify `ttNone` in the `transform` parameter.

```
CONST ttNone            = $0;
```

You can use these constants in the `transform` parameter to transform the icon in a manner analogous to certain Finder states for icons:

```
CONST
   ttDisabled           = $1;
   ttOffline            = $2;
   ttOpen               = $3;
   ttSelected           = $4000;
   ttSelectedDisabled   = (ttSelected + ttDisabled);
   ttSelectedOffline    = (ttSelected + ttOffline);
   ttSelectedOpen       = (ttSelected + ttOpen);
```

You can use another group of constants to color the icons using the Finder label colors. To determine the appropriate label for a file's icon, you can check bits 1–3 of the `fdFlags` field in the file's file information record. These bits contain a number from 0 to 7 indicating the label setting (0 indicates no label). Simply add the corresponding constant from this list to the `transform` parameter when you call `PlotIconSuite`:

```
CONST
   ttLabel1             = $0100;
   ttLabel2             = $0200;
   ttLabel3             = $0300;
   ttLabel4             = $0400;
   ttLabel5             = $0500;
   ttLabel6             = $0600;
   ttLabel7             = $0700;
```

If you don't specify a label constant in the `transform` parameter, `PlotIconSuite` displays the icon using the default label for that icon suite. When you create an icon suite using `GetIconSuite` or `NewIconSuite`, these functions set the default label for the suite to none. To set a new default label for an icon suite, use the `SetSuiteLabel` function.

RESULT CODES

`noErr`	0	No error
`noMaskFoundErr`	–1000	No mask found

SEE ALSO

For examples of the use of the `PlotIconSuite` function, see Listing 5-2 and Listing 5-3, starting on page 5-11.

For information on the `SetSuiteLabel` function, see page 5-40. See the chapter "Finder Interface" in *Inside Macintosh: Macintosh Toolbox Essentials* for more information about the file information record.

Performing Operations on Icons in an Icon Suite

You can perform an action on one or more icons in an icon suite using the `ForEachIconDo` function.

ForEachIconDo

You can use the `ForEachIconDo` function to perform an action on one or more icons in an icon suite.

```
FUNCTION ForEachIconDo (theSuite: Handle;
                        selector: IconSelectorValue;
                        action: IconAction;
                        yourDataPtr: Ptr): OSErr;
```

`theSuite` A handle to an icon suite.

`selector` A long integer whose bits determine which icons in the suite to perform the operation on. See the description that follows for a list of constants you can use in this parameter.

`action` A pointer to your icon action function.

`yourDataPtr` A pointer to data that is passed to your icon action function.

DESCRIPTION

The `ForEachIconDo` function uses the icon action function identified by the `action` parameter to perform an action on the specified icons in the icon suite. You can use these constants in the `selector` parameter to specify the icons on which to perform the action:

```
CONST
   svLarge1Bit       = $00000001; {'ICN#' resource}
   svLarge4Bit       = $00000002; {'icl4' resource}
   svLarge8Bit       = $00000004; {'icl8' resource}
   svSmall1Bit       = $00000100; {'ics#' resource}
   svSmall4Bit       = $00000200; {'ics4' resource}
```

```
svSmall8Bit       = $00000400; {'ics8' resource}
svMini1Bit        = $00010000; {'icm#' resource}
svMini4Bit        = $00020000; {'icm4' resource}
svMini8Bit        = $00040000; {'icm8' resource}
svAllLargeData    = $000000FF; {'ICN#', 'icl4', and 'icl8' }
                               { resources}
svAllSmallData    = $0000FF00; {'ics#', 'ics4', and 'ics8' }
                               { resources}
svAllMiniData     = $00FF0000; {'icm#', 'icm4', and 'icm8' }
                               { resources}
svAll1BitData     = (svLarge1Bit + svSmall1Bit + svMini1Bit);
svAll4BitData     = (svLarge4Bit + svSmall4Bit + svMini4Bit);
svAll8BitData     = (svLarge8Bit + svSmall8Bit + svMini8Bit);
svAllAvailableData= $FFFFFFFF; {all resources of given ID}
```

These constants are additive; that is, you can add several constants to include the corresponding family members in the icon suite.

You can use the `yourDataPtr` parameter to pass a pointer to data or other information required by your icon action function. Typically, you use this parameter to specify which action your icon action function should perform.

`ForEachIconDo` calls your icon action function once for each type of icon specified in the `selector` parameter. `ForEachIconDo` passes to your icon action function a handle to the icon to perform the action on. Your icon action function should perform any action as indicated by the `yourDataPtr` parameter and return a result code. `ForEachIconDo` returns the result code returned by your icon action function. If your icon action function returns a nonzero function result, `ForEachIconDo` immediately returns to the application.

RESULT CODE

`noErr`	0	No error

SEE ALSO

See “Icon Action Functions” beginning on page 5-57 for more information about icon action functions.

Getting and Setting the Label for an Icon Suite

The `GetSuiteLabel` and `SetSuiteLabel` functions allow you to get and set the default label associated with an icon suite.

GetSuiteLabel

You can use the `GetSuiteLabel` function to get the default label setting associated with an icon suite.

```
FUNCTION GetSuiteLabel (theSuite: Handle): Integer;
```

`theSuite` A handle to an icon suite.

DESCRIPTION

The `GetSuiteLabel` function returns, as its function result, the default label setting associated with the specified icon suite. The default label setting is an integer from 1 to 7 that specifies which of the label colors shown in the Finder's Label menu is applied to icons of that suite when your application displays them. `GetSuiteLabel` returns 0 if the suite doesn't have a label.

You can override the default label setting for a suite by specifying a label in the `transform` parameter of the `PlotIconSuite` function.

SEE ALSO

To get information about the color and string for a specific label, you can use the `GetLabel` function, which is described on page 5-41.

SetSuiteLabel

You can use the `SetSuiteLabel` function to specify the default label associated with an icon suite.

```
FUNCTION SetSuiteLabel (theSuite: Handle;
                        theLabel: Integer): OSErr;
```

`theSuite` A handle to an icon suite.

`theLabel` An integer from 1 to 7 that specifies a label for the icon suite, or 0 to set the icon suite's label to none.

DESCRIPTION

The `SetSuiteLabel` function sets the label associated with the specified icon suite. The default label setting helps to determine which of the label colors shown in the Finder's Label menu is applied to icons of that suite when your application displays them.

You can override the default label setting for a suite by specifying a label in the `transform` parameter of the `PlotIconSuite` function. For example, suppose the color currently set for the third label displayed in the Finder's Label menu is red, and the color for the fourth label is green. If you set the default label for a suite using `SetSuiteLabel(theSuite,3)`, then draw an icon from the same suite using `PlotIconSuite` and specifying `ttNone` in the `transform` parameter, the label color red is applied to the icon. However, if you specify `ttLabel4` in the `transform` parameter of the `PlotIconSuite` function, the label color green is applied to the icon.

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	The `theLabel` parameter is greater than 7

SEE ALSO

For a description of the `PlotIconSuite` function, see page 5-35.

Getting Label Information

If you wish to display an icon in your application with the label color and label string associated with a specific label in the Finder, you can use the `GetLabel` function to get the current label information for that label.

GetLabel

You can use the `GetLabel` function to get the color and string used for a given label in the Label menu of the Finder and in the Labels control panel.

```
FUNCTION GetLabel (labelNumber: Integer; VAR labelColor: RGBColor;
                   VAR labelString: Str255): OSErr;
```

`labelNumber`
An integer from 1 to 7 indicating which label's information is requested.

