# OFFSCREEN GRAPHICS WORLDS, PICTURES, CURSORS, AND ICONS

Includes Demonstration Program GWorldPicCursIcon

# Offscreen Graphics Worlds

#### Introduction

An **offscreen graphics world** may be regarded as a virtual screen on which your application can draw a complex image without the user seeing the various steps your application takes before completing the image. The image in an offscreen graphics world is drawn into a part of memory not used by the video device. It therefore remains hidden from the user.

One of the key advantages of using an offscreen graphics ports is that it allows you to improve onscreen drawing speed and visual smoothness. For example, suppose your application draws multiple graphics objects in a window and then needs to update part of that window. If your image is very complex, your application can copy it from an offscreen graphics world to the screen faster than it can repeat all of the steps necessary to draw the image on-screen. At the same time, the inelegant visual effects associated with the time-consuming drawing a large number of separate objects are avoided.

# Creating an Offscreen Graphics World

You create an offscreen graphics world with the NewGWorld function. NewGWorld creates a new offscreen colour graphics port, a new offscreen pixel map, and either a new GDevice structure or a link to an existing one. NewGWorld returns a pointer of type GWorldPtr which points to a colour graphics port:

typedef CGrafPtr GWorldPtr;

When you use <code>NewGWorld</code>, you can specify a pixel depth, a boundary rectangle (which also becomes the port rectangle), a colour table, a <code>GDevice</code> structure, and option flags for memory allocation. Passing 0 as the pixel depth, the window's port rectangle as the offscreen world's boundary rectangle, <code>NULL</code> for both the colour table and the <code>GDevice</code> structure and 0 as the options flags:

- Provides your application with the default behaviour of NewGWorld.
- Allows QuickDraw to optimise the CopyBits, CopyMask, and CopyDeepMask functions used to copy the image into the window's port rectangle.

# Setting the Colour Graphics Port for an Offscreen Graphics World

Before drawing into the offscreen graphics port, you should save the current colour graphics port and the current device's <code>GDevice</code> structure by calling <code>GetGWorld</code>. The offscreen graphics port should then be made the current port by a call to <code>SetGWorld</code>. After drawing into the offscreen graphics world, you should call <code>SetGWorld</code> to restore the saved colour graphics port as the current colour graphics port.

Set GWorl d takes two parameters (port and gdh). If the port parameter is of type CGrafPtr, the current port is set to the port specified in the port parameter and the current device is set to the device specified in the gdh parameter. If the port parameter is of type GWorl dPtr, the current port is set to the port specified in the port parameter, the gdh parameter is ignored, and the current device is set to the device attached to the offscreen graphics world.

# Preparing to Draw Into an Offscreen Graphics World

After setting the offscreen graphics world as the current port, you should use the <code>GetGWorldPixMap</code> function to get a handle to the offscreen pixel map. This is required as the parameter in a call to the <code>LockPixels</code> function, which you must call before drawing to, or copying from, an offscreen graphics world.

LockPi xels prevents the base address of an offscreen pixel image from being moved while you draw into it or copy from it. If the base address for an offscreen pixel image has not been purged by the Memory Manager, or if its base address is not purgeable, LockPi xels returns true. If LockPi xels returns false, your application should either call the UpdateGWorld function to reallocate the offscreen pixel image and then reconstruct it, or draw directly into an onscreen graphics port.

As a related matter, note that the <code>baseAddr</code> field of the <code>PixMap</code> structure for an offscreen graphics world contains a handle, whereas the <code>baseAddr</code> field for an onscreen pixel map contains a pointer. You must use the <code>GetPixBaseAddr</code> function to obtain a pointer to the <code>PixMap</code> structure for an offscreen graphics world

# Copying an Offscreen Image into a Window

After drawing the image in the offscreen graphics world, your application should call SetGWorld to restore the active window as the current graphics port.

The image is copied from the offscreen graphics world into the window using <code>CopyBits</code> (or, if masking is required, <code>CopyMask</code> or <code>CopyDeepMask</code>). Specify the offscreen graphics world as the source image for <code>CopyBits</code> and specify the window as its destination. Note that <code>CopyBits</code>, <code>CopyMask</code> and <code>CopyDeepMask</code> expect their source and destination parameters to be pointers to bit maps, not pixel maps. Accordingly, you must coerce the offscreen graphic's world's <code>GWorldPtr</code> data type to a data structure of type <code>GrafPtr</code>. Similarly, whenever a colour graphics port is your destination, you must coerce the window's <code>CGrafPtr</code> data type to data type <code>GrafPtr</code>.

As long as you are drawing into an offscreen graphics world or copying an image from it, you must leave its pixel image locked. When you are finished drawing into, and copying from, an offscreen graphics world, call UnlockPixels. Calling UnlockPixels will assist in preventing heap fragmentation.

# Updating an Offscreen Graphics World

If, for example, you are using an offscreen graphics world to support the window updating process, you can use <code>UpdateGWorld</code> to carry certain changes affecting the window (for example, resizing, changes to the pixel depth of the screen, or modifications to the colour table) through to the offscreen graphics world. <code>UpdateGWorld</code> allows you to change the pixel depth, boundary rectangle, or colour table for an existing offscreen graphics world without recreating it and redrawing its contents.

# Disposing of an Offscreen Graphics World

Call Di sposeGWorld when your application no longer needs the offscreen graphics world.

#### Introduction

QuickDraw provides a simple set of functions for recording a collection of its drawing commands and then playing the recording back later. Such a collection of drawing commands, as well as the resulting image, is called a **picture**. Pictures provide a common medium for the sharing of image data. They make it easier for your application to draw complex images defined in other applications, and vice versa.

When you use OpenCPicture to begin defining a picture, QuickDraw collects your subsequent drawing commands in a data structure of type Picture. By using DrawPicture, you can draw onscreen the picture defined by the instructions stored in the Picture structure.

#### Picture Format

The OpenCPicture function creates pictures in the **extended version 2 format**. This format permits your application to specify resolutions for pictures in colour or black-and-white.

#### Historical Note

During QuickDraw's evolution, three different formats evolved for the data contained in a Picture structure. The extended version 2 format is the latest format. The original format, the version 1 format, was created by the <code>OpenPicture</code> function on machines without Color QuickDraw or whenever the current graphics port was a basic graphics port. Pictures created in this format supported only black-and-white drawing operations at 72 dpi (dots per inch). The version 2 format was created by the <code>OpenPicture</code> function on machines with Color QuickDraw when the current graphics port was a colour graphics port. Pictures created in this format supported colour drawing operations at 72 dpi.

#### The Picture Structure

The Picture structure is as follows:

```
struct Picture
{
   short picSize; // For a version 1 picture: its size.
   Rect picFrame; // Bounding rectangle for the picture.
};
typedef struct Picture Picture;
typedef Picture *PicPtr;
typedef PicPtr *PicHandle;
```

#### Field Descriptions

pi cSi ze

The information in this field is useful only for version 1 pictures, which cannot exceed 32 KB in size. Version 2 and extended version 2 pictures can be larger than 32 KB. To maintain compatibility with the version 1 picture format, the picSize field was not changed for version 2 or extended version 2 picture formats.

You should use the Memory Manager function GetHandleSize to determine the size of a picture in memory, the File Manager function PBGetFInfo to determine the size of a picture in a file of type 'PICT', and the Resource Manager function MaxSizeResource to determine the size of a picture in a resource of type 'PICT'.

pi cFrame

Contains the bounding rectangle for the picture. DrawPicture uses this rectangle to scale the picture when you draw into a differently sized rectangle.

Compact drawing commands and picture comments constitute the rest of the structure, which is of variable length.

# **Opcodes: Drawing Commands and Picture Comments**

The variable length field in a Picture structure contains data in the form of **opcodes**, which are values that DrawPicture uses to determine what objects to draw or what mode to change for subsequent drawing.

In addition to compact drawing commands, opcodes can also specify picture comments, which are created using PicComment. A picture comment contains data or commands for special processing by output devices, such as PostScript printers. If your application requires capability beyond that provided by QuickDraw drawing functions, PicComment allows your application to pass data or commands direct to the output device.

You typically use QuickDraw commands when drawing to the screen and picture comments to include special drawing commands for printers only.

# 'PICT' Files, 'PICT' Resources, and 'PICT' Scrap Format

QuickDraw provides functions for creating and drawing pictures. File Manager and Resource Manager functions are used to read pictures from, and write pictures to, a disk. Scrap Manager functions are used to read pictures from, and write pictures to, the scrap<sup>1</sup>.

A picture can be stored as a 'PICT' resource in the resource fork of any file type. A picture can also be stored in the data fork of a file of type 'PICT'. The data fork of a 'PICT' file contains a 512-byte header that applications can use for their own purposes.

For each application, the Scrap Manager maintains a storage area to hold the last data cut or copied by the user. The area that is available to your application for this purpose is called the **scrap**. All applications that support copy-and-paste operations read data from, and write data to, the scrap. The 'PICT' scrap format is one of two standard scrap formats. (The other is 'TEXT'.)

#### The Picture Utilities

In addition to the QuickDraw functions for creating and drawing pictures, system software provides a group of functions called the **Picture Utilities** for examining the content of pictures. You typically use the Picture Utilities before displaying a picture.

The Picture utilities allow you to gather colour, comment, font, resolution, and other information about pictures. You might use the Picture Utilities, for example, to determine the 256 most-used colours in a picture, and then use the Palette Manager to make those colours available for the window in which the application needs to draw the picture.

# **Creating Pictures**

As previously stated, you use the OpenCPicture function to begin defining a picture. OpenCPicture collects your subsequent drawing commands in a new Picture structure. To complete the collection of drawing (and picture comment) commands which define your picture, call ClosePicture.

You pass information to OpenCPi cture in the form of an OpenCPi cParams structure:

```
struct OpenCPicParams
  Rect
         srcRect;
                     // Optimal bounding rectangle.
  Fixed hRes;
                     // Best horizontal resolution.
  Fixed vRes:
                     // Best vertical resolution.
  short version;
                     // Set to -2.
                     // (Reserved.
  short reserved1;
                                     Set to 0.)
        reserved2:
                     // (Reserved.
                                    Set to 0.)
typedef struct OpenCPicParams OpenCPicParams;
```

<sup>&</sup>lt;sup>1</sup> See Chapter 18 — Scrap.

This structure provides a simple mechanism for specifying resolutions when creating images. For example, applications that create pictures from scanned images can specify resolutions higher than 72 dpi.

#### Clipping Region

You should always use ClipRect to specify a clipping region appropriate to your picture before calling OpenCPicture. If you do not specify a clipping region, OpenCPicture uses the clipping region specified in the current colour graphics port. If this region is very large (as it is when the graphics port is initialised, being set to the size of the coordinate plane by that initialisation) and you scale the picture when drawing it, the clipping region can become invalid when DrawPicture scales the clipping region, in which case your picture will not be drawn. On the other hand, if the graphics port specifies a small clipping region, part of your drawing may be clipped when you draw it. Setting the clipping region equal to the port rectangle of the current graphics port always sets a valid clipping region.

# **Opening and Drawing Pictures**

Using File Manager functions, your application can retrieve pictures saved in 'PICT' files.<sup>2</sup> Using the GetPicture function, your application can retrieve pictures saved in the resource forks of other file types. Using the Scrap Manager function GetScrap, your application can retrieve pictures stored in the scrap.

When the picture is retrieved, you should call <code>DrawPicture</code> to draw the picture. The second parameter taken by <code>DrawPicture</code> is the destination rectangle, which should be specified in coordinates local to the current graphics port. <code>DrawPicture</code> shrinks or stretches the picture as necessary to make it fit into this rectangle.

When you are finished using a picture stored as a 'PICT' resource, you should use the resource Manager function ReleaseResource to release its memory.

# Saving Pictures

After creating or changing pictures, your application should allow the user to save them. To save a picture in a 'PICT' file, you should use the appropriate File Manager functions.<sup>2</sup> (Remember that the first 512 bytes of a 'PICT' file are reserved for your application's own purposes.) To save pictures in a 'PICT' resource, you should use the appropriate Resource Manager functions. To place a picture in the Scrap (for example, to respond to the user choosing the Copy command to copy a picture to the clipboard), you should use the Scrap Manager function PutScrap.

# **Gathering Picture Information**

GetPictInfo may be used to gather information about a single picture, and GetPixMapInfo may be used to gather colour information about a single pixel map or bit map. Each of these functions returns colour and resolution information in a PictInfo structure. A PictInfo structure can also contain information about the drawing objects, fonts, and comments in a picture.

 $<sup>^2</sup>$  The demonstration program at Chapter 16 — Files shows how to read pictures from, and save pictures to, files of type ' PICT' .

