13

OFFSCREEN GRAPHICS WORLDS, PICTURES, CURSORS, AND ICONS

Demonstration Program: GWorldPicCursIcn

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 involved. When your application draws into an offscreen graphics world, it draws into a part of memory not used by the video device. Thus the drawing process remains hidden from the user. When the drawing is completed, your application can copy the image from the offscreen graphics world to the active window using the `CopyBits`, `CopyMask`, or `CopyDeepMask` functions.

One of the key advantages of using an offscreen graphics world is speed. Copying a complex image from an offscreen graphics world to the active window is much faster than performing all the steps necessary to draw the image on-screen.

Creating an Offscreen Graphics World

The `NewGWorld` function is used to create an offscreen graphics world:

```
QDErr  NewGWorld(GWorldPtr *offscreenGWorld,short PixelDepth,const Rect *boundsRect,
                 CTabHandle cTable,GDHandle aGDevice,GWorldFlags flags)
```

Returns: A result code: `noErr` (no error); `paramErr` (illegal parameter); `cDepthErr` (invalid pixel depth).

`offscreenGWorld`	Pointer to the created offscreen graphics world.
`PixelDepth`	Pixel depth of the offscreen graphics world. Possible depths are 0, 1, 2, 4, 8, 16, and 32 bits per pixel. Specifying `0` sets the pixel depth to equal the greatest depth of those screens whose boundary rectangles intersect the rectangle passed in the `boundRect` parameter. `0` also causes `NewGWorld` to use the `GDevice` structure for this deepest device rather than create a new one.
`boundsRect`	The offscreen pixel maps's boundary and port rectangle. Applications typically pass in the port rectangle of the window to which the image in the offscreen graphics world will be copied.
`cTable`	Handle to a `ColorTable` structure. May be `NULL`.
`aGDevice`	Handle to a `GDevice` structure. This is used only when `noNewDevice` is passed in the `flags` parameter. `NewGWorld` will attach this `GDevice` structure to the offscreen graphics world. Should be `NULL` if `0` is passed in the `PixelDepth` parameter.

`flags`	Any combination of `pixPurge` (make base address of pixel image purgeable), `noNewDevice` (do not create offscreen `GDevice` structure), `useTempMem` (create base address for `offscreen` pixel image in temporary memory, and `keepLocal` (keep offscreen pixel image in main memory) may be passed in this parameter.

Calling `NewGWorld` results in the creation of a new offscreen graphics port. The function returns, in the `offscreenGWorld` parameter, a pointer of type `GWorldPtr` which points to the graphics port:

```
typedef CGrafPtr GWorldPtr;
```

`NewGWorld` also establishes a link to an existing `GDevice` structure, or creates a new `GDevice` structure and establishes a link to that.

Passing `0` in the `PixelDepth` parameter, a window's port rectangle in the `boundsRect` parameter, `NULL` in both the `cTable` and `aGDevice` parameters, and `0` in the `flags` parameter:

- Allows QuickDraw to optimise the `CopyBits`, `CopyMask`, and `CopyDeepMask` functions used to copy the image into the window's port rectangle.
- Results in the default behaviour of `NewGWorld`, meaning that the base address of the offscreen pixel image is unpurgeable, memory in the application heap is used, and graphics accelerators can cache the offscreen pixel image.

Setting the Graphics Port

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

`SetGWorld` 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 `GWorldPtr`, 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 linked 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 `GetGWorldPixMap` function to get a handle to the offscreen pixel map. This is required as the parameter in a call to the `LockPixels` function, which you must call before drawing to, or copying from, an offscreen graphics world.

`LockPixels` prevents the base address of an offscreen pixel image from being moved while you draw into it or copy from it. It returns `true` if the base address is not purgeable, or if the base address has not been purged by the Memory Manager. If `LockPixels` returns `false`, (meaning that the base address of the offscreen pixel image has been purged) your application must call the `UpdateGWorld` function to reallocate the offscreen pixel image and then reconstruct it.

As a related matter, note that the `baseAddr` field of the `PixMap` structure for an offscreen graphics world contains a handle, whereas the `baseAddr` field for an onscreen pixel map contains a pointer. Accordingly, the `GetPixBaseAddr` function must be used to obtain a pointer to the `PixMap` 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 set the active window as the current graphics port preparatory to copying the image to that port.

Your application copies the image from the offscreen graphics world into the target window using `CopyBits` (or, if masking is required, `CopyMask` or `CopyDeepMask`). Note that `CopyBits`, `CopyMask` and `CopyDeepMask` expect their source and destination parameters to be pointers to bit maps, not pixel maps. (These functions date from the era of black-and-white Macintoshes, which is why they expect a pointer to a bitmap. By looking at certain information in the graphics ports, `CopyBits`, `CopyMask`, and `CopyDeepMask` can establish that you have passed the functions a handle to a pixel map rather than the base address of a bitmap.)

You must leave the pixel image locked while you are drawing into an offscreen graphics world or copying an image from it, and you should call `UnlockPixels` when you are finished the copying or drawing operation. (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 `UpdateGWorld` to carry certain changes affecting the window (resizing the window, changes to the pixel depth of the screen, etc.) through to the offscreen graphics world. Calling `UpdateGWorld` obviates the necessity to recreate the offscreen graphics world and redraw its contents.

Disposing of an Offscreen Graphics World

You should call `DisposeGWorld` when your application no longer needs the offscreen graphics world.

Pictures

Introduction

QuickDraw provides a set of functions that allow your application to record a number of drawing commands and subsequently play the recording back. The collection of drawing commands is called a **picture.**

You begin defining a picture by calling the function `OpenCPicture`. Your subsequent drawing commands are collected in a data structure of type `Picture`. The picture defined within this data structure may be drawn by calling the function `DrawPicture`.

The `OpenCPicture` function creates pictures in the **extended version 2 format**, which allows your application to specify resolutions for pictures.