`labelColor`
`GetLabel` returns, in this parameter, the color of the specified label.

`labelString`
`GetLabel` returns, in this parameter, the string associated with the specified label.

DESCRIPTION

The `GetLabel` function returns the color and string used for a specified label in the Label menu of the Finder and in the Labels control panel.

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	The `labelNumber` parameter is greater than 7

SEE ALSO

For information on the `RGBColor` record, see *Inside Macintosh: Imaging With QuickDraw.*

Disposing of Icon Suites

When you are finished with an icon suite, you can release the memory it occupies by calling the `DisposeIconSuite` function.

DisposeIconSuite

You can use the `DisposeIconSuite` function to release the memory occupied by an icon suite.

```
FUNCTION DisposeIconSuite (theIconSuite: Handle;
                           disposeData: Boolean): OSErr;
```

`theIconSuite`
A handle to the icon suite to be disposed of.

`disposeData`
A Boolean value indicating whether or not to dispose of handles in the icon suite that are not associated with a resource fork.

DESCRIPTION

The `DisposeIconSuite` function releases the memory occupied by the specified icon suite. However, `DisposeIconSuite` does not release the memory of any icons explicitly associated with an open resource fork, that is, any handles to icon resource data that your application added to the suite using `GetIconSuite` or `AddIconToSuite`. For handles to icon data that your application added to the icon suite using `AddIconToSuite` (for example, if your application read in an icon resource, detached it, then added the handle to the suite), you can request that `AddIconToSuite` release the memory associated with the handles.

Set `disposeData` to `TRUE` to automatically release icon data that is associated with the specified icon suite but not explicitly associated with a resource fork. If you set `disposeData` to `FALSE`, `DisposeIconSuite` does not dispose of any icon data that is associated with the specified icon suite.

RESULT CODES

`noErr`	0	No error
`memWZErr`	–111	Attempt to operate on a free block

SEE ALSO

For more information on icon suites, see "Creating an Icon Suite" beginning on page 5-30.

Converting an Icon Mask to a Region

The `IconSuiteToRgn`, `IconIDToRgn`, and `IconMethodToRgn` functions create a region from an icon's mask. `IconSuiteToRgn` and `IconIDToRgn` operate on an icon identified by a handle to a suite and an icon ID, respectively. The `IconMethodToRgn` function performs this operation on the icon mask that it obtains with the aid of your icon getter function. Once you have a region that describes the icon mask for a given icon, you can use it to perform accurate hit-testing and outline dragging of the icon in your application.

IconSuiteToRgn

You can use the `IconSuiteToRgn` function to convert, to a region, the icon mask in an icon suite. You specify a rectangle as one of the parameters to this function. `IconSuiteToRgn` determines, from the size of the specified rectangle, which mask from the icon suite to convert. Once it has determined which icon mask to convert, `IconSuiteToRgn` uses the specified rectangle as the bounding box of the region.

```
FUNCTION IconSuiteToRgn (theRgn: RgnHandle; iconRect: Rect;
                         align: IconAlignmentType;
                         theIconSuite: Handle): OSErr;
```

`theRgn` `IconSuiteToRgn` returns a handle to the requested region in this parameter. You must allocate memory for the region handle before calling `IconSuiteToRgn`.

`iconRect` The rectangle in which the icon is to be drawn, specified in local coordinates of the current graphics port. `IconSuiteToRgn` uses this rectangle as the bounding box of the region. `IconSuiteToRgn` determines, from the size of the rectangle specified in this parameter, which icon mask to use from the icon suite.

`align` A value that specifies how `IconSuiteToRgn` should align the region within the rectangle. See the description of `PlotIconSuite` on page 5-35 for a list of constants you can use in this parameter.

`theIconSuite`
A handle to an icon suite.

DESCRIPTION

The `IconSuiteToRgn` function modifies the region referred to by the handle in the `theRgn` parameter. The returned region corresponds to the icon's mask (the mask defined by either an `'ICN#'` or `'ics#'` entry in an icon suite, according to the rectangle and alignment specified in the `iconRect` and `align` parameters).

RESULT CODES

`noErr`	0	No error
`noMaskFoundErr`	–1000	No mask found

IconIDToRgn

You can use the `IconIDToRgn` function to convert, to a region, the icon mask in an icon family. You specify a rectangle as one of the parameters to this function. `IconIDToRgn` determines, from the size of the specified rectangle, which mask from the icon family to convert. Once it has determined which icon mask to convert, `IconIDToRgn` uses the specified rectangle as the bounding box of the region.

```
FUNCTION IconIDToRgn (theRgn: RgnHandle; iconRect: Rect;
                      align: IconAlignmentType;
                      iconID: Integer): OSErr;
```

`theRgn` `IconIDToRgn` returns a handle to the requested region in this parameter. You must allocate memory for the region handle before calling `IconIDToRgn`.

`iconRect` The rectangle in which to draw the icon, specified in local coordinates of the current graphics port. `IconIDToRgn` uses this rectangle as the bounding box of the region. `IconIDToRgn` determines, from the size of the rectangle specified in this parameter, which icon mask to use from the icon family specified by `iconID`.

`align` A value that specifies how `IconIDToRgn` should align the mask within the rectangle. See the description of `PlotIconID` on page 5-20 for a list of constants you can use in this parameter.

`iconID` The resource ID of the icon for which to create a region.

DESCRIPTION

The `IconIDToRgn` function modifies the region referred to by the handle in the `theRgn` parameter. The returned region corresponds to the icon's mask (the mask defined by either an `'ICN#'` or `'ics#'` resource in an icon family, according to the rectangle and alignment specified in the `iconRect` and `align` parameters).

RESULT CODES

`noErr`	0	No error
`noMaskFoundErr`	–1000	No mask found

IconMethodToRgn

You can use the `IconMethodToRgn` function to convert, to a region, the mask for an icon that `IconMethodToRgn` obtains with the aid of your icon getter function.

```
FUNCTION IconMethodToRgn (theRgn: RgnHandle; iconRect: Rect;
                          align: IconAlignmentType;
                          theMethod: IconGetter;
                          yourDataPtr: Ptr): OSErr;
```

`theRgn` `IconMethodToRgn` returns a handle to the requested region in this parameter. You must allocate memory for the region handle before calling `IconMethodToRgn`.

`iconRect` The rectangle in which to draw the icon, specified in local coordinates of the current graphics port. The `IconMethodToRgn` function obtains the data for the icon mask from your icon getter function and then converts the icon mask to a region. `IconMethodToRgn` uses the rectangle specified in this parameter as the bounding box of the region.

`align` A value that specifies how `IconMethodToRgn` should align the region within the rectangle. See the description of `PlotIconID` on page 5-20 for a list of constants you can use in this parameter.

`theMethod` A pointer to an icon getter function.

`yourDataPtr`
A pointer to data that is passed to your icon getter function.

DESCRIPTION

The `IconMethodToRgn` function modifies the region referred to by the handle in the `theRgn` parameter. The region corresponds to the icon's mask (as returned by your icon getter function, and according to the rectangle and alignment specified in the `iconRect` and `align` parameters).

IconMethodToRgn passes to your icon getter function the type of the icon to get and the value specified in the yourDataPtr parameter. The IconMethodToRgn function examines the size of the rectangle and requests the appropriate icon from your icon getter function—an icon of icon type 'ICN#' or 'ics#'. Your icon getter function should return a handle to the data of the requested icon type. The IconMethodToRgn function extracts the mask from the icon data that your icon getter function returns. If your icon getter function returns data that does not correspond to an icon of type 'ICN#' or type 'ics#', IconMethodToRgn attempts to generate a mask from the returned data.