#### Introduction

A cursor is a 256-pixel, black-and-white image in a 16-by-16 pixel square usually defined by an application in a cursor ('CURS') or colour cursor ('CURS') resource.

# Cursor Movement, Hot Spot, Visibility, Colour and Shape

#### **Cursor Movement**

Whenever the user moves the mouse, the low-level interrupt-driven mouse functions move the cursor to a new location on the screen. Your application does not need to do anything to move the cursor.

# **Cursor Hot Spot**

One point in the cursor's image is designated as the **hot spot**, which in turn points to a location on the screen. The hot spot is the part of the pointer that must be positioned over a screen object before mouse clicks can have an effect on that object. Fig 1 illustrates two cursors and their hot spot points. Note that the hot spot is a point, not a bit.

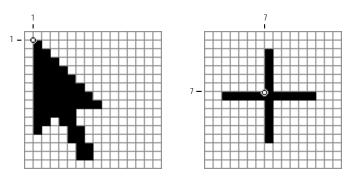


FIG 1 - HOT SPOTS IN CURSORS

# **Cursor Visibility**

In general, you should always make the cursor visible to your application, although there are a few cases where the cursor should not be visible. For example, in a text-editing application, the cursor should be made invisible, and the insertion point made to blink, when the user begins entering text. In such cases, the cursor should be made visible again only when the user moves the mouse.

#### **Cursor Colour**

When the cursor is used for choosing or selecting, it should remain black. You may want to display a colour cursor when the user is drawing or typing in colour. To ensure visibility over any background, colour cursors should generally be outlined in black.

#### **Cursor Shape**

Your application should change the shape of the cursor in the following circumstances:

- To indicate that the user is over a certain area of the screen. For example, when the cursor is in the menu bar, it should usually have an arrow shape. When the user moves the cursor over a text document, your application should change the cursor to the I-beam shape.
- To provide feedback about the status of the computer system. For example, if an operation will take a second or two, you should provide feedback to the user by changing the cursor to the wristwatch cursor (see Fig 2). If the operation takes several seconds and the user can do nothing

in your application but stop the operation, wait until it is completed, or switch to another application, you should display an animated cursor (see below).<sup>3</sup>

The System file in the System Folder contains 'CURS' resources for the common cursors shown at Fig 2.



FIG 2 - THE I-BEAM, CROSSHAIRS, PLUS SIGN, AND WRISTWATCH CURSORS

The following constants represent the 'CURS' resource IDs for the cursors shown at Fig 2:

Constant	Value	Description
i BeamCursor	1	Used in text editing.
crossCursor	2	Often used for manipulating graphics.
plusCursor	3	Often used for selecting fields in an array.
watchCursor	4	Used when a short operation is in progress.

# **Creating Custom Non-Animated Cursors Resources**

To create custom non-animated cursors, you need to:

- Define black-and-white cursors as 'CURS' resources in the resource file of your application.
- If you want to display colour cursors, define colour cursors in 'crsr' resources in the resource file of your application.

# **Changing Cursor Shape and Hiding Cursors**

# **Changing Cursor Shape**

To change cursor shape, your application must get a handle to the relevant cursor (either a custom cursor or one of the system cursors shown at Fig 2) by specifying its resource ID in a call to <code>GetCursor</code> or <code>GetCCursor</code> returns a handle to a <code>Cursor</code> structure. <code>GetCCursor</code> returns a handle to a <code>CCrsr</code> structure. The address of the <code>Cursor</code> or <code>CCrsr</code> structure is then used in a call to <code>SetCursor</code> or <code>SetCCursor</code> to change the cursor shape.

Your application is responsible for setting the initial appearance of the cursor and for changing the appearance of the cursor as appropriate for your application.

#### In Response to Mouse-Moved Events

For example, most applications set the cursor to the I-beam shape when the cursor is inside a textediting area of a document, and they change the cursor to an arrow when the cursor is inside the scroll bars. Your application can achieve this effect by requesting that the Event Manager report mouse-moved events if the user moves the cursor out of a region you specify in the <code>mouseRgn</code> parameter to the <code>WaitNextEvent</code> function. Then, when a mouse-moved event is detected in your main event loop, you can use <code>SetCursor</code> or <code>SetCCursor</code> to change the cursor to the appropriate shape.<sup>4</sup>

#### In Response to Resume Events

Your application also needs to adjust the cursor in response to resume events.

<sup>3</sup> If the operation takes longer than several seconds, you should display a dialog box with a progress indicator. (See Chapter 23 — Miscellany.)

<sup>&</sup>lt;sup>4</sup> Note that your application may also have to accommodate the cursor shape changing requirements of modeless dialog boxes with edit text field items.as well as its main windows.

#### **Hiding Cursors**

You can remove the cursor image from the screen using HideCursor. You can hide the cursor temporarily using ObscureCursor or you can hide the cursor in a given rectangle by using ShieldCursor. To display a hidden cursor, use ShowCursor. Note, however, that you do not need to explicitly show the cursor after your application uses ObscureCursor because the cursor automatically reappears when the user moves the mouse again.

# Creating an Animated Cursor

To create an animated cursor, you should:

- Create a series of 'CURS' resources that make up the "frames" of the animation. (Typically, an animated cursor uses four to seven frames.)
- Create an 'acur' resource. (The 'acur' resource collects and orders your 'CURS' frames into a single animation. It specifies the IDs of the resources and the sequence for displaying them in your animation.)
- Load the 'acur' resource into an application-defined structure which replicates the structure of an 'acur' resource, for example:

- Load the 'CURS' resources using GetCursor and assign handles to the resulting Cursor structures to the elements of the frame field.
- Call SetCursor to display each cursor, that is, each "frame", in rapid succession, returning to the first frame after the last frame has been displayed. This can be achieved by incrementing the frame at each null event (which means, of course, that the sleep parameter in the WaitNextEvent call must be set to the required interval between frame updates).

### Creating 'acur' and 'CURS' Resources

Fig 3 shows the structure of a compiled 'acur' resource, and an 'acur' resource and one of its associated 'CURS' resources being created using Resorcerer.

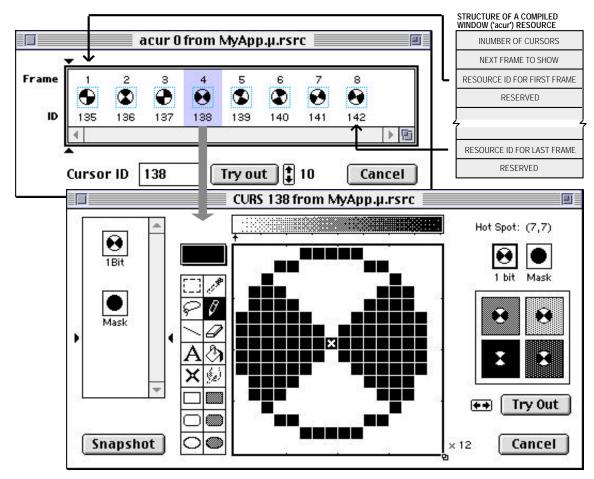


FIG 3 - CREATING AN 'acur' RESOURCE AND ASSOCIATED 'CURS' RESOURCES USING RESORCERER

#### **Icons**

#### Icons and the Finder

As stated at Chapter 9 — Finder Interface, the Finder uses **icons** to graphically represents objects, such as files and directories, on the desktop. Chapter 9 also introduced the subject of **icon families**, and stated that your application should provide the Finder with a family of specially designed icons for the application file itself and for each of the document types created by the application.

The provision of a family of icon types for each desktop object, rather than just one icon type, enables the Finder to automatically select the appropriate family member to display depending on the icon size specified by the user and the bit depth of the display device. Chapter 9 described the components of an icon family used by the Finder as follows:

Icon	Size (Pixels)	Resource in Which Defined
Large black-and-white icon, and mask	32 by 32	Icon list (' I CN#' ).
Small black-and-white icon, and mask	16 by 16	Small icon list (' i cs#')
Mini black-and-white icon, and mask	12 by 16	Mini icon list (' i cm#')
Large colour icon with 4 bits of colour data per pixel	32 by 32	Large 4-bit colour icon (' i cl 4' )
Small colour icon with 4 bits of colour data per pixel	16 by 16	Small 4-bit colour icon (' i cs4')
Mini colour icon with 4 bits of colour data per pixel	12 by 16	Mini 4-bit colour icon (' i cm4')
Large colour icon with 8 bits of colour data per pixel	32 by 32	Large 8-bit colour icon (' i cl 8' )
Small colour icon with 8 bits of colour data per pixel	16 by 16	Small 8-bit colour icon (' i cs8')
Mini colour icon with 8 bits of colour data per pixel	12 by 16	Mini 8-bit colour icon (' i cm8')

#### Creating Icon Family Resources Using Resorcerer

Fig 3 at Chapter 9 — Finder Interface shows icon families being created using Resorcerer.

# Other Icons — Icons, Colour Icons and Small Icons

Other icon types are the **icon**, **colour icon**, and **small icon**. Note that the Finder does not use or display these icon types.

#### Icon ('ICON')

The icon is defined in an 'I CON' resource, which contains a bit map for a 32-by-32 pixel black-and-white icon. Because it is always displayed on a white background, it does not need a mask.

#### Colour Icon ('cicn')

The colour icon is defined in a 'cicn' resource, which has a special format which includes a pixel map, a bit map, and a mask. You can use a 'cicn' resource to define a colour icon with a width and height between 8 and 256 pixels. You can also define the bit depth for a colour icon resource.

#### Small Icon ('SICN')

The small icon is defined in a 'SICN' resource. Small icons are 12 by 16 pixels even though they are stored in a resource as 16-by-16 pixel bitmaps. A 'SICN' resource consists of a list of 16-by-16 pixel bitmaps for black-and-white icons. $^5$ 

# Icons in Windows, Menus, and Alert and Dialog Boxes

The icons provided by your application for the Finder (or the default system-suppled icons used by the Finder if your application does not provide its own icons) are displayed on the desktop. Your application can also display icons in its menus, dialog boxes and windows.

#### Icons in Windows

You can display icons of any kind in your windows using the appropriate Icon Utilities functions.

#### Icons in Menus

The Menu Manager allows you to display icons of resource types 'ICON' (icon) 'cicn' (colour icon), and 'SICN' (small icon) in menu items. The procedure is as follows:

- Create the icon resource with a resource ID between 257 and 511. Subtract 256 from the resource ID to get a value called the **icon number**. Specify the icon number in the Icon field of the menu item definition.
- For an icon ('ICON'), specify 0x1D in the keyboard equivalent field of the menu item definition to indicate to the Menu Manager that the icon should be reduced to fit into a 16-by-16 pixel rectangle. Otherwise, specify a value of 0x00, or a value greater than 0x20, in the keyboard equivalent field to cause the Menu Manager to expand the item's rectangle so as to display the icon at its normal 32-by-32 pixel size. (A value greater than 0x20 in the keyboard equivalent field specifies the item's Command-key equivalent.)
- For a colour icon ('cicn'), specify 0x00 or a value greater than 0x20 in the keyboard equivalent field. The Menu Manager automatically enlarges the enclosing rectangle of the menu item according to the rectangle specified in the 'cicn' resource. (Colour icons, unlike icons, can be any height or width between 8 and 64 pixels.)

<sup>&</sup>lt;sup>5</sup> Typically, only the Finder and the Standard File Package use small icons.

• For a small icon ('SICN'), specify 0x1E in the keyboard equivalent field. This indicates that the item has an icon defined by a 'SICN' resource. The Menu Manager plots the icon in a 16-by-16 pixel rectangle.

The Menu Manager will then automatically display the icon whenever you display the menu using the MenuSelect function. The Menu Manager first looks for a 'cicn' resource with the resource ID calculated from the icon number and displays that icon if it is found. If a 'cicn' resource is not found and the keyboard equivalent field specifies 0x1E, the Menu Manager looks for a 'SICN' resource with the calculated resource ID. Otherwise, the Menu Manager searches for an 'ICON' resource and plots it in either a 32-by-32 pixel rectangle or a 16-by-16 bit rectangle, depending on the value in the menu item's keyboard equivalent field.6

#### Icons in Alert and Dialog Boxes

The Dialog Manager allows you to display icons of resource types 'I CON' (icon) and 'cicn' (colour icon) in alert and dialog boxes. You can display the icon alone or within an image well.

To display the icon alone, the procedure is to define an item of type I con and provide the resource ID of the icon in the item list ('DITL') resource for the dialog. This will cause the Dialog Manager to automatically display the icon whenever you display the alert or dialog box using Dialog Manager functions.

To display the icon within an image well, include an image well control in the alert or dialog box's item list and assign the resource ID of the icon to the control's minimum value field.