The Picture Structure

The `Picture` structure is as follows:

```
struct Picture
{
  short picSize;
  Rect  picFrame;
};
typedef struct Picture Picture;
typedef Picture *PicPtr;
typedef PicPtr *PicHandle;
```

Field Descriptions

`picSize` This field is irrelevant for version 2 format and extended version 2 format pictures.

> **Note**
>
> To determine the size of a picture in memory, use the Memory Manager function `GetHandleSize`. To determine the size of a picture in a file of type `'PICT'`, use the File Manager function `PBGetFInfo`. To determine the size of a picture in a resource of type `'PICT'`, use the Resource Manager function `MaxSizeResource`.

`picFrame` The picture's bounding rectangle. When you draw into a differently sized rectangle, `DrawPicture` uses this rectangle to scale the picture.

... 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 `DrawPicture` uses to determine what objects to draw or what mode to change for subsequent drawing. 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.

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

'PICT' Files, Resources, and Scrap Format

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. (See Chapter 20.)

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 first 512 bytes of the data fork of a `'PICT'` file are a header that your application can use for its own purposes.

The Scrap Manager maintains a storage area to hold the last data cut or copied by the user. This area is called the **scrap**. If your application supports cut, copy, and paste operations, it necessarily reads data from, and writes data to, the scrap. There are two standard scrap data formats, one of which is `'PICT'`.

Creating Pictures

As previously stated, you use the `OpenCPicture` function to begin defining a picture. You pass information to `OpenCPicture` in the form of an `OpenCPicParams` structure:

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

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.

You call `ClosePicture` to complete the collection of drawing (and picture comment) commands that define your picture.

Clipping Region

Before calling `OpenCPicture`, you should always use `ClipRect` to specify an appropriate clipping region. If you fail to do this, `OpenCPicture` will use the clipping region contained in the current graphics port object. By default, this region is very large (the size of the coordinate plane). In this circumstance, if you scale the picture when drawing it, the clipping region can become invalid and your picture will not be drawn. By the same token, if your application has previously set the clipping region for some other purpose, part of your drawing may be clipped.

Ordinarily, you should set the clipping region to equal the port rectangle of the current graphics port before recording a picture.

Opening and Drawing Pictures

You can retrieve pictures saved in `'PICT'` files using File Manager functions.[1] You can retrieve pictures saved in the resource forks of other file types using the `GetPicture` function. You can retrieve pictures stored in the scrap using the Carbon Scrap Manager function `GetScrapFlavorData`.

When the picture is retrieved, you can call `DrawPicture` to draw the picture. The second parameter passed in the `DrawPicture` function is the destination rectangle, which should be specified in coordinates local to the current graphics port. `DrawPicture` 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

To save a picture in a `'PICT'` file, you should use the appropriate File Manager functions.[1] (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 Carbon Scrap Manager function `PutScrapFlavorData`.

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.

Cursors

Introduction

A **cursor** is a 16-by-16 pixel image defined in a black-and-white cursor (`'CURS'`) or colour cursor (`'crsr'`) resource.

Cursor Movement, Hot Spot, Visibility, and Shape

Cursor Movement

Cursor movement is not the responsibility of your application. When the mouse is moved by the user, low-level interrupt-driven mouse functions move the cursor on the screen.

Cursor Hot Spot

A cursor's **hot spot** is that part of the cursor that actually points to an object on the screen. Mouse clicks only have an effect on that object when the hot spot, not the cursor as a whole, is over the object. Fig 1 illustrates two cursors and their hot spot points. Note that the hot spot is a point, not a bit.

[1] The demonstration program at Chapter 18 shows how to read pictures from, and save pictures to, files of type `'PICT'`.

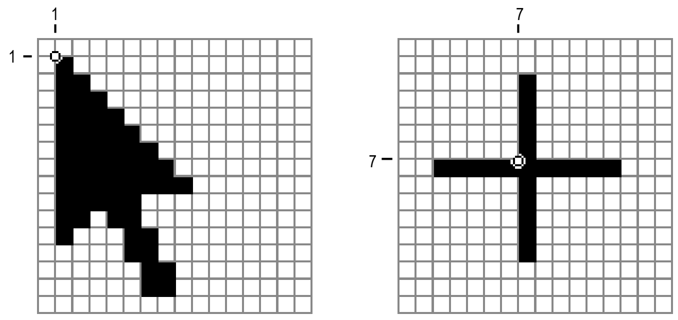

FIG 1 - CURSOR HOT SPOTS

Cursor Visibility

Generally speaking, your application should always make the cursor visible. There are, however, exceptions to this rule. 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 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. When the cursor is in the menu bar, for example, it should usually have an arrow shape. When the user moves the cursor over a text document, the cursor shape should be changed to the I-beam shape.
- To provide feedback to the user indicating that a time-consuming operation is in progress. 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 (on Mac OS 8/9) or wait cursor (Mac OS X) (see Fig 2). If the operation will take several seconds and the only options available to the user are to stop the operation, wait until it is completed, or switch to another application, you should display an animated cursor (on Mac OS 8/9 or wait cursor (on Mac OS X).[2]

Non-Animated Cursors

System 'CURS' and 'crsr' Resources

The system contains a number of `'CURS'` and `'crsr'` resources. The following constants represent the `'CURS'` resource IDs for the basic cursors shown at Fig 2:

Constant	***Value***	***Description***
`iBeamCursor`	`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.

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

[2] If the operation takes longer than several seconds, you should display a dialog with a progress indicator. (See Chapter 25.)

The following lists the `'CURS'` and `'crsr'` resource IDs for the additional cursors shown at Fig 3:

Constant	***Value***	***Description***
-	`-20488`	Contextual menu arrow cursor.
-	`-20487`	Alias arrow cursor.
-	`-20486`	Copy arrow cursor.
-	`-20452`	Resize left cursor. (Not available on Mac OS X.)
-	`-20451`	Resize right cursor. (Not available on Mac OS X.)
-	`-20450`	Resize left/right cursor. (Not available on Mac OS X.)
-	`-20877`	Pointing hand cursor. (Not available on Mac OS X.)
-	`-20876`	Open hand pointer. (Not available on Mac OS X.)
-	`-20875`	Close hand pointer. (Not available on Mac OS X.)