Your icon getter function can get the data for the icon and its mask using whatever method is appropriate to your application. For example, your application might maintain its own cache of icons (and pass a pointer to it in the yourDataPtr parameter) or use its icon getter function to get an icon from the desktop database.

RESULT CODES

noErr	0	No error
noMaskFoundErr	–1000	No mask found

Determining Whether a Point or Rectangle Is Within an Icon

You can use several Icon Utilities routines to perform hit-testing for points or rectangles against a specified icon. You specify a destination rectangle and alignment of the icon within the rectangle as parameters to these functions. The functions use this information to determine whether a specified point or rectangle is within the icon as it appears in the destination rectangle.

The PtInIconSuite and PtInIconID functions hit-test a specified point against the appropriate icon mask from an icon suite or icon family. The PtInIconMethod function hit-tests a specified point against an icon mask obtained with the aid of your icon getter function.

The RectInIconSuite and RectInIconID functions hit-test a specified rectangle against the appropriate icon mask from an icon suite or icon family. The RectInIconMethod function hit-tests a specified rectangle against an icon mask obtained with the aid of your icon getter function.

PtInIconSuite

You can use the `PtInIconSuite` function to determine whether a specified point is within an icon. (A point is considered to be within an icon if the point is within the icon's mask.) For example, you might use this function to determine whether a user clicked an icon in a window of your application. You specify as parameters to `PtInIconSuite` the same rectangle and alignment that you last used to draw the icon. `PtInIconSuite` uses the size of this rectangle to determine which icon mask from the icon suite to use for the operation. The `PtInIconSuite` function uses the location of this rectangle (along with the alignment) to determine whether a specified point is within the icon.

```
FUNCTION PtInIconSuite (testPt: Point; iconRect: Rect;
                        align: IconAlignmentType;
                        theIconSuite: Handle): Boolean;
```

`testPt` The point to be tested, specified in local coordinates of the current graphics port.

`iconRect` The rectangle in which the icon appears, specified in local coordinates of the current graphics port. `PtInIconSuite` determines, from the size of the rectangle specified in this parameter, which icon mask from the icon suite specified by `theIconSuite` to test the point against. `PtInIconSuite` then uses the location of this rectangle (and the location of the icon in the rectangle) to determine whether the specified point is within the icon.

`align` A value that specifies how the icon against which to hit-test is aligned within the rectangle specified by `iconRect`. See the description of `PlotIconSuite` on page 5-35 for a list of constants you can use in this parameter.

`theIconSuite`
A handle to an icon suite.

DESCRIPTION

The `PtInIconSuite` function hit-tests the point specified by `testPt` against the appropriate icon mask from the specified icon suite. `PtInIconSuite` determines which icon mask to use (`'ICN#'` or `'ics#'`) according to the rectangle specified in `iconRect`. The parameters `iconRect` and `align` should be the same as when the icon was last drawn. The `PtInIconSuite` function returns `TRUE` if the point is in the icon mask and `FALSE` if it is not.

PtInIconID

You can use the `PtInIconID` function to determine whether a specified point is within an icon. (A point is considered to be within an icon if the point is within the icon's mask.) For example, you might use this function to determine whether a user clicked an icon in a window of your application. You specify as parameters to `PtInIconID` the same rectangle and alignment that you last used to draw the icon. `PtInIconID` uses the size of this rectangle to determine which icon mask from the icon family to use for the operation. The `PtInIconID` function uses the location of this rectangle (along with the alignment) to determine whether a specified point is within the icon.

```
FUNCTION PtInIconID (testPt: Point; iconRect: Rect;
                     align: IconAlignmentType;
                     iconID: Integer): Boolean;
```

`testPt` The point to be tested, specified in local coordinates of the current graphics port.

`iconRect` The rectangle in which the icon appears, specified in local coordinates of the current graphics port. `PtInIconID` determines, from the size of the rectangle specified in this parameter, which icon mask from the icon family specified by `iconID` to test the point against. `PtInIconID` then uses the location of this rectangle (and the alignment of the icon in the rectangle) to determine whether the specified point is within the icon.

`align` A value that specifies how the icon against which to hit-test is aligned within the rectangle specified by `iconRect`. See the description of `PlotIconID` on page 5-20 for a list of constants you can use in this parameter.

`iconID` A resource ID for an icon family.

DESCRIPTION

The `PtInIconID` function hit-tests the point specified by `testPt` against the appropriate icon mask from the icon family identified by `iconID`, using the destination rectangle and alignment specified by `iconRect` and `align`. The parameters `iconRect` and `align` should be the same as when the icon was last drawn. The `PtInIconID` function returns `TRUE` if the point is in the icon mask and `FALSE` if it is not.

PtInIconMethod

You can use the `PtInIconMethod` function to determine whether a specified point is within an icon. (A point is considered to be within an icon if the point is within the icon's mask.) The `PtInIconMethod` function obtains the icon to test against with the aid of your icon getter function.

```
FUNCTION PtInIconMethod (testPt: Point; iconRect: Rect;
                         align: IconAlignmentType;
                         theMethod: IconGetter;
                         yourDataPtr: Ptr): Boolean;
```

`testPt` The point to be tested, specified in local coordinates of the current graphics port.

`iconRect` The rectangle in which the icon appears, specified in local coordinates of the current graphics port.

`align` A value that specifies how the icon against which to hit-test is aligned within the rectangle specified by `iconRect`. See the description of `PlotIconID` on page 5-20 for a list of constants you can use in this parameter.

`theMethod` A pointer to an icon getter function.

`yourDataPtr` A pointer to data that is passed to your icon getter function.

DESCRIPTION

The `PtInIconMethod` function hit-tests the point specified by `testPt` against an icon obtained with the aid of an icon getter function, using the destination rectangle and alignment specified by `iconRect` and `align`. The parameters `iconRect` and `align` should be the same as when the icon was last drawn. The `PtInIconMethod` function returns `TRUE` if the point is in the icon mask and `FALSE` if it is not.

`PtInIconMethod` passes to your icon getter function the type of icon your function should retrieve (either `'ICN#'` or `'ics#'`) and also passes the value specified in the `yourDataPtr` parameter. The `PtInIconMethod` function examines the size of the specified rectangle and requests the appropriate icon from your icon getter function. Your icon getter function should return a handle to the requested icon's data. The `PtInIconMethod` function extracts the mask from the icon data that your icon getter function returns. If your icon getter function returns data that does not correspond to an icon of type `'ICN#'` or type `'ics#'`, `PtInIconMethod` attempts to generate a mask from the returned data.

Your icon getter function can get the icon's data using whatever method is appropriate to your application. For example, your application might maintain its own cache of icons (and pass a pointer to it in the `yourDataPtr` parameter) or use its icon getter function to get an icon from the desktop database.

SEE ALSO

For more information about icon getter functions, see page 5-58.

RectInIconSuite

You can use the `RectInIconSuite` function to hit-test a rectangle against the appropriate icon mask from an icon suite for a specified destination rectangle and alignment.

```
FUNCTION RectInIconSuite (testRect: Rect; iconRect: Rect;
                          align: IconAlignmentType;
                          theIconSuite: Handle): Boolean;
```

`testRect` The rectangle to be tested, specified in local coordinates of the current graphics port.