If you provide a colour icon (' ci cn') resource with the same resource ID as an icon (' ICON') resource, the Dialog Manager displays the colour icon instead of the black-and-white icon.

Ordinarily, you would use the Alert function (which does not automatically draw a system-supplied alert icon in the alert box), or the StandardAlert function with kAlertPlainAlert passed in the inAlertType parameter, when you wish to display an alert containing your own icon (for example, in your application's About... alert box). If you invoke an alert box using the NoteAlert, CautionAlert, or StopAlert functions, or with the StandardAlert function with an alert type constant of other than kAlertPlainAlert passed in the inAlertType parameter, the Dialog Manager draws the system-supplied black-and-white icon as well as your icon. Since your icon is drawn last, you can obscure the system-suppled icon by positioning your icon at the same coordinates.

# **Drawing and Manipulating Icons**

The Icon Utilities allow your application (and the system software) to draw and manipulate icons of any standard resource type in windows and, subject to the limitations and requirements previously described, in menus and dialog boxes.

You need to use Icon Utilities functions only if:

- You wish to draw icons in your application's windows.
- You wish to draw icons which are not recognised by the Menu Manager and the Dialog Manager in, respectively, menu items and dialog boxes.

# Preamble - Icon Families, Suites, and Caches

#### Icon Families

You can define individual icons of resource types 'ICON', 'cicn', and 'SICN' that are not part of an icon family and use Icon Utilities functions to draw them as required. However, to display an icon

 $<sup>^6</sup>$  Note that, for the Apple and Application menus, the Menu Manager either automatically reduces the icon to fit within the enclosing rectangle of the menu item or uses the appropriate icon from the application's icon family, such as the 'icl8' resource, if one is available.

effectively at a variety of sizes and bit depths, you should provide an icon family  $^7$  in the same way that you provide icon families for the Finder. The advantage of providing an icon family is that you can then leave it to functions such as PlotIconID, which are used to draw icons, to automatically determine which icon in the icon family is best suited to the specified destination rectangle and current display bit depth.

#### Icon Suites

Some Icon Utilities functions take as a parameter a handle to an **icon suite**. An icon suite typically consists of one or more handles to icon resources from a single icon family which have been read into memory. The GetIconSuite function may be used to get a handle to an icon suite, which can then be passed to functions such as PlotIconSuite to draw that icon in the icon suite best suited to the destination rectangle and current display bit depth. An icon suite can 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. For best results, an icon suite should always include a resource of type 'ICN#' in addition to any other large icons you provide and 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 functions.

#### 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 is associated with the icon suite. You can pass a handle to an icon cache to any of the Icon Utilities functions which 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 functions call your application's icon getter function to retrieve the data for that 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. PlotIconID determines, from the size of the specified destination rectangle and the current bit depth of the display device, which icon to draw. The icon drawn is as follows:

Destination Rectangle Size	Icon Drawn
Width or height greater than or equal to 32.	The 32-by-32 pixel icon with the appropriate bit depth.
Less than 32 by 32 pixels and greater than 16 pixels wide or 12 pixels high.	The 16-by-16 pixel icon with the appropriate bit depth.
Height less than or equal to 12 pixels or width less than or equal to 16 pixels.	The 12-by-16 pixel icon with the appropriate bit depth.

#### Icon Stretching and Shrinking

Depending on the size of the rectangle, PlotIconID may stretch or shrink the icon to fit. To draw icons without stretching them, PlotIconID requires that the destination rectangle have the same dimensions as one of the standard icons.

#### Icon Alignment and Transform

In addition to destination rectangle and resource ID parameters, PlotIconID takes alignment and transform parameters. Icon Utilities functions can automatically align an icon within its destination rectangle. (For example, an icon which is taller than it is wide can be aligned to either the right or left of its destination rectangle.) These functions can also transform the appearance of the icon in standard ways analogous to Finder states for icons.

<sup>&</sup>lt;sup>7</sup> Each icon in an icon family shares the same resource ID as other icons in the family but has its own resource type identifying the icon data it contains.

Variables of type IconAlignmentType and IconTransformType should be declared and assigned values representing alignment and transform requirements. Constants, such as kAlignAbsoluteCenter and kTransformNone, are available to specify alignment and transform requirements.

# Getting an Icon Suite and Drawing One of Its Icons

The <code>GetIconSuite</code> function, with the constant <code>kSelectorAllAvailableData</code> passed in the third parameter, is used to get all icons from an icon family with a specified resource ID and to collect the handles to the data for each icon into an icon suite. An icon from this suite may then be drawn using <code>PlotIconSuite</code> which, like <code>PlotIconID</code>, takes destination rectangle, alignment and transform parameters and stretches or shrinks the icon if necessary.

# Drawing Specific Icons From an Icon Family

If you need to plot a specific icon from an icon family rather than use the Icon Utilities to automatically select a family member, you must first create an icon suite which contains only the icon of the desired resource type together with its corresponding mask. Constants such as kSelectorLarge4Bit (an icon selector mask for an 'i cl4' icon) are used as the third parameter of the GetIconSuite call to retrieve the required family member. You can then use PlotIconSuite to plot the icon.

# Drawing Icons That Are Not Part of an Icon Family

To draw icons of resource type 'ICON' and 'cicn' in menu items and dialog boxes, and icons of resource type 'SICN' in menu items, you use Menu Manager and Dialog Manager functions such as SetItemIcon and SetDialogItem.

To draw resources of resource type <code>'ICON'</code> , <code>'cicn'</code> , and <code>'SICN'</code> in your application's windows, you use the following functions:

Resource Type	Function to Get Icon	Functions to Draw Icon
' I CON'	GetIcon	Pl ot I conHandl e
		PlotIcon
'cicn'	GetCIcon	Pl otCI conHandl e
		PlotCIcon
' SI CN'	GetResource	Pl otSI CNHandl e

The functions in this list ending in Handle allow you to specify alignment and transforms for the icon.

#### Manipulating Icons

The GetIconFromSuite function may be used to get a handle to the pixel data for a specific icon from an icon suite. You can then use this handle to manipulate the icon data, for example, to alter its colour or add three-dimensional shading.

The Icon Utilities also include functions which allow you to perform an action on one or more icons in an icon suite and to perform hit testing on icons.

# Main Constants, Data Types and Functions — Offscreen Graphics Worlds

#### Constants

#### Flags for GWorldFlags Parameter

```
pixPurgeBit = 0 Set to make base address for offscreen pixel image purgeable.

noNewDeviceBit = 1 Set to not create a new GDevice structure for offscreen world.

pixelsPurgeableBit = 6 Set to make base address for pixel image purgeable.
```

# Data Types

```
typedef CGrafPtr GWorldPtr;
typedef unsigned long GWorldFlags;
```

#### **Functions**

#### Creating, Altering, and Disposing of Offscreen Graphics Worlds

```
 \begin{array}{lll} QDErr & NewGWorld(GWorldPtr\ *offscreenGWorld, short\ PixelDepth, const\ Rect \\ *boundsRect, CTabHandle\ cTable, GDHandle\ aGDevice, GWorldFlags\ flags); \\ GWorldFlags & UpdateGWorld(GWorldPtr\ *offscreenGWorld, short\ pixelDepth, const\ Rect \\ *boundsRect, CTabHandle\ cTable, GDHandle\ aGDevice, GWorldFlags\ flags); \\ void & DisposeGWorld(GWorldPtr\ offscreenGWorld); \\ \end{array}
```

#### Saving and Restoring Graphics Ports and Offscreen Graphics Worlds

```
void GetGWorld(CGrafPtr *port, GDHandle *gdh);
void SetGWorld(CGrafPtr port, GDHandle gdh);
```

#### Managing an Offscreen Graphics World's Pixel Image

```
PixMapHandle GetGWorldPixMap(GWorldPtr offscreenGWorld);
              LockPixels(PixMapHandle pm);
Bool ean
voi d
              UnlockPixels(PixMapHandle pm);
              AllowPurgePixels(PixMapHandle pm);
voi d
              NoPurgePixels(PixMapHandle pm);
voi d
GWorl dFl ags
              GetPi xelsState(Pi xMapHandle pm);
              SetPixelsState(PixMapHandle pm, GWorldFlags state);
voi d
              GetPixBaseAddr(PixMapHandle pm);
Ptr
Bool ean
              PixMap32Bit(PixMapHandle pmHandle);
```

# Main Constants, Data Types and Functions — Pictures

#### **Constants**

#### Verbs for the GetPictInfo, GetPixMapInfo, and NewPictInfo calls

#### Colour Pick Methods for the GetPictInfo, GetPixMapInfo, and NewPictInfo calls

```
\begin{array}{lll} \text{systemMethod} &=& 0 & \text{System color pick method.} \\ \text{popularMethod} &=& 1 & \text{Most popular set of colors.} \\ \text{medianMethod} &=& 2 & \text{A good average mix of colors.} \end{array}
```

### Data Types

#### **Picture**

#### **OpenCPicParams**

```
struct OpenCPicParams
                       \label{eq:conding} \ensuremath{\textit{//}}\ \ensuremath{\textit{Optimal}}\ \ bounding\ \ rectangle.
  Rect
          srcRect:
  Fi xed
          hRes:
                        // Best horizontal resolution.
  Fi xed
                        // Best vertical resolution.
          vRes:
                       // Set to -2
  short
          version;
          reserved1;
                       // (Reserved.
  short
          reserved2;
                       // (Reserved.
                                        Set to 0.)
  long
typedef struct OpenCPicParams OpenCPicParams;
PictInfo
struct PictInfo
                      version;
                                        // This is always zero, for now.
  short
                                        // Number of actual colors in the picture(s)/pixmap(s).
                      uni queColors;
  long
  Pal etteHandl e
                      thePalette;
                                        // Handle to the palette information.
  CTabHandl e
                      theColorTable;
                                        // Handle to the color table.
                                        // Maximum horizontal resolution for all the pixmaps.
  Fi xed
                      hRes:
  Fi xed
                      vRes;
                                        // Maximum vertical resolution for all the pixmaps.
  short
                      depth;
                                        // Maximum depth for all the pixmaps (in the picture).
                                        \ensuremath{//} Picture frame rectangle (contains the entire picture).
  Rect
                      sourceRect;
  long
                      textCount;
                                        // Total number of text strings in the picture.
  long
                      lineCount;
                                        // Total number of lines in the picture.
                                        // Total number of rectangles in the picture.
                      rectCount:
  long
  long
                      rRectCount;
                                        // Total number of round rectangles in the picture.
                                        // Total number of ovals in the picture.
  long
                      oval Count;
                                        // Total number of arcs in the picture.
                      arcCount:
  long
                                        // Total number of polygons in the picture.
  long
                      pol yCount;
                      regionCount:
                                        // Total number of regions in the picture.
  long
                                        // Total number of bitmaps in the picture.
                      bitMapCount;
  long
  long
                      pi xMapCount;
                                        \ensuremath{//} Total number of pixmaps in the picture.
  long
                      commentCount;
                                        // Total number of comments in the picture.
                                       // The number of unique comments in the picture.
                      uniqueComments;
  long
  Comment SpecHandle
                      commentHandle;
                                        // Handle to all the comment information.
  long
                      uni queFonts;
                                        // The number of unique fonts in the picture.
                                        // Handle to the FontSpec information.
  FontSpecHandle
                      fontHandle:
  Handl e
                      fontNamesHandle; // Handle to the font names.
  long
                      reserved1;
                      reserved2:
  long
typedef struct PictInfo PictInfo;
typedef PictInfo *PictInfoPtr;
typedef PictInfoPtr *PictInfoHandle;
CommentSpec
struct CommentSpec
  short
          count:
                        // Number of occurrences of this comment ID.
                        // ID for the comment in the picture.
  short
          ID:
typedef struct CommentSpec CommentSpec;
typedef CommentSpec *CommentSpecPtr;
typedef CommentSpecPtr *CommentSpecHandle;
FontSpec
struct FontSpec
          pictFontID; // ID of the font in the picture.
  short
                       // ID of the same font in the current system file.
  short
          sysFontID;
                       // Bit array of all the sizes found (1. 127) (bit 0 means > 127).
  long
          size[4];
                        // Combined style of all occurrances of the font.
  short
          style;
          nameOffset; // Offset into the fontNamesHdl handle for the font's name.
  long
}:
typedef struct FontSpec FontSpec;
typedef FontSpec *FontSpecPtr;
typedef FontSpecPtr *FontSpecHandle;
```