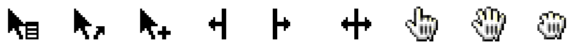

FIG 3 - ADDITIONAL CURSOR AND COLOUR CURSOR RESOURCES

Custom 'CURS' and 'crsr' Resources

To create custom cursors, you need to define `'CURS'` or `'crsr'` resources in the resource file of your application.

Changing 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.

Methodology 1

One method for changing cursor shape involves first getting a handle to the relevant cursor (either a custom cursor or one of the system cursors shown at Figs 2 and 3) by specifying its resource ID in a call to `GetCursor` or `GetCCursor`. `GetCursor` returns a handle to a `Cursor` structure. `GetCCursor` returns a handle to a `CCrsr` structure. The address of the `Cursor` or `CCrsr` structure is then used in a call to `SetCursor` or `SetCCursor` to change the cursor shape.

Methodology 2

Mac OS 8.5 introduced a new method for setting the cursor. You must pass one of the following constants, which are of type `ThemeCursor`, in the `inCursor` parameter of the function `SetThemeCursor`:

Constant	***Value***	***Comments***
`kThemeArrowCursor`	`0`	
`kThemeCopyArrowCursor`	`1`	
`kThemeAliasArrowCursor`	`2`	
`kThemeContextualMenuArrowCursor`	`3`	
`kThemeIBeamCursor`	`4`	
`kThemeCrossCursor`	`5`	
`kThemePlusCursor`	`6`	
`kThemeWatchCursor`	`7`	Can animate.
`kThemeClosedHandCursor`	`8`	
`kThemeOpenHandCursor`	`9`	
`kThemePointingHandCursor`	`10`	
`kThemeCountingUpHandCursor`	`11`	Can animate.
`kThemeCountingDownHandCursor`	`12`	Can animate.
`kThemeCountingUpAndDownHandCursor`	`13`	Can animate.
`kThemeSpinningCursor`	`14`	Can animate.
`kThemeResizeLeftCursor`	`15`	

kThemeResizeRightCursor	16
kThemeResizeLeftRightCursor	17

Changing Cursor Shape in Response to Mouse-Moved Events

Most applications set the cursor to the I-beam shape when the cursor is inside a text-editing 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 `mouseRgn` parameter to the `WaitNextEvent` function. Then, when a mouse-moved event is detected in your main event loop, you can use `SetCursor`, `SetCCursor`, or `SetThemeCursor`, to change the cursor to the appropriate shape.

Changing Cursor Shape in Response to Resume Events

Your application also needs to set the cursor shape in response to resume events, normally by setting the arrow cursor.

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.

Animated Cursors — Mac OS 8/9

Methodology 1

Mac OS 8.5 introduced a new function (`SetThemeAnimatedCursor`) for animating a specified cursor type. You must pass one of the following constants, which are of type `ThemeCursor`, in the `inCursor` parameter of `SetThemeAnimatedCursor`:

Constant	Value
kThemeWatchCursor	7
kThemeCountingUpHandCursor	11
kThemeCountingDownHandCursor	12
kThemeCountingUpAndDownHandCursor	13
kThemeSpinningCursor	14

Methodology 2

Another methodology requires:

- A series of `'CURS'` (or `'crsr'`) resources that make up the "frames" of the animation.
- An `'acur'` resource, which collects and orders the `'CURS'` frames into a single animation, specifying the IDs of the resources and the sequence for displaying them in the animation.

System 'acur', and 'CURS' Resources

The system contains an `'acur'` resource (ID -6079), together the associated eight `'CURS'` resources, for an animated watch cursor. It also contains eight `'CURS'` resources (IDs -20701 to -20708) for an animated spinning (beach ball) cursor and six `'CURS'` resources (IDs -20709 to -20714) for an animated counting hand cursor.

Custom 'acur' and 'CURS' Resources

Fig 4 shows the structure of a compiled `'acur'` resource, and an `'acur'` resource and one of its associated `'CURS'` resources being created using Resorcerer.

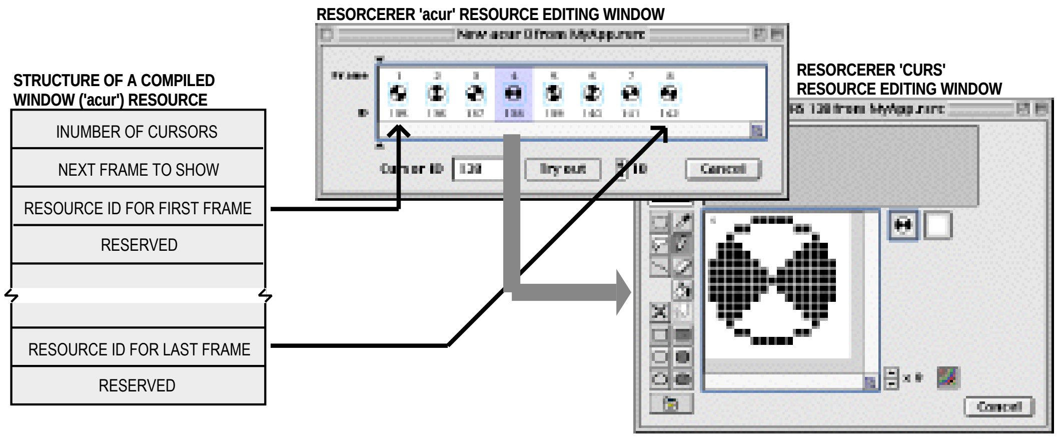
RESORCERER 'acur' RESOURCE EDITING WINDOW
STRUCTURE OF A COMPILED WINDOW ('acur') RESOURCE
NUMBER OF CURSORS
NEXT FRAME TO SHOW
RESOURCE ID FOR FIRST FRAME
RESERVED
RESOURCE ID FOR LAST FRAME
RESERVED
RESORCERER 'CURS' RESOURCE EDITING WINDOW
Cursor ID
Try out
Cancel