`iconRect` The rectangle in which the icon appears, specified in local coordinates of the current graphics port. Like `PtInIconSuite`, `RectInIconSuite` determines, from the size of the rectangle specified in this parameter, which icon mask from the icon suite specified by `theIconSuite` to test the `testRect` parameter against.

`align` A value that specifies how the icon against which to hit-test is aligned within the rectangle specified by `iconRect`. See the description of `PlotIconSuite` on page 5-35 for a list of constants you can use in this parameter.

`theIconSuite`
A handle to an icon suite.

DESCRIPTION

The `RectInIconSuite` function hit-tests the rectangle specified by `testRect` against the appropriate icon mask from the icon suite as it appears in the `iconRect` rectangle. The parameters `iconRect` and `align` should be the same as when the icon was last drawn. The `RectInIconSuite` function returns `TRUE` if the rectangle intersects the icon mask and `FALSE` if it doesn't.

For example, if the coordinates of the `iconRect` parameter are (100,100,116,116) and the icon cache contains entries for each icon family member, `RectInIconSuite` uses the icon mask defined by the `'ics#'` entry. The function aligns this mask (according to the `align` parameter) within the `iconRect` rectangle. The function then intersects the rectangle specified by `testRect` with the icon mask in the `iconRect` rectangle. Continuing with this example, if the icon mask is left-aligned so that its rightmost pixel appears at coordinates (112,112) and the coordinates of `testRect` are (114,114,130,130), then `RectInIconSuite` returns `FALSE`.

RectInIconID

You can use the `RectInIconID` function to hit-test a rectangle against the appropriate icon mask from an icon family for a specified destination rectangle and alignment.

```
FUNCTION RectInIconID (testRect: Rect; iconRect: Rect;
                       align: IconAlignmentType;
                       iconID: Integer): Boolean;
```

`testRect` The rectangle to be tested, specified in local coordinates of the current graphics port.

`iconRect` The rectangle in which the icon appears, specified in local coordinates of the current graphics port. Like `PtInIconID`, `RectInIconID` determines, from the size of the rectangle specified in this parameter, which icon mask from the icon family to test the `testRect` parameter against.

`align` A value that specifies how the icon against which to hit-test is aligned within the rectangle specified by `iconRect`. See the description of `PlotIconID` on page 5-20 for a list of constants you can use in this parameter.

`iconID` A resource ID for an icon family.

DESCRIPTION

The `RectInIconID` function hit-tests the rectangle specified by `testRect` against the appropriate icon mask from the icon family as it appears in the `iconRect` rectangle. The parameters `iconRect` and `align` should be the same as when the icon was last drawn. The `RectInIconID` function returns `TRUE` if the rectangle intersects the icon mask and `FALSE` if it doesn't.

RectInIconMethod

You can use the `RectInIconMethod` function to hit-test a rectangle against an icon obtained by your icon getter function for a specified destination rectangle and alignment.

```
FUNCTION RectInIconMethod (testRect: Rect; iconRect: Rect;
                           align: IconAlignmentType;
                           theMethod: IconGetter;
                           yourDataPtr: Ptr): Boolean;
```

`testRect` The rectangle to be tested, specified in local coordinates of the current graphics port.

`iconRect` The rectangle in which the icon appears, specified in local coordinates of the current graphics port.

`align` A value that specifies how the icon against which to hit-test is aligned within the rectangle specified by `iconRect`. See the description of `PlotIconID` on page 5-20 for a list of constants you can use in this parameter.

`theMethod` A pointer to an icon getter function.

`yourDataPtr` A pointer to data that is passed to your icon getter function.

DESCRIPTION

The `RectInIconMethod` function hit-tests the rectangle specified by `testRect` against an icon mask obtained with the aid of an icon getter function and as the icon appears in the destination rectangle. The parameters `iconRect` and `align` should be the same as when the icon was last drawn. The function returns `TRUE` if the rectangle intersects the icon mask and `FALSE` if it doesn't.

`RectInIconMethod` passes to your icon getter function the type of the icon your function should retrieve and the value specified in the `yourDataPtr` parameter. The `RectInIconMethod` function examines the size of the rectangle and requests the appropriate icon from your icon getter function—an icon of icon type `'ICN#'` or `'ics#'`. Your icon getter function should return a handle to the data of the requested icon type. The `RectInIconMethod` function extracts the mask from the icon data that your icon getter function returns. If your icon getter function returns data that does not correspond to an icon of type `'ICN#'` or type `'ics#'`, `RectInIconMethod` attempts to generate a mask from the returned data.

Your icon getter function can get the data for the icon and its mask using whatever method is appropriate to your application. For example, your application might maintain its own cache of icons (and pass a pointer to it in the `yourDataPtr` parameter) or use its icon getter function to get an icon from the desktop database.

SEE ALSO

For more information about icon getter functions, see page 5-58.

Working With Icon Caches

All the Icon Utilities routines that accept a handle to an icon suite also accept a handle to an icon cache. An icon cache is like an icon suite except that it also contains a pointer to an application-defined icon getter function and a pointer to data that can be used as a reference constant. An icon cache typically does not contain handles to the icon resources for all icon family members. Instead, if the icon cache does not contain an entry for a specific type of icon in an icon family, the Icon Utilities routines call your application's icon getter function to retrieve the data for that icon type.

You can use the routines described in this section to create and manipulate icon caches. To create an empty icon cache, you can use the `MakeIconCache` function, much as you use the `NewIconSuite` function to create an empty icon suite. Before drawing an icon in an icon cache, you can use the `LoadIconCache` function to load icon data for a specified destination rectangle, bit depth of the display device, and alignment.

To get and set the data associated with an icon cache or the icon getter function used with an icon cache, you can use the `GetIconCacheData`, `SetIconCacheData`, `GetIconCacheProc`, and `SetIconCacheProc` functions.

MakeIconCache

You can use the `MakeIconCache` function to get a handle to an empty icon cache, to which you can add icon data using the `LoadIconCache` function.

```
FUNCTION MakeIconCache (VAR theHandle: Handle;
                        makeIcon: IconGetter;
                        yourDataPtr: UNIV Ptr): OSErr;
```

`theHandle` `MakeIconCache` allocates memory for a new icon cache and returns a handle to the new icon cache in this parameter.

`makeIcon` A pointer to an icon getter function to associate with the icon cache.

`yourDataPtr` A pointer to the data to associate with the icon cache.

DESCRIPTION

`MakeIconCache` returns a handle to an empty icon cache in the parameter `theHandle`. The `MakeIconCache` function associates the icon getter function and the value specified in the parameters `makeIcon` and `yourDataPtr` with the new icon cache.

RESULT CODES

noErr	0	No error
memFullErr	–108	Not enough memory in heap zone

LoadIconCache

You can use the LoadIconCache function to load into an icon cache a handle to the appropriate icon data for a specified destination rectangle and the current bit depth, for drawing later with a specified alignment and transform.

```
FUNCTION LoadIconCache (theRect: Rect; align: IconAlignmentType;
                        transform: IconTransformType;
                        theIconCache: Handle): OSErr;
```

theRect The rectangle in which to draw the icon, specified in local coordinates of the current graphics port. LoadIconCache uses the rectangle specified in this parameter and the bit depth of the display device to determine which icon type to load into the cache.

align A value that specifies how to align the icon within the rectangle. See the description of PlotIconSuite on page 5-35 for a list of constants you can use in this parameter.

transform A value that specifies how to modify the appearance of the icon. See the description of PlotIconSuite beginning on page 5-35 for a list of constants you can use in this parameter.

theIconCache
A handle to the icon cache into which to load the icon data.