#### Creating and Disposing of Pictures

```
PicHandle OpenCPicture(const OpenCPicParams *newHeader);
           PicComment(short kind, short dataSize, Handle dataHandle);
voi d
voi d
           ClosePicture(void);
voi d
           KillPicture(PicHandle myPicture);
Drawing Pictures
           DrawPicture(PicHandle myPicture, const Rect *dstRect)
PicHandle GetPicture(Integer picID);
Collecting Picture Information
           GetPictInfo(PicHandle thePictHandle, PictInfo *thePictInfo, short verb, short
0SErr
           colors {\tt Requested}, short \ color {\tt PickMethod}, short \ version);\\
           GetPixMapInfo(PixMapHandle thePixMapHandle, PictInfo *thePictInfo, short verb, short
0SErr
           colorsRequested, short colorPickMethod, short version);
0SErr
           NewPictInfo(PictInfoID *thePictInfoID, short verb, short colorsRequested, short
           colorPickMethod, short version);
0SErr
           RecordPictInfo(PictInfoID thePictInfoID, PicHandle thePictHandle);
0SErr
           RecordPixMapInfo(PictInfoID thePictInfoID, PixMapHandle thePixMapHandle);
0SErr
           RetrievePictInfo(PictInfoID thePictInfoID, PictInfo *thePictInfo, short
           colorsRequested):
0SErr
           DisposPictInfo(PictInfoID thePictInfoID);
```

# Main Constants, Data Types and Functions — Cursors

#### **Constants**

### Data Types

struct Cursor

#### Cursor

```
Bits16
                data;
  Bits16
                mask:
  Poi nt
                hotSpot;
typedef struct Cursor Cursor;
typedef Cursor *CursPtr;
typedef CursPtr *CursHandle;
CCrsr
struct CCrsr
  short
                crsrType;
                               // Type of cursor.
  PixMapHandle crsrMap;
                               // The cursor's pixmap.
  Handl e
                crsrData;
                               // Cursor's data.
  Handl e
                crsrXData;
                               // Expanded cursor data.
                               // Depth of expanded data (0 if none).
                crsrXValid;
  short
  Handl e
                crsrXHandle;
                               // Future use.
  Bits16
                crsr1Data;
                               // One-bit cursor.
                               // Cursor's mask.
                crsrMask;
  Bits16
                               // Cursor's hotspot.
                crsrHotSpot;
  Poi nt
                crsrXTable;
                               // Private.
  long
                               // Private.
  long
                crsrID;
typedef struct CCrsr CCrsr;
typedef CCrsr *CCrsrPtr;
typedef CCrsrPtr *CCrsrHandle;
```

#### Acur

```
struct Acur
                              // Number of cursors (frames).
  short
  short
                index;
                              // (Reserved.)
               frame1;
                              // 'CURS' resource ID for frame #1.
  short
                              // (Recerved.)
  short
               fill1;
               frame2;
                              // 'CURS' resource ID for frame #2.
  short
               fill2;
                              // (Reserved.)
  short
                              // 'CURS' resource ID for frame #n.
  short
               frameN:
               fillN;
                              // (Reserved.)
  short
typedef struct Acur acur, *acurPtr, **acurHandle;
```

#### **Functions**

#### **Initialising Cursors**

#### Changing Black-and-White Cursors

```
CursHandle GetCursor(short cursorID);
void SetCursor(const Cursor *crsr);
```

#### **Changing Colour Cursors**

```
CCrsrHandle GetCCursor(short crsrID);
void SetCCursor(CCrsrHandle cCrsr);
void AllocCursor(void)
void DisposCCursor(CCrsrHandle cCrsr);
void DisposeCCursor(CCrsrHandle cCrsr);
```

#### Hiding, Showing, and Animating Cursors

# Main Constants, Data Types and Functions — Icons

#### **Constants**

#### Types for Icon Families

```
      kLarge1BitMask
      = FOUR_CHAR_CODE('ICN#')

      kLarge4BitData
      = FOUR_CHAR_CODE('icl4')

      kLarge8BitData
      = FOUR_CHAR_CODE('icl8')

      kSmall1BitMask
      = FOUR_CHAR_CODE('ics#')

      kSmall4BitData
      = FOUR_CHAR_CODE('ics4')

      kSmil1BitData
      = FOUR_CHAR_CODE('ics#')

      kMini1BitMask
      = FOUR_CHAR_CODE('icm#')

      kMini4BitData
      = FOUR_CHAR_CODE('icm#')

      kMini8BitData
      = FOUR_CHAR_CODE('icm8')
```

#### IconAlignmentType Values

```
kAl i gnNone
                            = 0x00
kAlignVertical Center
                            = 0x01
kAl i gnTop
                            = 0x02
kAl i gnBottom
                            = 0x03
kAlignHorizontalCenter
                            = 0x04
kAlignAbsoluteCenter
                            = kAlignVerticalCenter | kAlignHorizontalCenter
                                                      kAlignHorizontalCenter
kAl i gnCenterTop
                            = kAlignTop
kAlignCenterBottom
                            = kAlignBottom
                                                    kAlignHorizontalCenter
```

```
kAlignLeft
                              = 0x08
kAlignCenterLeft
                              = kAlignVerticalCenter
                                                          kAl i gnLeft
kAl i gnTopLeft
                              = kAl i gnTop
                                                          kAlignLeft
kAlignBottomLeft
                              = kAlignBottom
                                                          kAlignLeft
kAl i gnRi ght
                              = 0x0C
kAlignCenterRight
                              = kAlignVerticalCenter |
                                                          kAl i gnRi ght
kAl i gnTopRi ght
                              = kAl i gnTop
                                                          kAl i gnRi ght
kAl i gnBottomRi ght
                              = kAlignBottom
                                                          kAl i gnRi ght
```

#### IconTransformType Values

```
kTransformNone
                            = 0x00
kTransformDi sabl ed
                            = 0x01
kTransformOffline
                            = 0x02
kTransformOpen
                            = 0x03
kTransformLabel 1
                            = 0x0100
kTransformLabel 2
                            = 0x0200
kTransformLabel 3
                            = 0x0300
kTransformLabel 4
                            = 0x0400
kTransformLabel 5
                            = 0x0500
kTransformLabel6
                            = 0x0600
kTransformLabel7
                            = 0x0700
kTransformSel ected
                           = 0x4000
kTransformSelectedDisabled = kTransformSelected |
                                                    kTransformDi sabl ed
kTransformSelectedOffline = kTransformSelected
                                                    kTransformOffline
kTransformSelectedOpen
                            = kTransformSelected | kTransformOpen
```

#### IconSelectorValue Masks

```
= 0x00000001
kSelectorLarge1Bit
kSelectorLarge4Bit
                            = 0x00000002
                            = 0x00000004
kSelectorLarge8Bit
kSelectorSmall1Bit
                            = 0x00000100
kSelectorSmall4Bit
                            = 0x00000200
kSelectorSmall8Bit
                            = 0 \times 00000400
kSel ectorMi ni 1Bi t
                            = 0x00010000
kSelectorMi ni 4Bi t
                            = 0x00020000
kSelectorMi ni 8Bi t
                            = 0 \times 00040000
kSelectorAllLargeData
                            = 0x000000FF
kSelectorAllSmallData
                            = 0x0000FF00
kSelectorAllMiniData
                            = 0x00FF0000
kSelectorAll1BitData
                            = kSelectorLarge1Bit |
                                                     kSelectorSmall1Bit |
                                                                           kSel ectorMi ni 1Bi t
kSelectorAll4BitData
                            = kSelectorLarge4Bit
                                                     kSelectorSmall4Bit
                                                                           kSel ectorMi ni 4Bi t
kSelectorAll8BitData
                            = kSelectorLarge8Bit |
                                                     kSelectorSmall8Bit | kSelectorMini8Bit
kSelectorAllAvailableData = (long) 0xFFFFFFFF
```

# Data Types

```
typedef short
                 I conAl i gnmentType;
typedef short
                 IconTransformType;
typedef UInt32
                 I conSel ectorValue;
typedef Handle
                 I conSuiteRef:
typedef Handle
                 IconCacheRef;
Clcon
struct CIcon
  Pi xMap
           i conPMap;
                              // Icon's pixMap.
  BitMap
                              // Icon's mask.
           i conMask;
  BitMap
           i conBMap;
                              // Icon's bitMap.
  Handl e
           iconData;
                              // Icon's data.
           iconMaskData[1]; // Icon's mask and BitMap data.
  short
typedef struct CIcon CIcon;
typedef CIcon *CIconPtr;
typedef CIconPtr *CIconHandle;
```

#### **Functions**

#### **Drawing Icons From Resources**

 $0 S Err & Plot I con I \ D (const Rect *the Rect, I con Alignment Type \ align, I con Transform Type \ transform, \\ short \ the Res I \ D);$ 

void PlotIcon(const Rect \*theRect, Handle theIcon);

OSErr PlotIconHandle(const Rect \*theRect,IconAlignmentType align,

IconTransformType transform, Handle theIcon);

void PlotCIcon(const Rect \*theRect, CIconHandle theIcon);

OSErr PlotCIconHandle(const Rect \*theRect, IconAlignmentType align,

IconTransformType transform, CI conHandle theIcon);

OSErr PlotSICNHandle(const Rect \*theRect,IconAlignmentType align,

IconTransformType transform, Handle theSICN);

#### Getting Icons From Resources Which do Not Belong to an Icon Family

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

#### Disposing of Icons

OSErr DisposeClcon(ClconHandle thelcon);

#### Creating an Icon Suite

OSErr GetIconSuite(Handle \*theIconSuite, short theResID, IconSelectorValue selector);;

OSErr NewI conSuite(Handle \*theI conSuite);

OSErr AddI conToSuite(Handle theI conData, Handle theSuite, ResType theType);

#### Getting Icons From an Icon Suite

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

#### Drawing Icons From an Icon Suite

OSErr PlotIconSuite(const Rect \*theRect, IconAlignmentType align,

IconTransformType transform, Handle theIconSuite);

#### Performing Operations on Icons in an Icon Suite

OSErr ForEachIconDo(handle theSuite, IconSelectorValue selector, IconActionUPP action,

void \*yourDataPtr);

#### Disposing of Icon Suites

OSErr DisposeIconSuite(Handle theIconSuite, Boolean disposeData);

#### Converting an Icon Mask to a Region

 $OSErr \qquad I conSuiteToRgn(RgnHandle\ theRgn, const\ Rect\ *iconRect,$ 

IconAlignmentType align, Handle theIconSuite);

OSErr IconIDToRegion(RgnHandle theRgn, const Rect \*iconRect,

IconAlignmentType align, short iconID);

#### Determining Whether a Point or Rectangle is Within an Icon

Boolean PtInIconSuite(Point testPt, const Rect \*iconRect, IconAlignmentType align,

Handle the I con Suite);

Boolean PtInIconID(Point testPt, const Rect \*iconRect, IconAlignmentType align,

short iconID);

Boolean RectInIconSuite(const Rect \*testRect, const Rect \*iconRect, IconAlignmentType align,

Handle theIconSuite);

Boolean RectInIconID(const Rect \*testRect, const Rect \*iconRect, IconAlignmentType align,

short i conID);

#### Working With Icon Caches

OSErr MakeIconCache(Handle \*theHandle,IconGetterProcPtr makeIcon,void \*yourDataPtr);

 $OSErr \\ LoadIconCache (const \ Rect \ *theRect, IconAlignmentType \ align, \\$ 

IconTransformType transform, Handle theIconCache);