FIG 4 - CREATING AN 'acur' RESOURCE AND ASSOCIATED 'CURS' RESOURCES USING RESORCERER

Creating the Animated Cursor

The following are the steps required to create the animated cursor:

- If you do not intend to use the system-supplied `'acur'` and associated `'CURS'` resources:
  - Create a series of `'CURS'` resources that make up the "frames" of the animation.
  - Create an `'acur'` resource.
- Load the `'acur'` resource into a structure which replicates the structure of an `'acur'` resource, for example:

```
typedef struct
{
  short      numberOfFrames;
  short      whichFrame;
  CursHandle frame[];
} animCurs, *animCursPtr, **animCursHandle;
```

- Load the `'CURS'` resources using `GetCursor` and assign handles to the resulting `Cursor` structures to the elements of the `frame` field.
- At the desired interval, 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.

Animated Cursor — Mac OS X

When the Mac OS X wait cursor appears automatically, it means that the application has stopped calling an event handling API for more than a certain period of time (about two seconds).

Your application can also turn the wait cursor on and off using the QuickDraw function `QDDisplayWaitCursor`. Passing `true` in the `forceWaitCursor` parameter turns the cursor on and passing `false` resumes automatic operation. The function keeps track of nested calls.

Icons

Icons and the Finder — Icon Families

As stated at Chapter 9, the Finder uses **icons** to graphically represents objects, such as files and directories. 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.

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 a black-and-white icon defined in an `'ICON'` resource, which contains a 32-by-32 pixel bit map. Icons do not need a mask because they are always displayed on a white background.

Colour Icon ('cicn')

The colour icon is defined in a `'cicn'` resource, which includes a pixel map, a bit map, and a mask. You can use a `'cicn'` resource to define a colour icon with any width and height and with a bit depth up to 8. Fig 5 shows an 8-bit 32 by 32 pixel `'cicn'` resource being created using Resorcerer.

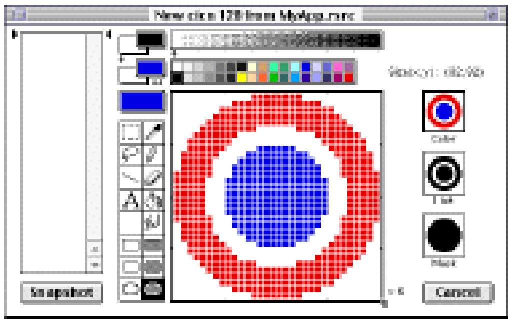

FIG 5 - CREATING AN 8-BIT 32 BY 32 PIXEL 'cicn' RESOURCE USING RESORCERER

Small Icon ('SICN')

The small icon is a black-and-white icon 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. Small icons are of doubtful utility in the Carbon era and will not be considered further.

Icons in Windows, Menus, and Alerts and Dialogs

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, dialogs 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) and `'cicn'` (colour 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 of the menu item definition. 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.)

When the menu is displayed, 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,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.

Icons in Alerts and Dialogs

The Dialog Manager allows you to display icons of resource types `'ICON'` (icon) and `'cicn'` (colour icon) in Mac OS 8/9 alerts and in dialogs. 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 `Icon` 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 using Dialog Manager functions.

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

If you provide a colour icon (`'cicn'`) 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.

On Mac OS 8/9, you would ordinarily use the `Alert` function (which does not automatically draw a system-supplied alert icon in the alert), 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). If you invoke an alert 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 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 dialogs.

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 dialogs.

Preamble - Icon Families and Icon Suites

Icon Families

You can define individual icons of resource types `'ICON'` and `'cicn'` that are not part of an icon family and use Icon Utilities functions to draw them as required. However, to display an icon effectively at a variety of sizes and bit depths, you should provide an icon family 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**. Typically, an icon suite comprises 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 all of the six icon resources that an icon family can contain. Alternatively, it can contain handles to only a subset of those resources.

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

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.	The 16-by-16 pixel icon with the appropriate bit depth.

Icon Stretching and Shrinking

`PlotIconID` may stretch or shrink the icon to fit depending on the size of the destination rectangle,. 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.

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 `GetIconSuite` function, with the constant `kSelectorAllAvailableData` 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 `PlotIconSuite` which, like `PlotIconID`, 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 that contains only the icon of the desired resource type together with its corresponding mask. Constants such as `kSelectorLarge4Bit` (an icon selector mask for an `'icl4'` 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 dialogs, you use Menu Manager and Dialog Manager functions such as `SetItemIcon` and `SetDialogItem`.

To draw resources of resource type `'ICON'` and `'cicn'` in your application's windows, you use the following functions:

Resource Type	Function to Get Icon	Functions to Draw Icon
`'ICON'`	`GetIcon`	`PlotIconHandle` `PlotIcon`
`'cicn'`	`GetCIcon`	`PlotCIconHandle` `PlotCIcon`

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.
pixelsLockedBit    = 7  Set to lock base address for offscreen pixel image.
```

Data Types

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

Functions

Creating, Altering, and Disposing of Offscreen Graphics Worlds

```
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);
```

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);
Boolean       LockPixels(PixMapHandle pm);
void          UnlockPixels(PixMapHandle pm);
void          AllowPurgePixels(PixMapHandle pm);
void          NoPurgePixels(PixMapHandle pm);
GWorldFlags   GetPixelsState(PixMapHandle pm);
void          SetPixelsState(PixMapHandle pm,GWorldFlags state);
Ptr           GetPixBaseAddr(PixMapHandle pm);
Boolean       PixMap32Bit(PixMapHandle pmHandle);
```