DESCRIPTION

You can load icon data into an icon cache with the LoadIconCache function for drawing at a later time. For example, this can be useful if you suspect that the icon may be drawn at a time not convenient for loading resource data (for instance, when the resource fork isn't in the current resource chain). The LoadIconCache function uses the same criteria as PlotIconSuite to select the icon to load.

LoadIconCache uses the icon getter function associated with the icon cache to get the appropriate icon. The icon getter function returns a handle to the requested icon data, and LoadIconCache adds the returned handle to the entry for that icon in the icon cache.

After calling `LoadIconCache`, you can pass the same parameters to `PlotIconSuite` to plot the icon data. Note that if you specify an alignment when you call `LoadIconCache`, then call `PlotIconSuite` and specify no alignment, `PlotIconSuite` draws the icon using the alignment that you originally specified to `LoadIconCache`.

RESULT CODES

`noErr`	0	No error
`noMaskFoundErr`	–1000	No mask found

SEE ALSO

For a description of the `PlotIconSuite` function, see page 5-35.

GetIconCacheData

You can use the `GetIconCacheData` function to get the data associated with an icon cache.

```
FUNCTION GetIconCacheData (theCache: Handle;
                           VAR theData: Ptr): OSErr;
```

`theCache` A handle to the icon cache whose data is desired.

`theData` `GetIconCacheData` returns, in this parameter, a pointer to the data associated with the icon cache.

DESCRIPTION

The `GetIconCacheData` function returns, in the parameter `theData`, a pointer to the data associated with the specified icon cache. You associate data with an icon cache when you first create the cache using `MakeIconCache`. You can also set this data using `SetIconCacheData`.

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	The parameter `theCache` must be a handle to an icon cache

SetIconCacheData

You can use the `SetIconCacheData` function to set the data associated with an icon cache.

```
FUNCTION SetIconCacheData (theCache: Handle; theData: Ptr): OSErr;
```

`theCache` A handle to the icon cache whose data is to be set.

`theData` A pointer to the data to set.

DESCRIPTION

The `SetIconCacheData` function sets the data associated with the specified icon cache to the data identified by `theData` parameter.

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	The parameter `theCache` must be a handle to an icon cache

GetIconCacheProc

You can use the `GetIconCacheProc` function to get the icon getter function associated with an icon cache.

```
FUNCTION GetIconCacheProc (theCache: Handle;
                           VAR theProc: IconGetter): OSErr;
```

`theCache` A handle to the icon cache whose icon getter function is desired.

`theProc` `GetIconCacheProc` returns a pointer to the requested icon getter function in this parameter.

DESCRIPTION

The `GetIconCacheProc` function returns, in the parameter `theProc`, a pointer to the icon getter function currently associated with the specified icon cache.

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	The parameter `theCache` must be a handle to an icon cache

SetIconCacheProc

You can use the `SetIconCacheProc` function to set the icon getter function associated with an icon cache.

```
FUNCTION SetIconCacheProc (theCache: Handle;
                           theProc: IconGetter): OSErr;
```

`theCache` A handle to the icon cache whose icon getter function is to be set.

`theProc` A pointer to the icon getter function to set.

DESCRIPTION

The `SetIconCacheProc` function sets the icon getter function for the specified icon cache to the icon getter function specified by the parameter `theProc`.

RESULT CODES

`noErr`	0	No error
`paramErr`	–50	The parameter `theCache` must be a handle to an icon cache

Application-Defined Routines

Your application can provide two functions for use by Icon Utilities routines. If you want to use the `ForEachIconDo` function to perform operations on icons, you must provide an icon action function. If you use icon caches or use any of the routines that end in `Method`, you must provide at least one icon getter function.

Icon Action Functions

You can perform operations on every icon in an icon suite by providing a pointer to an icon action function as a parameter to the `ForEachIconDo` function. The `ForEachIconDo` function calls your icon action function for specified icon resource types.

MyIconAction

The `action` parameter of `ForEachIconDo` must point to a function that uses this syntax:

```
FUNCTION MyIconAction (theType: ResType; VAR theIcon: Handle;
                       yourDataPtr: Ptr): OSErr;
```

`theType` The resource type of the icon.

`theIcon` A handle to the icon on which to perform the operation.

`yourDataPtr`
A pointer to data as specified in the `yourDataPtr` parameter of the `ForEachIconDo` function. When your application calls `ForEachIconDo`, it typically provides in the `yourDataPtr` parameter a value that identifies the action your function should perform.

DESCRIPTION

The `ForEachIconDo` function uses your icon action function to perform actions on specified icons in an icon suite. Your icon action function should return a result code indicating whether it sucessfully performed the action on the icon.

RESULT CODE

`noErr`	0	No error

SEE ALSO

For a description of the `ForEachIconDo` function, see page 5-38.

Icon Getter Functions

If you use icon caches, you must provide at least one icon getter function. You provide a pointer to an icon getter function as a parameter to the `MakeIconCache` function. Subsequent calls to Icon Utilities routines that use icon types not present in the icon cache use the icon getter function associated with the icon cache to return a handle to the icon data.

You can also specify an icon getter function as a parameter to Icon Utilities routines that end in `Method`. Like Icon Utilities routines that work with icon caches, the icon getter function that you provide as a parameter to `PlotIconMethod` should return a handle to the requested icon's data. Note that the icon getter function that you provide as a parameter to `IconMethodToRgn`, `PtInIconMethod`, and `RectInIconMethod` should also return a handle to the requested icon; these three functions then extract the icon mask from the icon data your icon getter function returns.

MyIconGetter

Here is the syntax of an icon getter function:

```
FUNCTION MyIconGetter (theType: ResType;
                       yourDataPtr: Ptr): Handle;
```

`theType` The resource type of the icon.

`yourDataPtr`
If your icon getter was called by an icon cache routine, this parameter contains a pointer to the data associated with the icon cache. Otherwise, this parameter contains the value your application specified in the `yourDataPtr` parameter. For icon caches, you initially set this value when you first create a cache using `MakeIconCache`. You can change this value using `SetIconCacheData`. The icon getter function can use this data as needed.

DESCRIPTION

An icon getter function should return as its function result a handle to the requested icon's data.

The `MakeIconCache` function takes a pointer to an icon getter function for use with a new icon cache. To get and set an existing icon cache's icon getter function, use the `GetIconCacheProc` and `SetIconCacheProc` functions. You can also specify an icon getter function for use by the `PlotIconMethod`, `IconMethodToRgn`, `PtInIconMethod`, and `RectInIconMethod` functions.

SEE ALSO

For descriptions of the `MakeIconCache`, `GetIconCacheProc`, and `SetIconCacheProc` functions, see "Working With Icon Caches" beginning on page 5-53.

For information on the `PlotIconMethod` function, see page 5-22. For a description of the `IconMethodToRgn` function, see "Converting an Icon Mask to a Region" beginning on page 5-43.

For descriptions of the `PtInIconMethod` and `RectInIconMethod` functions, see "Determining Whether a Point or Rectangle Is Within an Icon" beginning on page 5-46.