# Demonstration Program

```
// GWorl dPi cCursI con. c
// **
//
// This program:
11
// • Opens a window in which the results of various drawing, copying, and cursor shape
//
     change operations are displayed.
//
     Demonstrates offscreen graphics world, picture, cursor, cursor shape change,
// •
11
     animated cursor, and icon operations as a result of the user choosing items from
//
     a Demonstration menu.
11
// • Demonstrates a modal dialog-based About... box containing a picture.
11
// To keep the non-demonstration code to a minimum, the program contains no functions
// for updating the window or for responding to activate and operating system events.
// The program utilises the following resources:
//
// •
     An 'MBAR' resource and associated 'MENU' resources (preload, non-purgeable).
// • A 'WIND' resource (purgeable) (initially visible).
// • An 'acur' resource (purgeable).
// •
    'CURS' resources associated with the 'acur' resource (preload, purgeable).
11
     Two 'cicn' resources (purgeable), one for the Icons menu item and one for drawing
//
     in the window.
// • Two icon family resources (purgeable), both for drawing in the window.
//
     A 'DLOG' resource (purgeable) and an associated 'DITL' resource (purgeable) and
// •
//
      'PICT' resource for an About GWorldPicCursIcon... dialog box.
//
// • A 'STR#' resource (purgeable) containing transform constants.
11
// • A 'SIZE' resource with the acceptSuspendResumeEvents and is32BitCompatible flags
11
#include <Appearance.h>
#include <CursorCtl.h>
#include <Devices.h>
#include <Fonts.h>
#include <Menus.h>
#include <PictUtils.h>
#include <Processes.h>
#include <Sound. h>
#include <TextUtils.h>
#include <ToolUtils.h>
#include <Resources.h>
                                                                    ..... defines
#define rMenubar
                               128
#define rWindow
                               128
                               128
#define mApple
#define iAbout
                               1
#define mFile
                               129
#define iQuit
                               11
#define mDemonstration
                               131
#define iOffScreenGWorld1
                               1
#define iOffScreenGWorld2
                               2
#define iPicture
                               3
#define iCursor
                               4
#define iAnimatedCursor
                               5
#define ilcon
                               6
#define rBeachBallCursor
                               128
#define rPicture
                               128
```

```
#define rTransformStrings
                                 128
#define rIconFamily1
                                 128
#define rIconFamily2
                                 129
#define rColourIcon
                                 128
#define rAboutDialog
                                 128
#define kSleepTime
#define kBeachBallTickInterval 5
                             0x7FFFFFFF
#define MAXLONG
                           (((Point *) &(r))[0])
(((Point *) &(r))[1])
#define topLeft(r)
#define botRight(r)
                                                                  ..... typedefs
typedef struct
  SInt16
              numberOfFrames;
  SInt 16
              whichFrame:
  CursHandle frame[];
} animCurs, *animCursPtr, **animCursHandle;
                                                    .....global variables
                 gPreAllocatedBlockPtr;
Ptr
WindowPtr
                 gWindowPtr;
Bool ean
                gDone;
                 gInBackground;
Bool ean
SI nt 32
                 gSleepTime;
          gCursorRegion;
gCursorRegionsActive = false;
gAnimCursActive = false;
RgnHandl e
Bool ean
Bool ean
                gAni mCursResourceHdl = NULL;
Handl e
ani mCursHandle gAni mCursHdl;
                 gAni mCursTi ckI nterval;
SI nt 16
SInt32
                gAni mCursLastTi ck;
               gBl ackCol our = { 0x0000, 0x0000, 0x0000 };
gWhiteCol our = { 0xFFFF, 0xFFFF, 0xFFFF };
gBeigeCol our = { 0xF000, 0xE300, 0xC200 };
gBl ueCol our = { 0x4444, 0x4444, 0x9999 };
RGBCol or
RGBCol or
RGBColor
RGBCol or
// ______function prototypes
voi d
                             (voi d);
        main
        doInitManagers (void);
eventLoop (void);
voi d
voi d
voi d
        doEvents
                             (EventRecord *);
        doMenuChoi ce
                             (SInt 32);
voi d
        doOffScreenGWorld1 (void);
doOffScreenGWorld2 (void);
voi d
voi d
void doChangeCursor void doAnimCursor (void);

Boolean doGetAnimCursor void doSpinAnimCursor (SInt16 STate)
        doSpi nAni mCursor
voi d
        doReleaseAnimCursor (void);
        doI dl e
                             (void);
voi d
voi d
        doIcon
                             (void):
voi d
        doAboutDi ai og
doGWorl dDrawi ng
        doAboutDi al og
                           (voi d);
voi d
                             (void):
UInt16 doRandomNumber
                             (UI nt 16, UI nt 16);
void main(void)
{
              menubarHdl;
  Handle
  MenuHandle menuHdl;
             .....get nonrelocatable block low in heap for About... dialog structure
  if(!(gPreAllocatedBlockPtr = NewPtr(sizeof(DialogRecord))))
    Exi tToShel l();
  // ______initialise managers
  doInitManagers();
```

```
// ..... cause the Appearance-compliant menu bar definition function to be called directly
 Regi sterAppearanceCl i ent();
 // ______see random number generator
 GetDateTime((UInt32 *) (&qd.randSeed));
                                     ..... set up menu bar and menus
 if(!(menubarHdl = GetNewMBar(rMenubar)))
  ExitToShell();
 SetMenuBar(menubarHdl);
 DrawMenuBar();
 if(!(menuHdl = GetMenuHandle(mApple)))
  ExitToShell();
  AppendResMenu(menuHdl, 'DRVR');
 // _____open window
 if(!(gWindowPtr = GetNewCWindow(rWindow, NULL, (WindowPtr)-1)))
  ExitToShell();
 SetPort(gWindowPtr);
 TextSi ze(10);
 // _____enter event loop
 eventLoop();
void doInitManagers(void)
 MaxAppl Zone();
 MoreMasters();
 InitGraf(&qd.thePort);
 InitFonts():
 InitWindows():
 InitMenus():
 TEI nit();
 I ni t Di al ogs(NULL);
 InitCursor();
 FlushEvents(everyEvent, 0);
void eventLoop(void)
 EventRecord eventStructure;
 Bool ean
         gotEvent;
 gDone = false;
 gSleepTime = MAXLONG;
 gCursorRegion = NULL;
 while(!gDone)
  gotEvent = WaitNextEvent(everyEvent, &eventStructure, gSl eepTi me, gCursorRegi on);
  if(gotEvent)
    doEvents(&eventStructure);
  else
    doIdle();
  void doEvents(EventRecord *eventStrucPtr)
```

```
WindowPtr windowPtr;
  SInt16
           partCode;
  SInt8
           charCode;
  switch(eventStrucPtr->what)
    case mouseDown:
     partCode = Fi ndWi ndow(eventStrucPtr->where, &wi ndowPtr);
      switch(partCode)
        case inMenuBar:
         doMenuChoi ce(MenuSel ect(eventStrucPtr->where));
         break:
        case inContent:
         if(windowPtr != FrontWindow())
           Sel ectWi ndow(wi ndowPtr);
         break:
        case inDrag:
         DragWi ndow(wi ndowPtr, eventStrucPtr->where, &qd. screenBi ts. bounds);
         if(gCursorRegionsActive)
           doChangeCursor(windowPtr, gCursorRegion);
         break;
      break;
    case keyDown:
    case autoKey:
      charCode = eventStrucPtr->message & charCodeMask;
      if((eventStrucPtr->modifiers & cmdKey) != 0)
        doMenuChoi ce(MenuEvent(eventStrucPtr));
     break;
    case updateEvt:
      BeginUpdate((WindowPtr) eventStrucPtr->message);
      EndUpdate((WindowPtr) eventStrucPtr->message);
     break;
    case osEvt:
      switch((eventStrucPtr->message >> 24) \& 0x000000FF)
        case suspendResumeMessage:
         if((eventStrucPtr->message & resumeFlag) == 1)
           gInBackground = false;
           SetCursor(&qd. arrow);
         }
         else
           gInBackground = true;
         break;
        case mouseMovedMessage:
         doChangeCursor(FrontWindow(), gCursorRegion);
         break;
     break:
  }
                    void doMenuChoice(SInt32 menuChoice)
  SInt16 menuID, menuItem;
  Str255 itemName;
  SInt16 daDriverRefNum;
  menuID = HiWord(menuChoice);
  menuItem = LoWord(menuChoice);
  if(menuID == 0)
   return;
  if(gAnimCursActive == true)
```

```
gAnimCursActive = false;
    SetCursor(&qd. arrow);
   doRel easeAni mCursor();
   gSleepTime = MAXLONG;
  if(gCursorRegionsActive == true)
    gCursorRegionsActive = false;
    Di sposeRgn(gCursorRegion);
   gCursorRegion = NULL;
  switch(menuID)
   case mApple:
      if(menuItem == iAbout)
        doAboutDialog();
      else
        GetMenuItemText(GetMenuHandle(mApple), menuItem, itemName);
        daDriverRefNum = OpenDeskAcc(itemName);
     break;
   case mFile:
      if(menuItem == iQuit)
        gDone = true;
      break;
    case mDemonstration:
     switch(menuItem)
        case iOffScreenGWorld1:
          doOffScreenGWorld1();
          break;
        case iOffScreenGWorld2:
          doOffScreenGWorld2();
          break:
        case iPicture:
         doPi cture();
          break;
        case i Cursor:
          doCursor();
          break:
        case iAnimatedCursor:
          doAnimCursor();
          break:
        case iIcon:
          doI con();
          break;
      break;
 }
  HiliteMenu(0);
                  ********** doOffScreenGWorld1
void doOffScreenGWorld1(void)
                wi ndowPortPtr;
  CGrafPtr
  GDHandl e
                devi ceHdl;
  QDErr
                qdErr;
                gworl dPortPtr;
  GWorl dPtr
                gworl dPi xMapHdl;
 Pi xMapHandl e
  Bool ean
                lockPixResult;
                sourceRect, destRect;
  Rect
```

```
RGBBackColor(&gBlueColour);
EraseRect(&gWindowPtr->portRect);
SetWTitle(gWindowPtr, "\pSimulated time-consuming drawing operation");
doGWorl dDrawi ng();
SetWTitle(gWindowPtr, "\pClick mouse to repeat in offscreen graphics port");
while(!Button()) ;
RGBBackColor(&gBlueColour);
EraseRect(&gWindowPtr->portRect);
RGBForeColor(&gWhiteColour);
MoveTo(190, 180);
DrawString("\pPlease Wait. Drawing in offscreen graphics port.");
                   .....draw in offscreen graphics port and copy to window
SetCursor(*(GetCursor(watchCursor)));
// ...... save current graphics world and create offscreen graphics world
GetGWorld(&windowPortPtr, &deviceHdl);
 qdErr \ = \ NewGWorl \ d(\&gworl \ dPortPtr, \ 0, \&gWindowPtr->portRect, \ NULL, \ NULL, \ 0) \ ; 
if(gworldPortPtr == NULL || qdErr != noErr)
 SysBeep(10);
 return;
SetGWorld(gworldPortPtr, NULL);
// ..... lock pixel image for duration of drawing and erase offscreen to white
gworl dPi xMapHdl = GetGWorl dPi xMap(gworl dPortPtr);
if(!(lockPixResult = LockPixels(gworldPixMapHdl)))
 SysBeep(10);
 return:
EraseRect(&(gworldPortPtr->portRect));
// ...... draw into the offscreen graphics port
doGWorl dDrawi ng();
// ..... restore saved graphics world
SetGWorld(windowPortPtr, deviceHdl);
// ..... set source and destination rectangles
sourceRect = gworldPortPtr->portRect;
destRect = windowPortPtr->portRect;
// ..... ensure background colour is white and foreground colour in black, then copy
RGBBackColor(&gWhiteColour);
RGBForeColor(&gBlackColour);
{\tt CopyBits(\&((GrafPtr)\ gworldPortPtr)->portBits},
        &((GrafPtr) windowPortPtr)->portBits,
        &sourceRect, &destRect, srcCopy, NULL);
if(QDError() != noErr)
 SysBeep(10);
// ..... clean up
UnlockPixels(gworldPixMapHdl);
Di sposeGWorl d(gworl dPortPtr);
```

```
SetCursor(&qd. arrow);
 SetWTitle((WindowPtr) windowPortPtr,
            \pOffscreen Graphics Worlds, Pictures, Cursors and Icons");
}
// ********* do0ffScreenGWorld2
void doOffScreenGWorld2(void)
 Pi cHandl e
               picture1Hdl, picture2Hdl;
               sourceRect, maskRect, maskDisplayRect, dest1Rect, dest2Rect, destRect;
 Rect
 CGrafPtr
               windowPortPtr:
 GDHandl e
               devi ceHdl;
  QDErr
               qdErr;
 GWorl dPtr
               gworl dPortPtr;
  PixMapHandle gworldPixMapHdl;
 RgnHandl e
               region1Hdl, region2Hdl, regionHdl;
 SInt 16
               a, sourceMode;
 RGBBackColor(&gBeigeColour);
 EraseRect(&gWindowPtr->portRect);
  // .....get the source picture and draw it in the window
  if(!(picture1Hdl = GetPicture(rPicture)))
   ExitToShell();
 HNoPurge((Handle) picture1Hdl);
  SetRect(&sourceRect, 116, 35, 273, 147);
  DrawPi cture(pi cture1Hdl, &sourceRect);
 HPurge((Handle) picture1Hdl);
  MoveTo(116, 32);
 DrawString("\pSource image");
  // .....save current graphics world and create offscreen graphics world
 GetGWorld(&windowPortPtr, &deviceHdl);
 SetRect(&maskRect, 0, 0, 157, 112);
  qdErr = NewGWorld(&gworldPortPtr, 0, &maskRect, NULL, NULL, 0);
 if(gworldPortPtr == NULL || qdErr != noErr)
   SysBeep(10);
   return;
 }
 SetGWorld(gworldPortPtr, NULL);
  // ......lock pixel image for duration of drawing and erase offscreen to white
 gworl dPi xMapHdl = GetGWorl dPi xMap(gworl dPortPtr);
 if(!(LockPixels(gworldPixMapHdl)))
   SysBeep(10);
   return:
 }
  EraseRect(&gworl dPortPtr->portRect);
  // .....get mask picture and draw it in offscreen graphics port
  if(!(picture2Hdl = GetPicture(rPicture + 1)))
   ExitToShell();
  HNoPurge((Handle) picture2Hdl);
 DrawPi cture(pi cture2Hdl, &maskRect);
                                                 .....also draw it in the window
  SetGWorld(windowPortPtr, deviceHdl);
 SetRect (&maskDi spl ayRect, 329, 35, 485, 146);
  DrawPi cture(pi cture2Hdl, &maskDi spl ayRect);
 HPurge((Handle) picture2Hdl);
  MoveTo(329, 32);
  DrawString("\pCopy of offscreen mask");
```

```
// .....define an oval-shaped region and a round rectangle-shaped region
  SetRect(&dest1Rect, 22, 171, 296, 366);
  region1Hdl = NewRgn();
  OpenRgn();
  FrameOval (&dest1Rect);
  CloseRgn(region1Hdl);
  SetRect(&dest2Rect, 308, 171, 582, 366);
  region2Hdl = NewRgn();
  OpenRgn();
  FrameRoundRect(&dest2Rect, 100, 100);
  CloseRgn(region2Hdl);
  SetWTitle((WindowPtr) windowPortPtr, "\pClick mouse to copy");
  while(!Button()) ;
  // ...... set background and foreground colour, then copy source to destination using mask
  RGBForeColor(&gBlackColour);
  RGBBackColor(&gWhiteColour);
  for (a=0; a<2; a++)
    if(a == 0)
    {
      regionHdl = region1Hdl;
      destRect = dest1Rect;
      sourceMode = srcCopy;
      MoveTo(22, 168);
      DrawString("\pBoolean source mode srcCopy");
    else
    {
      regionHdl = region2Hdl;
      destRect = dest2Rect;
      sourceMode = srcXor;
      MoveTo(308, 168);
      DrawString("\pBoolean source mode srcXor");
   }
    CopyDeepMask(&((GrafPtr) windowPortPtr)->portBits,
                 &((GrafPtr) gworldPortPtr)->portBits,
                 &((GrafPtr) windowPortPtr)->portBits,
                 \& source Rect, \& mask Rect, \& dest Rect, source Mode + dither Copy, region Hdl);\\
    if(QDError() != noErr)
      SysBeep(10);
                                                                   ..... clean up
  UnlockPixels(gworldPixMapHdl);
  Di sposeGWorl d(gworl dPortPtr);
  ReleaseResource((Handle) picture1Hdl);
  ReleaseResource((Handle) picture2Hdl);
  Di sposeRgn(regi on1Hdl);
  Di sposeRgn(regi on2Hdl);
  SetWTitle((WindowPtr) windowPortPtr,
            "\pOffscreen Graphics Worlds, Pictures, Cursors and Icons");
// ******************** doPicture
void doPicture(void)
{
                  pictureRect, theRect;
  Rect
  OpenCPicParams picParams;
  RgnHandl e
                  ol dCl i pRgn;
  Pi cHandle
                  pictureHdl;
  SInt16
                 a, left, top, right, bottom, random;
                 theColour;
  RGBCol or
  0SErr
                  osErr;
  PictInfo
                  pictInfo;
```