Main Constants, Data Types and Functions — Pictures

Constants

Verbs for the GetPictInfo, GetPixMapInfo, and NewPictInfo calls

```
returnColorTable      = 0x0001  Return a ColorTable structure.
returnPalette         = 0x0002  Return a Palette structure.
recordComments        = 0x0004  Return comment information.
recordFontInfo        = 0x0008  Return font information.
suppressBlackAndWhite = 0x0010  Do not include black and white.
```

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

```
systemMethod          = 0       System color pick method.
popularMethod         = 1       Most popular set of colors.
medianMethod          = 2       A good average mix of colors.
```

Data Types

Picture

```
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;
```

OpenCPicParams

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

PictInfo

```
struct  PictInfo
{
  short             version;         // This is always zero, for now.
  long              uniqueColors;    // Number of actual colors in the picture(s)/pixmap(s).
  PaletteHandle     thePalette;      // Handle to the palette information.
  CTabHandle        theColorTable;   // Handle to the color table.
  Fixed             hRes;            // Maximum horizontal resolution for all the pixmaps.
  Fixed             vRes;            // Maximum vertical resolution for all the pixmaps.
  short             depth;           // Maximum depth for all the pixmaps (in the picture).
  Rect              sourceRect;      // Picture frame rectangle (contains the entire picture).
  long              textCount;       // Total number of text strings in the picture.
  long              lineCount;       // Total number of lines in the picture.
  long              rectCount;       // Total number of rectangles in the picture.
  long              rRectCount;      // Total number of round rectangles in the picture.
  long              ovalCount;       // Total number of ovals in the picture.
  long              arcCount;        // Total number of arcs in the picture.
  long              polyCount;       // Total number of polygons in the picture.
  long              regionCount;     // Total number of regions in the picture.
  long              bitMapCount;     // Total number of bitmaps in the picture.
  long              pixMapCount;     // Total number of pixmaps in the picture.
  long              commentCount;    // Total number of comments in the picture.
  long              uniqueComments;  // The number of unique comments in the picture.
  CommentSpecHandle commentHandle;   // Handle to all the comment information.
  long              uniqueFonts;     // The number of unique fonts in the picture.
  FontSpecHandle    fontHandle;      // Handle to the FontSpec information.
  Handle            fontNamesHandle; // Handle to the font names.
  long  reserved1;
  long  reserved2;
};
typedef struct PictInfo PictInfo;
typedef PictInfo *PictInfoPtr;
typedef PictInfoPtr *PictInfoHandle;
```

CommentSpec

```
struct CommentSpec
{
  short count;       // Number of occurrences of this comment ID.
  short ID;          // ID for the comment in the picture.
};
typedef struct CommentSpec CommentSpec;
typedef CommentSpec *CommentSpecPtr;
typedef CommentSpecPtr *CommentSpecHandle;
```

FontSpec

```
struct FontSpec
{
  short pictFontID;  // ID of the font in the picture.
  short sysFontID;   // ID of the same font in the current system file.
  long  size[4];     // Bit array of all the sizes found (1..127) (bit 0 means > 127).
  short style;       // Combined style of all occurrances of the font.
  long  nameOffset;  // Offset into the fontNamesHdl handle for the font's name.
};
typedef struct FontSpec FontSpec;
typedef FontSpec *FontSpecPtr;
typedef FontSpecPtr *FontSpecHandle;
```

Functions

Creating and Disposing of Pictures

```
PicHandle  OpenCPicture(const OpenCPicParams *newHeader);
void       PicComment(short kind,short dataSize,Handle dataHandle);
void       ClosePicture(void);
void       KillPicture(PicHandle myPicture);
```

Drawing Pictures

```
void       DrawPicture(PicHandle myPicture,const Rect *dstRect)
PicHandle  GetPicture(Integer picID);
```

Collecting Picture Information

```
OSErr      GetPictInfo(PicHandle thePictHandle,PictInfo *thePictInfo,short verb,
           short colorsRequested,short colorPickMethod,short version);
OSErr      GetPixMapInfo(PixMapHandle thePixMapHandle,PictInfo *thePictInfo,short verb,
           short colorsRequested,short colorPickMethod,short version);
OSErr      NewPictInfo(PictInfoID *thePictInfoID,short verb,short colorsRequested,
           short colorPickMethod,short version);
OSErr      RecordPictInfo(PictInfoID thePictInfoID,PicHandle thePictHandle);
OSErr      RecordPixMapInfo(PictInfoID thePictInfoID,PixMapHandle thePixMapHandle);
OSErr      RetrievePictInfo(PictInfoID thePictInfoID,PictInfo *thePictInfo,
           short colorsRequested);
OSErr      DisposPictInfo(PictInfoID thePictInfoID);
```

Main Constants, Data Types and Functions — Cursors

Constants

```
iBeamCursor  = 1
crossCursor  = 2
plusCursor   = 3
watchCursor  = 4
```

Data Types

Cursor

```
struct Cursor
{
  Bits16      data;
  Bits16      mask;
  Point       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.
  Handle       crsrData;     // Cursor's data.
  Handle       crsrXData;    // Expanded cursor data.
  short        crsrXValid;   // Depth of expanded data (0 if none).
  Handle       crsrXHandle;  // Future use.
  Bits16       crsr1Data;    // One-bit cursor.
  Bits16       crsrMask;     // Cursor's mask.
  Point        crsrHotSpot;  // Cursor's hotspot.
  long         crsrXTable;   // Private.
  long         crsrID;       // Private.
};
typedef struct CCrsr CCrsr;
typedef CCrsr *CCrsrPtr;
typedef CCrsrPtr *CCrsrHandle;
```

Acur

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

Functions

Initialising Cursors

```
void         InitCursor(void);
void         InitCursorCtl(acurHandle newCursors);
```

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

```
void         HideCursor(void);
void         ShowCursor(void);
void         ObscureCursor(void);
void         ShieldCursor(const Rect *shieldRect,Point offsetPt);
void         RotateCursor(long counter);
pascal       void SpinCursor(short increment);
```