Summary of the Icon Utilities

Pascal Summary

Constants

```
CONST
   gestaltIconUtilitiesAttr   = 'icon'; {Icon Utilities attributes}
   gestaltIconUtilitiesPresent= 0;      {check this bit in the }
                                        { response parameter}
   {types for icon families}
   large1BitMask              = 'ICN#'; {icon list resource for large icons}
   large4BitData              = 'icl4'; {large 4-bit color icon resource}
   large8BitData              = 'icl8'; {large 8-bit color icon resource}
   small1BitMask              = 'ics#'; {icon list resource for small icons}
   small4BitData              = 'ics4'; {small 4-bit color icon resource}
   small8BitData              = 'ics8'; {small 8-bit color icon resource}
   mini1BitMask               = 'icm#'; {icon list resource for mini icons}
   mini4BitData               = 'icm4'; {4-bit color mini icon}
   mini8BitData               = 'icm8'; {8-bit color mini icon resource}

   {IconAlignmentType values}
   atNone                     = $0;     {no alignment}
   atVerticalCenter           = $1;     {centered vertically}
   atTop                      = $2;     {top aligned}
   atBottom                   = $3;     {bottom aligned}
   atHorizontalCenter         = $4;     {centered horizontally}
   atLeft                     = $8;     {left aligned}
   atRight                    = $C;     {right aligned}
   atAbsoluteCenter           = (atVerticalCenter + atHorizontalCenter);
   atCenterTop                = (atTop + atHorizontalCenter);
   atCenterBottom             = (atBottom + atHorizontalCenter);
   atCenterLeft               = (atVerticalCenter + atLeft);
   atTopLeft                  = (atTop + atLeft);
   atBottomLeft               = (atBottom + atLeft);
   atCenterRight              = (atVerticalCenter + atRight);
   atTopRight                 = (atTop + atRight);
   atBottomRight              = (atBottom + atRight);
```

```
{IconTransformType values}
ttNone                      =  $0;
ttDisabled                  =  $1;
ttOffline                   =  $2;
ttOpen                      =  $3;
ttLabel1                    =  $0100;
ttLabel2                    =  $0200;
ttLabel3                    =  $0300;
ttLabel4                    =  $0400;
ttLabel5                    =  $0500;
ttLabel6                    =  $0600;
ttLabel7                    =  $0700;
ttSelected                  =  $4000;
ttSelectedDisabled          =  (ttSelected + ttDisabled);
ttSelectedOffline           =  (ttSelected + ttOffline);
ttSelectedOpen              =  (ttSelected + ttOpen);

{IconSelectorValue masks}
svLarge1Bit                 = $00000001; {'ICN#' resource}
svLarge4Bit                 = $00000002; {'icl4' resource}
svLarge8Bit                 = $00000004; {'icl8' resource}
svSmall1Bit                 = $00000100; {'ics#' resource}
svSmall4Bit                 = $00000200; {'ics4' resource}
svSmall8Bit                 = $00000400; {'ics8' resource}
svMini1Bit                  = $00010000; {'icm#' resource}
svMini4Bit                  = $00020000; {'icm4' resource}
svMini8Bit                  = $00040000; {'icm8' resource}
svAllLargeData              = $000000FF; {'ICN#', 'icl4', and 'icl8' }
                                         { resources}

svAllSmallData              = $0000FF00; {'ics#', 'ics4', and 'ics8' }
                                         { resources}
svAllMiniData               = $00FF0000; {'icm#', 'icm4', and 'icm8' }
                                         { resources}
svAll1BitData               = (svLarge1Bit + svSmall1Bit + svMini1Bit);
svAll4BitData               = (svLarge4Bit + svSmall4Bit + svMini4Bit);
svAll8BitData               = (svLarge8Bit + svSmall8Bit + svMini8Bit);
svAllAvailableData          = $FFFFFFFF; {all resources of given ID}
```

Data Types

```
TYPE
   CIcon =
   RECORD
      iconPMap:      PixMap;           {the icon's pixel map}
      iconMask:      BitMap;           {the icon's mask}
      iconBMap:      BitMap;           {the icon's bitmap}
      iconData:      Handle;           {handle to the icon's data}
      iconMaskData:                    {the data for the icon's mask}
                     ARRAY[0..0] OF Integer;
   END;

   CIconPtr          = ^CIcon;         {pointer to color icon record}
   CIconHandle       = ^CIconPtr;      {handle to color icon record}

   IconSelectorValue = LongInt;        {icon selector type}
   IconAlignmentType = Integer;        {icon alignment type}
   IconTransformType = Integer;        {icon transform type}

   IconAction        = ProcPtr;        {pointer to action function}
   IconGetter        = ProcPtr;        {pointer to icon getter function}
```

Icon Utilities Routines

Drawing Icons From Resources

```
FUNCTION PlotIconID         (theRect: Rect; align: IconAlignmentType;
                             transform: IconTransformType;
                             theResID: Integer): OSErr;
FUNCTION PlotIconMethod     (theRect: Rect; align: IconAlignmentType;
                             transform: IconTransformType;
                             theMethod: IconGetter;
                             yourDataPtr: UNIV Ptr): OSErr;
PROCEDURE PlotIcon          (theRect: Rect; theIcon: Handle);
FUNCTION PlotIconHandle     (theRect: Rect; align: IconAlignmentType;
                             transform: IconTransformType;
                             theIcon: Handle): OSErr;
PROCEDURE PlotCIcon         (theRect: Rect; theIcon: CIconHandle);
FUNCTION PlotCIconHandle    (theRect: Rect; align: IconAlignmentType;
                             transform: IconTransformType;
                             theCIcon: CIconHandle): OSErr;
```

```
FUNCTION PlotSICNHandle     (theRect: Rect; align: IconAlignmentType;
                             transform: IconTransformType;
                             theSICN: Handle): OSErr;
```

Getting Icons From Resources That Don't Belong to an Icon Family

```
FUNCTION GetIcon            (iconID: Integer): Handle;
FUNCTION GetCIcon           (iconID: Integer): CIconHandle;
```

Disposing of Icons

```
PROCEDURE DisposeCIcon      (theIcon: CIconHandle);
```

Creating an Icon Suite

```
FUNCTION GetIconSuite       (VAR theIconSuite: Handle; theResID: Integer;
                             selector: IconSelectorValue): OSErr;
FUNCTION NewIconSuite       (VAR theIconSuite: Handle): OSErr;
FUNCTION AddIconToSuite     (theIconData: Handle; theSuite: Handle;
                             theType: ResType): OSErr;
```

Getting Icons From an Icon Suite

```
FUNCTION GetIconFromSuite   (VAR theIconData: Handle; theSuite: Handle;
                             theType: ResType): OSErr;
```

Drawing Icons From an Icon Suite

```
FUNCTION PlotIconSuite      (theRect: Rect; align: IconAlignmentType;
                             transform: IconTransformType;
                             theIconSuite: Handle): OSErr;
```

Performing Operations on Icons in an Icon Suite

```
FUNCTION ForEachIconDo      (theSuite: Handle; selector: IconSelectorValue;
                             action: IconAction; yourDataPtr: Ptr): OSErr;
```

Getting and Setting the Label for an Icon Suite

```
FUNCTION GetSuiteLabel      (theSuite: Handle): Integer;
FUNCTION SetSuiteLabel      (theSuite: Handle; theLabel: Integer): OSErr;
```

Getting Label Information

```
FUNCTION GetLabel           (labelNumber: Integer; VAR labelColor: RGBColor;
                             VAR labelString: Str255): OSErr;
```

Disposing of Icon Suites

```
FUNCTION DisposeIconSuite   (theIconSuite: Handle;
                             disposeData: Boolean): OSErr;
```