}

```
Str255
               theString;
RGBBackColor(&gWhiteColour);
EraseRect(&gWindowPtr->portRect);
                       _____define picture rectangle
pictureRect = gWindowPtr->portRect;
pictureRect.right = (gWindowPtr->portRect.right - gWindowPtr->portRect.left) / 2;
InsetRect(&pictureRect, 10, 10);
// _____set clipping region
oldClipRgn = NewRgn();
GetClip(oldClipRgn);
ClipRect(&pictureRect);
                           _____set up OpenCPicParams structure
picParams.srcRect = pictureRect;
pi cParams. hRes = 0x00480000;
pi cParams. vRes = 0x00480000;
picParams. version = -2;
                                                            .....record picture
pictureHdl = OpenCPicture(&picParams);
RGBBackColor(&gBlueColour);
EraseRect(&pictureRect);
for (a=0; a<300; a++)
 theRect = pictureRect;
 theColour.red
                = doRandomNumber(0, 65535);
 theColour.green = doRandomNumber(0,65535);
  theColour. blue = doRandomNumber(0, 65535);
 RGBForeColor(&theColour);
 left = doRandomNumber(10, (UInt16) theRect.right + 1);
 top = doRandomNumber(10, (UInt16) theRect.bottom + 1);
right = doRandomNumber((UInt16) left, (UInt16) theRect.right + 1);
 bottom = doRandomNumber((UInt16) top, (UInt16) theRect.bottom + 1);
 SetRect(&theRect,left,top,right,bottom);
 PenMode(doRandomNumber(add0ver, adMi n + 1));
 random = doRandomNumber(0,6);
 if(random == 0)
   MoveTo(left, top);
   LineTo(right - 1, bottom - 1);
 else if(random == 1)
   PaintRect(&theRect);
  else if(random == 2)
   PaintRoundRect(&theRect, 30, 30);
 else if(random == 3)
   PaintOval(&theRect);
  else if(random == 4)
   PaintArc(&theRect, 0, 300);
  else if(random == 5)
   TextSize(doRandomNumber(10, 70));
   MoveTo(left, right);
    DrawString("\pGWorldPicCursIcon");
}
// .....stop recording, draw picture, restore saved clipping region
ClosePicture();
DrawPi cture(pi ctureHdl, &pi ctureRect);
```

```
SetClip(oldClipRgn);
Di sposeRgn(oldClipRgn);
^{\prime\prime} ...... display some information from the PictInfo structure
RGBForeCol or (&gBl ueCol our);
RGBBackCol or (&gBei geCol our);
PenMode(patCopy);
OffsetRect(&pictureRect, 300, 0);
EraseRect(&pictureRect);
FrameRect(&pictureRect);
TextSize(10);
if(osErr = GetPictInfo(pictureHdl, &pictInfo, recordFontInfo + returnColorTable, 1,
                           systemMethod, 0))
  SysBeep(10);
MoveTo(380, 70):
DrawString("\pSome Picture Information:");
MoveTo(380, 100);
DrawString("\pLines: ");
NumToString(pictInfo.lineCount, theString);
DrawString(theString);
MoveTo(380, 115);
DrawString("\pRectangles: ");
NumToString(pictInfo.rectCount, theString);
DrawString(theString);
MoveTo(380, 130);
DrawString("\pRound rectangles: ");
NumToString(pictInfo.rRectCount, theString);
DrawString(theString);
MoveTo(380, 145);
DrawString("\pOvals: ");
NumToString(pictInfo.ovalCount,theString);
DrawString(theString);
MoveTo(380, 160);
DrawString("\pArcs: ");
NumToString(pictInfo.arcCount, theString);
DrawString(theString);
MoveTo(380, 175);
DrawString("\pPolygons: ");
NumToString(pictInfo.polyCount, theString);
DrawString(theString);
MoveTo(380, 190);
DrawString("\pRegions: ");
NumToString(pictInfo.regionCount,theString);
DrawString(theString);
MoveTo(380, 205);
DrawString("\pText strings: ");
NumToString(pictInfo.textCount, theString);
DrawString(theString);
MoveTo(380, 220);
DrawString("\pUnique fonts: ");
NumToString(pictInfo.uniqueFonts, theString);
DrawString(theString);
MoveTo(380, 235);
DrawString("\pUnique colours: ");
NumToString(pictInfo.uniqueColors,theString);
DrawString(theString);
MoveTo(380, 250);
DrawString("\pFrame rectangle left: ");
NumToString(pictInfo.sourceRect.left, theString);
DrawString(theString);
MoveTo(380, 265);
DrawString("\pFrame rectangle top: ");
```

```
NumToString(pictInfo.sourceRect.top, theString);
  DrawString(theString);
  MoveTo(380, 280);
  DrawString("\pFrame rectangle right: ");
  NumToString(pictInfo.sourceRect.right, theString);
  DrawString(theString);
  MoveTo(380, 295);
 DrawString("\pFrame rectangle bottom: ");
NumToString(pictInfo.sourceRect.bottom, theString);
  DrawString(theString);
  // .....release memory occupied by Picture structure
  KillPicture(pictureHdl);
                  voi d doCursor(voi d)
  Rect
         cursorRect;
  SInt16 a;
  RGBBackColor(&gBlueColour);
  EraseRect(&gWi ndowPtr->portRect);
  cursorRect = gWindowPtr->portRect;
  for (a=0; a<3; a++)
   InsetRect(&cursorRect, 40, 40);
   i f(a == 0 | | a == 2)
     RGBBackCol or(&gBei geCol our);
    else
      RGBBackColor(&gBlueColour);
   EraseRect(&cursorRect);
  RGBForeColor(&gBeigeColour);
  MoveTo(10, 20);
 DrawString("\pArrow cursor region");
RGBForeColor(&gBlueColour);
  MoveTo(50, 60);
 DrawString("\pIBeam cursor region");
RGBForeColor(&gBeigeColour);
  MoveTo(90, 100);
  DrawString("\pCross cursor region");
  RGBForeColor(&gBlueColour);
  MoveTo(130, 140);
  DrawString("\pPlus cursor region");
  gCursorRegionsActive = true;
  gCursorRegion = NewRgn();
void doChangeCursor(WindowPtr windowPtr, RgnHandle cursorRegion)
{
  Rect
           cursorRect;
  RgnHandle arrowCursorRgn;
  RgnHandle ibeamCursorRgn;
  RgnHandle crossCursorRgn;
  RgnHandle plusCursorRgn;
  Poi nt
           mousePosition;
  arrowCursorRgn = NewRgn();
  i beamCursorRgn = NewRgn();
  crossCursorRgn = NewRgn();
  plusCursorRgn = NewRgn();
  SetRectRgn(arrowCursorRgn, - 32768, - 32768, 32766, 32766);
```

```
cursorRect = windowPtr->portRect;
  Local ToGl obal (&topLeft(cursorRect));
  Local ToGl obal (&botRi ght(cursorRect));
  InsetRect(&cursorRect, 40, 40);
  RectRgn(ibeamCursorRgn, &cursorRect);
  \label{lem:cursor} \mbox{DiffRgn(arrowCursorRgn, ibeamCursorRgn, arrowCursorRgn);}
  InsetRect(&cursorRect, 40, 40);
  RectRgn(crossCursorRgn, &cursorRect);
  DiffRgn(ibeamCursorRgn, crossCursorRgn, ibeamCursorRgn);
  InsetRect(&cursorRect, 40, 40);
  RectRgn(plusCursorRgn, &cursorRect);
  DiffRgn(crossCursorRgn, plusCursorRgn, crossCursorRgn);
  GetMouse(&mousePosition);
  Local ToGlobal (&mousePosition);
  if(PtInRgn(mousePosition, ibeamCursorRgn))
  {
    SetCursor(*(GetCursor(iBeamCursor)));
    CopyRgn(i beamCursorRgn, cursorRegi on);
  else if(PtInRgn(mousePosition, crossCursorRgn))
    SetCursor(*(GetCursor(crossCursor)));
    CopyRgn(crossCursorRgn, cursorRegion);
  else if(PtInRgn(mousePosition, plusCursorRgn))
    SetCursor(*(GetCursor(plusCursor)));
    CopyRgn(plusCursorRgn, cursorRegion);
  else
  {
    SetCursor(&qd. arrow);
    CopyRgn(arrowCursorRgn, cursorRegion);
  Di sposeRgn(arrowCursorRgn);
  Di sposeRgn(i beamCursorRgn);
  Di sposeRgn(crossCursorRgn);
  Di sposeRgn(plusCursorRgn);
                     ******** doAni mCursor
voi d
    doAni mCursor(voi d)
  SInt16
           ani mCursResourceID, ani mCursTickInterval;
  BackCol or(whiteCol or);
  FillRect(&(gWindowPtr->portRect), &qd. white);
  ani mCursResourceI D
                     = rBeachBallCursor;
  animCursTickInterval = kBeachBallTickInterval;
  if(doGetAnimCursor(animCursResourceID, animCursTickInterval))
  {
    gAnimCursActive = true;
    gSleepTime = animCursTickInterval;
  else
    SysBeep(10);
   Boolean doGetAnimCursor(SInt16 resourceID, SInt16 tickInterval)
  SInt16 cursorID, a = 0;
  Boolean noError = false;
  if((gAnimCursHdl = (animCursHandle) GetResource('acur', resourceID)))
    noError = true;
```

```
while((a < (*gAnimCursHdl)->numberOfFrames) && noError)
     cursorID = (SInt16) \ Hi \ Word((SInt32) \ (*gAnimCursHdl) -> frame[a]);
     (*gAnimCursHdl)->frame[a] = GetCursor(cursorID);
     if((*gAnimCursHdl)->frame[a])
      a++:
     else
      noError = false;
   }
 }
 if(noError)
   gAnimCursTickInterval = tickInterval;
   gAni mCursLastTi ck = Ti ckCount();
   (*gAni mCursHdl) - >whi chFrame = 0;
 return(noError);
  voi d doSpi nAni mCursor(voi d)
 register SInt32 newTick;
 newTi ck = Ti ckCount();
 if(newTick < (gAnimCursLastTick + gAnimCursTickInterval))</pre>
   return:
 SetCursor(*((*gAni mCursHdl)->frame[(*gAni mCursHdl)->whi chFrame++]));
 if((*gAnimCursHdl)->whichFrame == (*gAnimCursHdl)->numberOfFrames)
   (*gAni mCursHdl)->whi chFrame = 0;
 gAnimCursLastTick = newTick;
  void doRel easeAni mCursor(void)
 SInt16 a;
 for(a=0;\,a<(\,{}^*gAni\,\,mCursHdl\,)\,\,\hbox{-}\,\,>number0fFrames;\,a++)
   ReleaseResource((Handle) (*gAnimCursHdl)->frame[a]);
 ReleaseResource((Handle) gAnimCursHdl);
  voi d doI dle(voi d)
 if(gAnimCursActive == true)
   doSpi nAni mCursor();
// ************************** dol con
voi d doI con(voi d)
 SInt 16
                a, b, stringIndex = 1;
 Rect
                theRect;
 IconTransformType transform = 0;
 Str255
                theString;
 Handl e
                i conSui teHdl;
 CI conHandl e
                ci conHdl;
 RGBForeColor(&gBlueColour);
 RGBBackColor(&gBeigeColour);
 EraseRect(&gWi ndowPtr->portRect);
                 PlotIconID with transforms
 MoveTo(50, 28);
 DrawString("\pPlotIconID with transforms");
```

```
for (a=50; a<471; a+=140)
   if(a == 190)
     transform = 16384;
    if(a == 330)
     transform = 256;
    for (b=0; b<4; b++)
      if(a == 470 \&\& b == 3)
        continue;
      GetIndString(theString, rTransformStrings, stringIndex++);
      MoveTo(a, b * 60 + 47);
     DrawString(theString);
      SetRect(&theRect, a, b * 60 + 50, a + 32, b * 60 + 82);
     PlotIconID(&theRect, 0, transform, rIconFamily1);
      SetRect(&theRect, a + 40, b * 60 + 50, a + 56, b * 60 + 66);
      PlotIconID(&theRect, 0, transform, rIconFamily1);
      SetRect(&theRect, a + 64, b * 60 + 50, a + 80, b * 60 + 62);
      PlotIconID(&theRect, 0, transform, rIconFamily1);
      if(a >= 330)
       transform += 256;
      else
        transform ++;
  }
                                      ......GetIconSuite and PlotIconSuite
  MoveTo(50, 275);
  Li neTo (550, 275);
  MoveTo(50, 299);
  DrawString("\pGetIconSuite and PlotIconSuite");
  GetIconSuite(&iconSuiteHdl, rIconFamily2, kSelectorAllLargeData);
  SetRect(&theRect, 50, 324, 82, 356);
  PlotIconSuite(&theRect, kAlignNone, kTransformNone, iconSuiteHdl);
  SetRect(&theRect, 118, 316, 166, 364);
  PlotIconSuite(&theRect, kAlignNone, kTransformNone, iconSuiteHdl);
  SetRect (&theRect, 202, 308, 266, 372);
  PlotIconSuite(&theRect, kAlignNone, kTransformNone, iconSuiteHdl);
                                                    ..... GetCIcon and PlotCIcon
  MoveTo(330, 299);
  DrawString("\pGetCIcon and PlotCIcon");
  ci conHdl = GetCI con(rCol ourI con);
  SetRect(&theRect, 330, 324, 362, 356);
  PlotCIcon(&theRect, ciconHdl);
  SetRect(&theRect, 398, 316, 446, 364);
  PlotCIcon(&theRect, ciconHdl);
  SetRect(&theRect, 482, 308, 546, 372);
  PlotCIcon(&theRect, ciconHdl);
voi d doAboutDi al og(voi d)
  DialogPtr dialogPtr;
           itemHit;
  di al ogPtr = GetNewDi al og(rAboutDi al og, gPreAllocatedBlockPtr, (WindowPtr) - 1);
  Modal Di al og(NULL, &i temHit);
  Di sposeDi al og(di al ogPtr);
// *********** doGWorldDrawing
void doGWorldDrawing(void)
```

```
SInt32
            a;
  SInt 16
            b. c:
  Rect
            theRect;
  RGBColor theColour;
  RGBForeColor(&gBeigeColour);
  Pai ntRect(&gWi ndowPtr->portRect);
  for (a=0; a<55000; a+=300)
    theColour.red = a;
    the Colour. blue = 55000 - a;
    theColour.green = 0;
    RGBForeColor(&theColour);
    for (b=10; b<531; b+=65)
      for(c=10; c<331; c+=64)
        SetRect(&theRect, b, c, b + 62, c + 61);
        Pai ntRect(&theRect);
   }
                                                      ****** doRandomNumber
UInt16 doRandomNumber(UInt16 minimum, UInt16 maximum)
  UInt16 randomNumber;
 SInt32 range, t;
  randomNumber = Random();
  range = maxi mum - mi ni mum;
  t = (randomNumber * range) / 65535;
  return(t + minimum);
```