Appearance Manager Constants, Data Types and Functions — Cursors

Constants

```
KThemeArrowCursor            = 0
KThemeCopyArrowCursor        = 1
KThemeAliasArrowCursor       = 2
```

```
KThemeContextualMenuArrowCursor   = 3
KThemeIBeamCursor                 = 4
KThemeCrossCursor                 = 5
KThemePlusCursor                  = 6
KThemeWatchCursor                 = 7    Can animate
KThemeClosedHandCursor            = 8
KThemeOpenHandCursor              = 9
KThemePointingHandCursor          = 10
KThemeCountingUpHandCursor        = 11   Can animate
KThemeCountingDownHandCursor      = 12   Can animate
KThemeCountingUpAndDownHandCursor = 13   Can animate
KThemeSpinningCursor              = 14   Can Animate
KThemeResizeLeftCursor            = 15
KThemeResizeRightCursor           = 16
KThemeResizeLeftRightCursor       = 17
```

Data Types

```
typedef UInt32 ThemeCursor;
```

Functions

```
OSStatus  SetThemeCursor(ThemeCursor inCursor);
OSStatus  SetAnimatedThemeCursor(ThemeCursor inCursor,UInt32 inAnimationStep);
```

Mac OS X Only

```
void      QDDisplayWaitCursor(Boolean forceWaitCursor);
```

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')
kSmall8BitData              = FOUR_CHAR_CODE('ics8')
kMini1BitMask               = FOUR_CHAR_CODE('icm#')
kMini4BitData               = FOUR_CHAR_CODE('icm4')
kMini8BitData               = FOUR_CHAR_CODE('icm8')
```

IconAlignmentType Values

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

IconTransformType Values

```
kTransformNone              = 0x00
kTransformDisabled          = 0x01
kTransformOffline           = 0x02
kTransformOpen              = 0x03
kTransformLabel1            = 0x0100
```

```
kTransformLabel2          = 0x0200
kTransformLabel3          = 0x0300
kTransformLabel4          = 0x0400
kTransformLabel5          = 0x0500
kTransformLabel6          = 0x0600
kTransformLabel7          = 0x0700
kTransformSelected        = 0x4000
kTransformSelectedDisabled = kTransformSelected | kTransformDisabled
kTransformSelectedOffline  = kTransformSelected | kTransformOffline
kTransformSelectedOpen     = kTransformSelected | kTransformOpen
```

IconSelectorValue Masks

```
kSelectorLarge1Bit        = 0x00000001
kSelectorLarge4Bit        = 0x00000002
kSelectorLarge8Bit        = 0x00000004
kSelectorSmall1Bit        = 0x00000100
kSelectorSmall4Bit        = 0x00000200
kSelectorSmall8Bit        = 0x00000400
kSelectorMini1Bit         = 0x00010000
kSelectorMini4Bit         = 0x00020000
kSelectorMini8Bit         = 0x00040000
kSelectorAllLargeData     = 0x000000FF
kSelectorAllSmallData     = 0x0000FF00
kSelectorAllMiniData      = 0x00FF0000
kSelectorAll1BitData      = kSelectorLarge1Bit | kSelectorSmall1Bit | kSelectorMini1Bit
kSelectorAll4BitData      = kSelectorLarge4Bit | kSelectorSmall4Bit | kSelectorMini4Bit
kSelectorAll8BitData      = kSelectorLarge8Bit | kSelectorSmall8Bit | kSelectorMini8Bit
kSelectorAllAvailableData = (long)0xFFFFFFFF
```

Data Types

```
typedef short   IconAlignmentType;
typedef short   IconTransformType;
typedef UInt32  IconSelectorValue;
typedef Handle  IconSuiteRef;
typedef Handle  IconCacheRef;
```

CIcon

```
struct CIcon
{
  PixMap iconPMap;         // Icon's pixMap.
  BitMap iconMask;         // Icon's mask.
  BitMap iconBMap;         // Icon's bitMap.
  Handle iconData;         // Icon's data.
  short  iconMaskData[1];  // Icon's mask and BitMap data.
};
typedef struct CIcon CIcon;
typedef CIcon *CIconPtr;
typedef CIconPtr *CIconHandle;
```

Functions

Drawing Icons From Resources

```
OSErr    PlotIconID(constRect *theRect,IconAlignmentType align,IconTransformType transform,
         short theResID);
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,CIconHandle 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    DisposeCIcon(CIconHandle theIcon);
```

Creating an Icon Suite

```
OSErr    GetIconSuite(Handle *theIconSuite,short theResID,IconSelectorValue selector);
OSErr    NewIconSuite(Handle *theIconSuite);
OSErr    AddIconToSuite(Handle theIconData,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    IconSuiteToRgn(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 theIconSuite);
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 iconID);
```

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 GworldPicCursIcn Listing

```
// *******************************************************************************************
// GWorldPicCursIcon.c                                                   CLASSIC EVENT MODEL
// *******************************************************************************************
//
// This program demonstrates offscreen graphics world, picture, cursor, cursor shape change,
// animated cursor, and icon operations as a result of the user choosing items from a
// Demonstration menu.  It also demonstrates a modal dialog-based About… box containing a
// picture.
//
// 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:
//
// •  A 'plst' resource.
//
// •  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).
//
// •  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.
//
// •  A 'SIZE' resource with the acceptSuspendResumeEvents, canBackground,
//    doesActivateOnFGSwitch, and isHighLevelEventAware flags set.
//
// *******************************************************************************************

// ……………………………………………………………………………………………………………………………………………………………………………………… includes