Converting an Icon Mask to a Region

```
FUNCTION IconSuiteToRgn     (theRgn: RgnHandle; iconRect: Rect;
                             align: IconAlignmentType;
                             theIconSuite: Handle): OSErr;
FUNCTION IconIDToRgn        (theRgn: RgnHandle; iconRect: Rect;
                             align: IconAlignmentType;
                             iconID: Integer): OSErr;
FUNCTION IconMethodToRgn    (theRgn: RgnHandle; iconRect: Rect;
                             align: IconAlignmentType;
                             theMethod: IconGetter;
                             yourDataPtr: Ptr): OSErr;
```

Determining Whether a Point or Rectangle Is Within an Icon

```
FUNCTION PtInIconSuite      (testPt: Point; iconRect: Rect;
                             align: IconAlignmentType;
                             theIconSuite: Handle): Boolean;
FUNCTION PtInIconID         (testPt: Point; iconRect: Rect;
                             align: IconAlignmentType;
                             iconID: Integer): Boolean;
FUNCTION PtInIconMethod     (testPt: Point; iconRect: Rect;
                             align: IconAlignmentType;
                             theMethod: IconGetter;
                             yourDataPtr: Ptr): Boolean;
FUNCTION RectInIconSuite    (testRect: Rect; iconRect: Rect;
                             align: IconAlignmentType;
                             theIconSuite: Handle): Boolean;
FUNCTION RectInIconID       (testRect: Rect; iconRect: Rect;
                             align: IconAlignmentType;
                             iconID: Integer): Boolean;
FUNCTION RectInIconMethod   (testRect: Rect; iconRect: Rect;
                             align: IconAlignmentType;
                             theMethod: IconGetter;
                             yourDataPtr: Ptr): Boolean;
```

Working With Icon Caches

```
FUNCTION MakeIconCache      (VAR theHandle: Handle;
                             makeIcon: IconGetter;
                             yourDataPtr: UNIV Ptr): OSErr;
```

```
FUNCTION LoadIconCache      (theRect: Rect; align: IconAlignmentType;
                             transform: IconTransformType;
                             theIconCache: Handle): OSErr;
FUNCTION GetIconCacheData   (theCache: Handle; VAR theData: Ptr): OSErr;
FUNCTION SetIconCacheData   (theCache: Handle; theData: Ptr): OSErr;
FUNCTION GetIconCacheProc   (theCache: Handle;
                             VAR theProc: IconGetter): OSErr;
FUNCTION SetIconCacheProc   (theCache: Handle; theProc: IconGetter): OSErr;
```

Application-Defined Routines

Icon Action Functions

```
FUNCTION MyIconAction       (theType: ResType; VAR theIcon: Handle;
                             yourDataPtr: Ptr): OSErr;
```

Icon Getter Functions

```
FUNCTION MyIconGetter       (theType: ResType; yourDataPtr: Ptr): Handle;
```

C Summary

Constants

```
enum {
   #define gestaltIconUtilitiesAttr 'icon'   /*Icon Utilities attributes*/
   gestaltIconUtilitiesPresent      = 0      /*check this bit in the */
                                             /* response parameter*/
};
/*types for icon families*/
#define large1BitMask         'ICN#' /*icon list resource for large icons*/
#define large4BitData         'icl4' /*large 4-bit color icon resource*/
#define large8BitData         'icl8' /*large 8-bit color icon resource*/
#define small1BitMask         'ics#' /*icon list resource for small icons*/
#define small4BitData         'ics4' /*small 4-bit color icon resource*/
#define small8BitData         'ics8' /*small 8-bit color icon resource*/
#define mini1BitMask          'icm#' /*icon list resource for mini icons*/
#define mini4BitData          'icm4' /*mini 4-bit color icon resource*/
#define mini8BitData          'icm8' /*mini 4-bit color icon resource*/
```

```
enum { /*IconAlignmentType values*/
   atNone                  = 0x0,              /*no alignment*/
   atVerticalCenter        = 0x1,              /*centered vertically*/
   atTop                   = 0x2,              /*top aligned*/
   atBottom                = 0x3,              /*bottom aligned*/
   atHorizontalCenter      = 0x4,              /*centered horizontally*/
   atAbsoluteCenter        = (atVerticalCenter | atHorizontalCenter),
   atCenterTop             = (atTop | atHorizontalCenter),
   atCenterBottom          = (atBottom | atHorizontalCenter),
   atLeft                  = 0x8,              /*left aligned*/
   atCenterLeft            = (atVerticalCenter | atLeft),
   atTopLeft               = (atTop | atLeft),
   atBottomLeft            = (atBottom | atLeft),
   atRight                 = 0xC,              /*right aligned*/
   atCenterRight           = (atVerticalCenter | atRight),
   atTopRight              = (atTop | atRight),
   atBottomRight           = (atBottom | atRight),
};

enum { /*IconTransformType values*/
   ttNone                  =  0x0,
   ttDisabled              =  0x1,
   ttOffline               =  0x2,
   ttOpen                  =  0x3,
   ttLabel1                =  0x0100,
   ttLabel2                =  0x0200,
   ttLabel3                =  0x0300,
   ttLabel4                =  0x0400,
   ttLabel5                =  0x0500,
   ttLabel6                =  0x0600,
   ttLabel7                =  0x0700,
   ttSelected              =  0x4000,
   ttSelectedDisabled      =  (ttSelected | ttDisabled),
   ttSelectedOffline       =  (ttSelected | ttOffline),
   ttSelectedOpen          =  (ttSelected | ttOpen),
};

enum { /*IconSelectorValue masks*/
   svLarge1Bit             = 0x00000001, /*'ICN#' resource*/
   svLarge4Bit             = 0x00000002, /*'icl4' resource*/
   svLarge8Bit             = 0x00000004, /*'icl8' resource*/
   svSmall1Bit             = 0x00000100, /*'ics#' resource*/
   svSmall4Bit             = 0x00000200, /*'ics4' resource*/
```

```
   svSmall8Bit               = 0x00000400, /*'ics8' resource*/
   svMini1Bit                = 0x00010000, /*'icm#' resource*/
   svMini4Bit                = 0x00020000, /*'icm4' resource*/
   svMini8Bit                = 0x00040000, /*'icm8' resource*/
   svAllLargeData            = 0x000000FF, /*'ICN#', 'icl4', and 'icl8' */
                                           /* resources*/
   svAllSmallData            = 0x0000FF00, /*'ics#', 'ics4', and 'ics8' */
                                           /* resources*/
   svAllMiniData             = 0x00FF0000, /*'icm#', 'icm4', and 'icm8' */
                                           /* resources*/
   svAll1BitData             = (svLarge1Bit | svSmall1Bit | svMini1Bit),
   svAll4BitData             = (svLarge4Bit | svSmall4Bit | svMini4Bit),
   svAll8BitData             = (svLarge8Bit | svSmall8Bit | svMini8Bit),
   svAllAvailableData        = (long)0xFFFFFFFF /*all resources of given ID*/
};
```

Data Types

```
struct CIcon {
   PixMap iconPMap;                    /*the icon's pixel map*/
   BitMap iconMask;                    /*the icon's mask*/
   BitMap iconBMap;                    /*the icon's bitmap*/
   Handle iconData;                    /*handle to the icon's data*/
   short iconMaskData:                 /*the data for the icon's mask*/
};

typedef struct CIcon CIcon;
typedef Cicon *CIconPtr, **CIconHandle; /*ptr, handle to color icon record*/

typedef unsigned long IconSelectorValue;/*icon selector type*/
typedef short IconAlignmentType;        /*icon alignment type*/
typedef short IconTransformType;        /*icon transform type*/

/*pointer to action function*/
typedef pascal OSErr (*IconActionProcPtr)(ResType theType, Handle *theIcon,
                                          void *yourDataPtr);
typedef IconActionProcPtr IconAction;

/*pointer to icon getter function*/
typedef pascal Handle (*IconGetterProcPtr)(ResType theType,
                                           void *yourDataPtr);
typedef IconGetterProcPtr IconGetter;
```