# **Demonstration Program Comments**

If monitor colour depth is set to Thousands (or less), the Minimum Heap Size of 680K set in the CodeWarrior project will be adequate. If monitor depth is set to Millions, the Minimum Heap Size will need to be increased. The determinant is the memory required by the offscreen graphics world created within the function doOffScreenGWorld1.

When this program is run, the user should:

- Invoke the demonstrations by choosing items from the Demonstration menu, clicking the mouse when instructed to do so by the text in the window's title bar.
- Click outside and inside the window when the cursor and animated cursor demonstrations have been invoked.
- · Choose the About... item in the Apple menu to display the About... dialog box.
- · Note that the Icons item in the Demonstration menu contains an icon.

#### #define

In addition to the usual window and menu-related constants, constants are established for the resource IDs of 'acur', 'PICT', 'STR#', icon family, and 'cicn' resources, and a 'DLOG' resource. KSleeptime and MAXLONG will be assigned WaitNextEvent's sleep parameter at various points in the program. kBeachBallTickInterval represents the interval between frame changes for an animated cursor.

#### #typedef

The data type anumCurs is identical to the structure of an 'acur' resource.

#### Global Variables

gPreAllocatedBlock will be assigned a pointer to a nonrelocatable block for the dialog structure associated with the About... dialog. gWindowPtr will be assigned a pointer to the window's window structure. gDone controls exit from the main event loop and thus program termination. gInBackground relates to foreground/background switching.

In this program, the sleep and cursor region parameters in the WaitNextEvent call will be changed during program execution, hence the global variables gSleepTime and gCursorRegion.

gCursorRegion will be assigned a handle to a region which will passed in the mouseRgn parameter of the WaitNextEvent call. This relates to the cursor shape changing demonstration.

gCursorRegionActive and gAnimCursActive will be set to true during, respectively, the cursor and animated cursor demonstrations. gAnimCursHdl will be assigned a handle to the animCurs structure used during the animated cursor demonstration. gAnimCursTickInterval and gAnimCursLastTick also relate to the animated cursor demonstration.

gCursorRegionsActive will be set to true during the cursor shape changing demonstration. gAnimCursResourceHdl will be assigned a handle to the 'acur' structure associated with the second of two animated cursors. gAnimCurs1Active and gAnimCurs2Active will be set to true during the animated cursor demonstrations.

#### main

The first action in the main function is to pre-allocate a nonrelocatable block for the dialog structure for the About... modal dialog. This is an anti-heap-fragmentation measure.

Random numbers will be used in the function doPicture. The call to GetDateTime seeds the random number generator.

Note that error handling here and in other areas of the program is somewhat rudimentary: the program simply terminates, sometimes with a call to SysBeep(10).

#### eventLoop

eventLoop contains the main event loop. The event loop terminates when gDone is set to true. Before the loop is entered, gSleepTime is set to MAXLONG. Initially, therefore, the sleep parameter in the WaitNextEvent call is set to the maximum possible value.

The global variable passed in the mouseRgn parameter of the WaitNextEvent call is assigned NULL so as to defeat the generation of mouse-moved events.

Each time round the loop, before the WaitNextEvent call, if the cursor shape changing demonstration is under way (gCursorRegionActive = true) and the application is not in the background, the application-defined function doChangeCursor is called.

If a null event is received, the application-defined function doIdle is called. The doIdle function has to do with the animated cursors demonstrations.

#### doEvents

In the inDrag case, after the call to DragWindow, and provided the cursor shape changing demonstration is currently under way, the application-defined function doChangeCursor is called. The regions controlling the generation of mouse-moved events are defined in global coordinates, and are based on the window's port rectangle. Accordingly, when the window is moved, the new location of the port rectangle, in global coordinates, must be re-calculated so that the various cursor regions may be re-defined. The call to doChangeCursor re-defines these regions for the new window location and copies the handle to one of them, depending on the current location of the mouse cursor, to the global variable gCursorRegion. (Note that this call to doChangeCursor is also required, for the same reason, when a window is re-sized or zoomed.)

In the case of a resume event, SetCursor is called to ensure that the cursor is set to the arrow shape. The QuickDraw global variable arrow, which is of type Cursor, and which contains the arrow shaped cursor image, is passed in the newCursor parameter.

In the case of a mouse-moved event (which occurs when the mouse cursor has moved outside the region whose handle is currently being passed in WaitNextEvent's mouseRgn parameter), doChangeCursor is called to change the handle passed in the mouseRgn parameter according to the current location of the mouse.

#### doMenuChoice

doMenuChoice processes Apple and File menu choices to completion, with the exception of a choice of the About… item in the Apple menu. In this latter case, the application-defined function doAboutDialog is called. Choices from the Demonstration menu result in calls to application-defined functions.

Before the main switch, however, certain actions relevant to the animated cursor and cursor shape changing demonstrations are taken.

Firstly, if the animated cursor demonstration is currently under way, the flag which indicates this condition is set to false, the cursor is set to the standard arrow cursor, memory associated with the animated cursor is deallocated, and WaitNextEvent's sleep parameter is set to the maximum possible value.

Secondly, if the cursor shape changing demonstration is currently under way, the global variable which signifies that situation is set to false. In addition, the region containing the current cursor region is disposed of and the associated global variable is set to NULL, thus defeating the generation of mouse-moved events.

#### doOffScreenGWorld1

doWithoutOffScreenGWorld is the first demonstration.

#### Draw in Window

As a prelude for what is to come, the application-defined function doGWorldDrawing is called to repeatedly paint some rectangles in the window in simulation of drawing operations that take a short but nonetheless perceptible period of time to complete. This will be contrasted with the alternative of completing the drawing in an offscreen graphics port and then copying it to the on-screen port.

#### Draw in Offscreen Graphics Port and Copy to Window

Firstly, the cursor is set the watch shape to indicate to the user that an operation which will take but a second or two is taking place.

The call to GetGWorld saves the current graphics world, that is, the current colour graphics port and the current device.

The call to NewGWorld creates an offscreen graphics world. The offscreenGWorld parameter receives a pointer to the offscreen graphics world's graphics port. O in the pixelDepth means that the offscreen world's pixel depth will be set to the deepest device intersecting the rectangle passed as the boundsRect parameter. This Rect passed in the boundsRect parameter becomes the offscreen port's portRect, the offscreen pixel map's bounds and the offscreen device's gdRect. NULL in the cTable parameter causes the default colour table for the pixel depth to be used. The aGDevice parameter is set to NULL because the noNewDevice flag is not set. O in the flags parameter means that no flags are set.

The call to SetGWorld sets the graphics port pointed to by gworldPortPtr as the current graphics port. (When the first parameter is a GWorldPtr, the current device is set to the device attached to the offscreen world and the second parameter is ignored.)

GetGWorldPixMap gets a handle to the offscreen pixel map and LockPixels called to prevent the base address of the pixel image from being moved when the pixel image it is drawn into or copied from.

The call to EraseRect clears the offscreen graphics port before the application-defined function doGWorldDrawing is called to draw some graphics in the offscreen port.

With the drawing complete, the call to SetGWorld sets the (saved) window's colour graphics port as the current port and the saved device as the current device.