#include <Carbon.h>

// ……………………………………………………………………………………………………………………………………………………………………………………… defines

#define rMenubar                 128
#define rWindow                  128
#define mAppleApplication        128
#define  iAbout                  1
#define mFile                    129
#define  iQuit                   12
#define mDemonstration           131
#define  iOffScreenGWorld1       1
#define  iOffScreenGWorld2       2
#define  iPicture                3
#define  iCursor                 4
#define  iAnimatedCursor1        5
#define  iAnimatedCursor2        6
#define  iAnimatedCursorOSX      7
#define  iIcon                   8
#define rBeachBallCursor         128
#define rPicture                 128
#define rTransformStrings        128
#define rIconFamily1             128
#define rIconFamily2             129
#define rColourIcon              128
```

```
#define rAboutDialog              128
#define kSleepTime                1
#define kBeachBallTickInterval    5
#define kCountingHandTickInterval 30
#define MAX_UINT32                0xFFFFFFFF
#define topLeft(r)                (((Point *) &(r))[0])
#define botRight(r)               (((Point *) &(r))[1])

// .................................................................................. typedefs

typedef struct
{
  SInt16     numberOfFrames;
  SInt16     whichFrame;
  CursHandle frame[];
} animCurs, *animCursPtr, **animCursHandle;

// .......................................................................... global variables

Boolean        gRunningOnX = false;
WindowRef      gWindowRef;
Boolean        gDone;
SInt32         gSleepTime;
RgnHandle      gCursorRegion;
Boolean        gCursorRegionsActive     = false;
Boolean        gAnimatedCursor1Active   = false;
Boolean        gAnimatedCursor2Active   = false;
Boolean        gAnimatedCursorOSXActive = false;
animCursHandle gAnimCursHdl;
SInt16         gAnimCursTickInterval;
SInt32         gAnimCursLastTick;
RGBColor       gBlackColour = { 0x0000, 0x0000, 0x0000 };
RGBColor       gWhiteColour = { 0xFFFF, 0xFFFF, 0xFFFF };
RGBColor       gBeigeColour = { 0xF000, 0xE300, 0xC200 };
RGBColor       gBlueColour  = { 0x4444, 0x4444, 0x9999 };

// ....................................................................... function prototypes

void    main                  (void);
void    doPreliminaries       (void);
OSErr   quitAppEventHandler   (AppleEvent *,AppleEvent *,SInt32);
void    eventLoop             (void);
void    doIdle                (void);
void    doEvents              (EventRecord *);
void    doMenuChoice          (SInt32);
void    doOffScreenGWorld1    (void);
void    doOffScreenGWorld2    (void);
void    doPicture             (void);
void    doCursor              (void);
void    doChangeCursor        (WindowRef,RgnHandle);
void    doAnimatedCursor1     (void);
void    doAnimatedCursor2     (void);
Boolean doGetAnimCursor       (SInt16,SInt16);
void    doIncrementAnimCursor (void);
void    doReleaseAnimCursor   (void);
void    doAnimatedCursorOSX   (void);
void    doIcon                (void);
void    doAboutDialog         (void);
void    doDrawStuff           (void);
UInt16  doRandomNumber        (UInt16,UInt16);

// ************************************************************************************ main

void  main(void)
{
  UInt32        seconds;
  MenuBarHandle menubarHdl;
  SInt32        response;
  MenuRef       menuRef;
  // ........................................................................................................ initialise managers

  doPreliminaries();

  // ......................................................................................... seed random number generator

  GetDateTime(&seconds);
  SetQDGlobalsRandomSeed(seconds);

  // ............................................................................................... set up menu bar and menus

  menubarHdl = GetNewMBar(rMenubar);
  if(menubarHdl == NULL)
    ExitToShell();
  SetMenuBar(menubarHdl);
  DrawMenuBar();

  Gestalt(gestaltMenuMgrAttr,&response);
  if(response & gestaltMenuMgrAquaLayoutMask)
  {
    menuRef = GetMenuRef(mFile);
    if(menuRef != NULL)
    {
      DeleteMenuItem(menuRef,iQuit);
      DeleteMenuItem(menuRef,iQuit - 1);
      DisableMenuItem(menuRef,0);
    }

    menuRef = GetMenuRef(mDemonstration);
    if(menuRef != NULL)
      EnableMenuItem(menuRef,iAnimatedCursorOSX);

    gRunningOnX = true;
  }

  // ........................................................................................................................ open window

  if(!(gWindowRef = GetNewCWindow(rWindow,NULL,(WindowRef)-1)))
    ExitToShell();

  SetPortWindowPort(gWindowRef);
  TextSize(10);

  // .............................................................................................................. enter event loop

  eventLoop();
}

// **************************************************************************** do preliminaries

void  doPreliminaries(void)
{
  OSErr osError;

  MoreMasterPointers(64);
  InitCursor();
  FlushEvents(everyEvent,0);

  osError = AEInstallEventHandler(kCoreEventClass,kAEQuitApplication,
                            NewAEEventHandlerUPP((AEEventHandlerProcPtr) quitAppEventHandler),
                            0L,false);
  if(osError != noErr)
    ExitToShell();
}

// ****************************************************************************** doQuitAppEvent

OSErr  quitAppEventHandler(AppleEvent *appEvent,AppleEvent *reply,SInt32 handlerRefcon)
```

```
{
  OSErr    osError;
  DescType returnedType;
  Size     actualSize;

  osError = AEGetAttributePtr(appEvent,keyMissedKeywordAttr,typeWildCard,&returnedType,NULL,0,
                              &actualSize);

  if(osError == errAEDescNotFound)
  {
    gDone = true;
    osError = noErr;
  }
  else if(osError == noErr)
    osError = errAEParamMissed;

  return osError;
}

// ********************************************************************************** eventLoop

void  eventLoop(void)
{
  EventRecord eventStructure;
  Boolean     gotEvent;

  gDone = false;
  gSleepTime = MAX_UINT32;
  gCursorRegion = NULL;

  while(!gDone)
  {
    gotEvent = WaitNextEvent(everyEvent,&eventStructure,gSleepTime,gCursorRegion);
    if(gotEvent)
      doEvents(&eventStructure);
    else
    {
      if(eventStructure.what == nullEvent)
        doIdle();
    }
  }
}