Icon Utilities Routines

Drawing Icons From Resources

```
pascal OSErr PlotIconID     (const Rect *theRect, IconAlignmentType align,
                             IconTransformType transform, short theResID);
pascal OSErr PlotIconMethod
                            (const Rect *theRect, IconAlignmentType align,
                             IconTransformType transform,
                             IconGetterProcPtr theMethod,
                             void *yourDataPtr);
pascal void PlotIcon        (const Rect *theRect, Handle theIcon);
pascal OSErr PlotIconHandle
                            (const Rect *theRect, IconAlignmentType align,
                             IconTransformType transform, Handle theIcon);
pascal OSErr PlotCIcon      (const Rect *theRect, CIconHandle theIcon);
pascal OSErr PlotCIconHandle
                            (const Rect *theRect, IconAlignmentType align,
                             IconTransformType transform,
                             CIconHandle theCIcon);
pascal OSErr PlotSICNHandle
                            (const Rect *theRect, IconAlignmentType align,
                             IconTransformType transform, Handle theSICN);
```

Getting Icons From Resources That Don't Belong to an Icon Family

```
pascal Handle GetIcon       (short iconID);
pascal CIconHandle GetCIcon
                            (short iconID);
```

Disposing of Icons

```
pascal OSErr DisposeCIcon   (CIconHandle theIcon);
```

Creating an Icon Suite

```
pascal OSErr GetIconSuite   (Handle *theIconSuite, short theResID,
                             IconSelectorValue selector);
pascal OSErr NewIconSuite   (Handle *theIconSuite);
```

```
pascal OSErr AddIconToSuite
                            (Handle theIconData, Handle theSuite,
                             ResType theType);
```

Getting Icons From an Icon Suite

```
pascal OSErr GetIconFromSuite
                            (Handle *theIconData, Handle theSuite,
                             ResType theType);
```

Drawing Icons From an Icon Suite

```
pascal OSErr PlotIconSuite  (const Rect *theRect, IconAlignmentType align,
                             IconTransformType transform,
                             Handle theIconSuite);
```

Performing Operations on Icons in an Icon Suite

```
pascal OSErr ForEachIconDo  (Handle theSuite, IconSelectorValue selector,
                             IconActionProcPtr action, void *yourDataPtr);
```

Getting and Setting the Label for an Icon Suite

```
pascal short GetSuiteLabel  (Handle theSuite);
pascal OSErr SetSuiteLabel  (Handle theSuite, short theLabel);
```

Getting Label Information

```
pascal OSErr GetLabel       (short labelNumber, RGBColor *labelColor,
                             Str255 labelString);
```

Disposing of Icon Suites

```
pascal OSErr DisposeIconSuite
                            (Handle theIconSuite, Boolean disposeData);
```

Converting an Icon Mask to a Region

```
pascal OSErr IconSuiteToRgn
                            (RgnHandle theRgn, const Rect *iconRect,
                             IconAlignmentType align, Handle theIconSuite);
pascal OSErr IconIDToRgn    (RgnHandle theRgn, const Rect *iconRect,
                             IconAlignmentType align, short iconID);
```

```
pascal OSErr IconMethodToRgn
                            (RgnHandle theRgn, const Rect *iconRect,
                             IconAlignmentType align,
                             IconGetterProcPtr theMethod,
                             void *yourDataPtr);
```

Determining Whether a Point or Rectangle Is Within an Icon

```
pascal Boolean PtInIconSuite
                            (Point testPt, const Rect *iconRect,
                             IconAlignmentType align, Handle theIconSuite);
pascal Boolean PtInIconID   (Point testPt, const Rect *iconRect,
                             IconAlignmentType align, short iconID);
pascal Boolean PtInIconMethod
                            (Point testPt, const Rect *iconRect,
                             IconAlignmentType align,
                             IconGetterProcPtr theMethod,
                             void *yourDataPtr);
pascal Boolean RectInIconSuite
                            (const Rect *testRect, const Rect *iconRect,
                             IconAlignmentType align, Handle theIconSuite);
pascal Boolean RectInIconID
                            (const Rect *testRect, const Rect *iconRect,
                             IconAlignmentType align, short iconID);
pascal Boolean RectInIconMethod
                            (const Rect *testRect, const Rect *iconRect,
                             IconAlignmentType align,
                             IconGetterProcPtr theMethod,
                             void *yourDataPtr);
```

Working With Icon Caches

```
pascal OSErr MakeIconCache  (Handle *theHandle, IconGetterProcPtr makeIcon,
                             void *yourDataPtr);
pascal OSErr LoadIconCache  (const Rect *theRect, IconAlignmentType align,
                             IconTransformType transform,
                             Handle theIconCache);
```

```
pascal OSErr GetIconCacheData
                            (Handle theCache, void **theData);
pascal OSErr SetIconCacheData
                            (Handle theCache, void *theData);
pascal OSErr GetIconCacheProc
                            (Handle theCache, IconGetter *theProc);
pascal OSErr SetIconCacheProc
                            (Handle theCache, IconGetter theProc);
```

Application-Defined Routines

Icon Action Functions

```
pascal OSErr MyIconAction   (ResType theType, Handle *theIcon,
                             void *yourDataPtr);
```

Icon Getter Functions

```
pascal Handle MyIconGetter  (ResType theType, void *yourDataPtr);
```

Assembly-Language Summary

Data Structure

Color Icon Data Structure

0	iconPMap	60 bytes	icon's pixel map
50	iconMask	14 bytes	icon's mask
64	iconBMap	14 bytes	icon's bitmap
78	iconData	4 bytes	handle to icon's data
82	iconMaskData	variable	data for icon's mask

Trap Macros

Trap Macros Requiring Routine Selectors

_IconDispatch

Selector	Routine
$0702	NewIconSuite
$1702	GetSuiteLabel
$0203	DisposeIconSuite
$1603	SetSuiteLabel
$1904	GetIconCacheData
$1A04	SetIconCacheData
$1B04	GetIconCacheProc
$1C04	SetIconCacheProc
$0005	PlotIconID
$0105	GetIconSuite
$0B05	GetLabel
$0306	PlotIconSuite
$0406	MakeIconCache
$0606	LoadIconCache
$0806	AddIconToSuite
$0906	GetIconFromSuite
$0D06	PtInIconID
$1006	RectInIconID
$1306	IconIDToRgn
$1D06	PlotIconHandle
$1E06	PlotSICNHandle
$1F06	PlotCIconHandle
$0E07	PtInIconSuite
$1107	RectInIconSuite
$1407	IconSuiteToRgn
$0A08	ForEachIconDo
$0508	PlotIconMethod
$0F09	PtInIconMethod
$1209	RectInIconMethod
$1509	IconMethodToRgn

Result Codes

`noErr`	0	No error
`paramErr`	–50	Error in parameter list
`memFullErr`	–108	Not enough memory in heap zone
`memWZErr`	–111	Attempt to operate on a free block
`resNotFound`	–192	Resource not found
`noMaskFoundErr`	–1000	Cannot find or create mask for the icon family