The next two lines establish the source and destination rectangles (required by the forthcoming call to CopyBits) as equivalent to the offscreen graphics world and window port rectangles respectively. The calls to RGBForeColor and RGBBackColor set the foreground and background colours to black and white respectively, which is required to ensure that the CopyBits call will produce predictable results in the colour sense.

The CopyBits call copies the image from the offscreen world to the window. The call to QDError checks for any error resulting from the last QuickDraw call (in this case, CopyBits).

UnlockPixels unlocks the offscreen pixel image buffer and DisposeGWorld deallocates all of the memory previously allocated for the offscreen graphics world.

Finally, SetCursor sets the cursor shape back to the standard arrow cursor.

#### DoOffScreenGWorld2

doWithoutOffScreenGWorld demonstrates the use of CopyDeepMask to copy a source pixel map to a destination pixel map using a pixel map as a mask, and clipping the copying operation to a designated region. Because mask pixel maps cannot come from the screen, an offscreen graphics world is created for the mask.

The first block loads a 'PICT' resource and draws the picture in the window.

The current graphics world is then saved and an offscreen graphics world the same size as the drawn picture is created. The offscreen graphics port is set as the current port, the pixel map is locked, and the offscreen port is erased.

The second call to GetPicture loads the 'PICT' resource representing the mask and DrawPicture is called to draw the mask in the offscreen port.

SetGWorld is then called again to make the window's colour graphics port the current port. The mask is then also drawn in the window next to the source image so that the user can see a copy of the mask in the offscreen graphics port.

The next two blocks define two regions, one containing an oval and one a rounded rectangle. The handles to these regions will be passed in the maskRgn parameter of two separate calls to CopyDeepMask.

Before the calls to CopyDeepMask, the foreground and background colours are set to black and white respectively so that the results of the copying operation, in terms of colour, will be predictable.

The for loop causes the source image to be copied to two locations in the window using a different mask region and Boolean source mode for each copy. The first time CopyDeepMask is called, the oval-shaped region is passed in the maskRgn parameter and the source mode srcCopy is passed in the mode parameter. The second time CopyDeepMask is called, the round rectangle-shaped region and srcOr and passed.

QDError checks for any error resulting from the last QuickDraw call (in this case, CopyDeepMask).

In the clean-up, UnlockPixels unlocks the offscreen pixel image buffer, DisposeGWorld deallocates all of the memory previously allocated for the offscreen graphics world, and the memory occupied by the picture resources and regions is released. Note that, because the pictures are resources obtained via GetPicture, ReleaseResource, rather than KillPicture, is used.

#### doPicture

doPicture demonstrates the recording and playing back a picture.

#### Define Picture Rectangle and Set Clipping Region

The window's port rectangle is copied to a local Rect variable. This rectangle is then made equal to the left half of the port rectangle, and then inset by 10 pixels all round. This is the picture rectangle

The clipping region is then set to be the equivalent of this rectangle. (Before this call, the clipping region is very large. In fact, it is as large as the coordinate plane. If the clipping region is very large and you scale a picture while drawing it, the clipping region can become invalid when DrawPicture scales the clipping region - in which case the picture will not be drawn.)

#### Set up OpenCPicParams Structure

This block assigns values to the fields of an OpenCPicParams structure. These specify the previously defined rectangle as the bounding rectangle, and 72 pixels per inch resolution both horizontally and vertically. The version field should always be set to -2.

#### **Record Picture**

OpenCPicture initiates the recording of the picture definition. The address of the OpenCPicParams structure is passed in the newHeader parameter.

The picture is then drawn. Lines, rectangles, round rectangles, ovals, wedges, and text are drawn in random colours, and sizes.

#### Stop Recording, Draw Picture, Restore Saved Clipping Region

The call to ClosePicture terminates picture recording and the call to DrawPicture draws the picture by "playing back" the "recording" stored in the specified Picture structure.

The call to SetClip restores the saved clipping region and DisposeRgn frees the memory associated with the saved region.

#### Display Some Information From The Pictinfo Structure

The call to GetPictInfo returns information about the picture in a picture information structure. Information in some of the fields of this structure is then drawn in the right side of the window.

#### Release Memory Occupied By Picture Structure

The call to KillPicture releases the memory occupied by the Picture structure.

#### doCursor

doCursor's chief purpose is to assign true to the global variable gCursorRegionsActive, which will cause the application-defined function doChangeCursor to be called from within main event loop provided the application is not in the background. In addition, it erases some rectangles in the window which visually represent to the user some cursor regions which will later be established by the doChangeCursor function.

The last two lines sets the gCursorRegionsActive flag to true and create an empty region for the last parameter of the WaitNextEvent call in the main event loop. A handle to a cursor region will be copied to gCursorRegion in the application-defined function doChangeCursor.

#### doChangeCursor

doChangeCursor is called whenever a mouse-moved event is received and after the window is dragged.

The first four lines create new empty regions to serve as the regions within which the cursor shape will be changed to, respectively, the arrow, I-beam, cross, and plus shapes.

The SetRectRgn call sets the arrow cursor region to, initially, the boundaries of the coordinate plane. The next three lines establish a rectangle equivalent to the window's port rectangle and change this rectangle's coordinates from local to global coordinates so that the regions calculated from it will be in the required global coordinates. The call to InsetRect insets this rectangle by 40 pixels all round and the call to RectRgn establishes this as the I-beam region. The call to DiffRgn, in effect, cuts the rectangle represented by the I-beam region from the arrow region, leaving a hollow arrow region.

The next six lines use the same procedure to establish a rectangular hollow region for the cross cursor and an interior rectangular region for the plus cursor. The result of all this is a rectangular plus cursor region in the centre of the window, surrounded by (but not overlapped by) a hollow rectangular cross cursor region, this surrounded by (but not overlapped by) a hollow rectangular I-beam cursor region, this surrounded by (but not overlapped by) a hollow rectangular arrow cursor region the outside of which equates to the boundaries of the coordinate plane.

The call to GetMouse gets the point representing the mouse's current position. Since GetMouse returns this point in local coordinates, the next line converts it to global coordinates.

The next task is to determine the region in which the cursor is currently located. The calls to PtInRgn are made for that purpose. Depending on which region is established as the region in which the cursor in currently located, the cursor is set to the appropriate shape and the handle to that region is copied to the global variable passed in WaitNextEvent's mouseRgn parameter.

That accomplished, the last four lines deallocate the memory associated with the regions created earlier in the function.

#### doAnimCursor

 $\ do Ani\, m Cursor\ responds\ to\ the\ user's\ selection\ of\ the\ Ani\, mated\ Cursor\ item\ in\ the\ Demonstration\ menu.$ 

In this demonstration, application-defined functions are utilised to retrieve 'acur' and 'CURS' resources, spin the cursor, and deallocate the memory associated with the animated cursor when the cursor is no longer required. These functions are generic in that they may be

used to initialise, spin and release any animated cursor passed to the getAnimCursor function as a formal parameter. A "beach-ball" cursor is utilised in this demonstration. doAnimCursor's major role is simply to call getAnimCursor with the beach-ball 'acur' resource as a parameter.

The fourth line assigns the resource ID of the beach-ball 'acur' resource to the variable used as the first parameter in the later call to getAnimCursor call. The fifth line assigns a value represented by a constant to the second parameter in the getAnimCursor call. This value controls the frame rate of the cursor, that is, the number of ticks which must elapse before the next frame (cursor) is displayed. (The best frame rate depends on the type of animated cursor used.)

If the call to getAnimCursor is successful, the flag gAnimCursActive is set to true and, importantly, the sleep parameter in the WaitNextEvent call is set to the same ticks value as that used to control the cursor's frame rate. This latter will cause null events to be generated at that tick interval (assuming, of course, that no other events intervene). Recall that the doIdle function is called whenever a null event is received and that, if the flag gAnimCursActive is set to true, doIdle calls the spinCursor function.

If the call to  $getAnimCursor\ fails$ , doAnimCursor  $simply\ plays\ the\ system\ alert\ sound\ and\ returns.$ 

#### doGetAnimatedCursor

doGetAnimatedCursor retrieves the data in the specified 'acur' resource and stores it in an animCurs structure, retrieves the 'CURS' resources specified in the 'acur' resource and assigns the handles to the resulting Cursor structures to elements in an array in the animCurs structure, establishes the frame rate for the cursor, and sets the starting frame number.

GetResource is called to read the 'acur' resource into memory and return a handle to the resource. The handle is cast to type animCursHandle and assigned to the global variable gAnimCursHdl (a handle to a structure of type animCurs, which is identical to the structure of an 'acur' resource). If this call is not successful (that is, GetResource returns NULL), the function will simply exit, returning false to doAnimCursor. If the call is successful, noError is set to true before a while loop is entered. This loop will cycle once for each of the 'CURS' resources specified in the 'acur' resource - assuming that noError is not set to false at some time during this process.

The ID of each cursor is stored in the high word of the specified element of the frame[] field of the animCurs structure. This is retrieved. The cursor ID is then used in the call to GetCursor to read in the resource (if necessary) and assign the handle to the resulting 68-byte Cursor structure to the specified element of the frame[] field of the animCurs structure. If this pass through the loop was successful, the array index is incremented; otherwise, noError is set to false, causing the loop and the function to exit, returning false to doAnimCursor

The first line within the if block assigns the ticks value passed to doGetAnimCursor to a global variable which will be utilised in the function doSpinCursor. The next line assigns the number of ticks since system startup to another variable which will also be utilised in the function doSpinCursor. The third line sets the starting frame number.

At this stage, the animated cursor has been initialised and doIdle will call doSpinAnimCursor whenever null events are received.

#### doSpinAnimCursor

doSpinAnimCursor is called whenever null events are received (that is, in this demonstration, every 5 ticks assuming no other events intervene).

The first line assigns the number of ticks since system startup to newTick. The next line checks whether 5 ticks have elapsed since the previous call to doSpinAnimCursor. If 5 ticks have not elapsed, the function simply returns. Otherwise, SetCursor sets the cursor shape to that represented by the handle stored in the specified element of the frame[] field of the animCurs structure. This line also increments the frame counter field (whichFrame) of the animCurs structure. If whichFrame has been incremented to the last cursor in the series, the frame counter is re-set to 0. The last line retrieves and stores the tick count at exit for use at the first line the next time the function is called.

#### doReleaseAnimCursor

doReleaseAnimCursor deallocates the memory occupied by the Cursor structures and the 'acur' resource.

Recall that doReleaseAnimCursor is called when the user clicks in the menu bar and that, at the same time, the gAnimCursActive flag is set to false, the cursor is reset to the standard arrow shape, and WaitNextEvent's sleep parameter is reset to the maximum possible value.

#### doldle

doIdle is called from the main event loop whenever a null event is received. If the active demonstration is the animated cursor demonstration, the application-defined function doSpinCursor is called.

#### dolcon

 $\label{localization} \mbox{doI\,con demonstrates the drawing of icons in a window using PlotIconID, PoltIconSuite, and PlotCIcon.}$ 

#### PlotIconID With Transforms

This block uses the function PlotIconID to draw an icon from an icon family with the specified ID fifteen times, once for each of the fifteen available transform types. PlotIconID automatically chooses the appropriate icon resource from the icon suite depending on the specified destination rectangle and the bit depth of the current device.

#### GetIconSuite and PlotIconSuite

This block uses GetIconSuite to get an icon suite comprising only the 'ICN#', 'icl4', and 'icl8' resources from the icon family with the specified resource ID. PlotIconSuite is then called three times to draw the appropriate icon within destination rectangles of three different sizes. PlotIconSuite automatically chooses the appropriate icon resource from the icon suite depending on the specified destination rectangle and the bit depth of the current device. PlotIconSuite also expands the icon to fit the last two destination rectangles.

#### GetClcon and PlotClcon

This block uses GetCIcon to load the specified 'cicn' resource and PlotCIcon to draw the colour icon within destination rectangles of three different sizes. PlotCIcon expands the 32 by 32 pixel icon to fit the last two destination rectangles.

#### doAboutDialog

doAboutDialog is called when the user chooses the About... item in the Apple menu.

GetNewDialog creates a modal dialog. The dialog's item list contains a picture item, which fills the entire dialog window. Note that a pointer to a pre-allocated nonrelocatable memory block is passed in the dStorage parameter

The call to Modal Dialog means that the dialog will remain displayed until the user clicks somewhere within the dialog box, at which time DisposeDialog is called to dismiss the dialog and free the associated memory. A dialog box rather than an alert box is used to obviate the need for a button for dismissing the dialog.

#### doGWorldDrawing and doRandomNumber

doGWorldDrawing and doRandomNumber are both incidental to the demonstration, doGWorldDrawing is called from doOffScreenGWorld1 to execute a drawing operation which will take a short but nonetheless perceptible period of time. doRandomNumber generates the random numbers used within the doPicture function.