// ************************************************************************************* doIdle

void  doIdle(void)
{
  if(gAnimatedCursor1Active || gAnimatedCursor2Active)
    doIncrementAnimCursor();
}

// *********************************************************************************** doEvents

void  doEvents(EventRecord *eventStrucPtr)
{
  WindowRef      windowRef;
  WindowPartCode partCode;

  switch(eventStrucPtr->what)
  {
    case kHighLevelEvent:
      AEProcessAppleEvent(eventStrucPtr);
      break;

    case mouseDown:
      partCode = FindWindow(eventStrucPtr->where,&windowRef);
      switch(partCode)
      {
        case inMenuBar:
          doMenuChoice(MenuSelect(eventStrucPtr->where));
          break;

        case inContent:
          if(windowRef != FrontWindow())
            SelectWindow(windowRef);
          break;

        case inDrag:
          DragWindow(windowRef,eventStrucPtr->where,NULL);
          if(gCursorRegionsActive)
            doChangeCursor(windowRef,gCursorRegion);
          break;
      }
      break;

    case keyDown:
      if((eventStrucPtr->modifiers & cmdKey) != 0)
        doMenuChoice(MenuEvent(eventStrucPtr));
      break;

    case updateEvt:
      BeginUpdate((WindowRef) eventStrucPtr->message);
      EndUpdate((WindowRef) eventStrucPtr->message);
      break;

    case osEvt:
      switch((eventStrucPtr->message >> 24) & 0x000000FF)
      {
        case suspendResumeMessage:
          if((eventStrucPtr->message & resumeFlag) == 1)
            SetThemeCursor(kThemeArrowCursor);
          break;

        case mouseMovedMessage:
          if(gCursorRegionsActive)
            doChangeCursor(FrontWindow(),gCursorRegion);
          break;
      }
      break;
  }
}

// ************************************************************************** doMenuChoice

void  doMenuChoice(SInt32 menuChoice)
{
  MenuID        menuID;
  MenuItemIndex menuItem;

  menuID = HiWord(menuChoice);
  menuItem = LoWord(menuChoice);

  if(menuID == 0)
    return;

  if(gAnimatedCursor1Active || gAnimatedCursor2Active)
  {
    if(gAnimatedCursor2Active)
      doReleaseAnimCursor();

    SetThemeCursor(kThemeArrowCursor);
    gSleepTime = MAX_UINT32;

    gAnimatedCursor1Active = false;
    gAnimatedCursor2Active = false;
  }

  if(gAnimatedCursorOSXActive)
```

```
    doAnimatedCursorOSX();

  if(gCursorRegionsActive == true)
  {
    gCursorRegionsActive = false;
    DisposeRgn(gCursorRegion);
    gCursorRegion = NULL;
  }

  switch(menuID)
  {
    case mAppleApplication:
      if(menuItem == iAbout)
        doAboutDialog();
      break;

    case mFile:
      if(menuItem == iQuit)
        gDone = true;
      break;

    case mDemonstration:
      switch(menuItem)
      {
        case iOffScreenGWorld1:
          doOffScreenGWorld1();
          break;

        case iOffScreenGWorld2:
          doOffScreenGWorld2();
          break;

        case iPicture:
          doPicture();
          break;

        case iCursor:
          doCursor();
          break;

        case iAnimatedCursor1:
          doAnimatedCursor1();
          break;

        case iAnimatedCursor2:
          doAnimatedCursor2();
          break;

        case iAnimatedCursorOSX:
          doAnimatedCursorOSX();
          break;

        case iIcon:
          doIcon();
          break;
      }
      break;
  }

  HiliteMenu(0);
}

// ********************************************************************** doOffScreenGWorld1

void  doOffScreenGWorld1(void)
{
  Rect         portRect, sourceRect, destRect;
  GrafPtr      windowPortPtr;
  GDHandle     deviceHdl;
```

```
QDErr        qdErr;
GWorldPtr    gworldPortPtr;
PixMapHandle gworldPixMapHdl, windowPixMapHdl;
Boolean      lockPixResult;

// .................................................................................... draw in window

SetWTitle(gWindowRef,"\pTime-consuming drawing operation");

if(!gRunningOnX)
  SetThemeCursor(kThemeWatchCursor);

doDrawStuff();

if(!gRunningOnX)
  SetThemeCursor(kThemeArrowCursor);

SetWTitle(gWindowRef,"\pClick mouse to repeat in offscreen graphics port");
QDFlushPortBuffer(GetWindowPort(gWindowRef),NULL);

while(!Button()) ;

if(!gRunningOnX)
  SetThemeCursor(kThemeWatchCursor);

GetWindowPortBounds(gWindowRef,&portRect);

RGBBackColor(&gBlueColour);
EraseRect(&portRect);
RGBForeColor(&gWhiteColour);
MoveTo(190,180);
DrawString("\pPlease Wait.  Drawing in offscreen graphics port.");

// ........................................ draw in offscreen graphics port and copy to window

// ......................... save current graphics world and create offscreen graphics world

GetGWorld(&windowPortPtr,&deviceHdl);

qdErr = NewGWorld(&gworldPortPtr,0,&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

gworldPixMapHdl = GetGWorldPixMap(gworldPortPtr);
if(!(lockPixResult = LockPixels(gworldPixMapHdl)))
{
  SysBeep(10);
  return;
}

EraseRect(&portRect);

// ................................................. draw into the offscreen graphics port

doDrawStuff();

// ......................................................... restore saved graphics world

SetGWorld(windowPortPtr,deviceHdl);

// ................................................. set source and destination rectangles
```

```
  GetPortBounds(gworldPortPtr,&sourceRect);
  GetPortBounds(windowPortPtr,&destRect);

  // ............................................................ get window port's pixel map

  windowPixMapHdl = GetGWorldPixMap(windowPortPtr);

  // ............ ensure background colour is white and foreground colour in black, then copy

  RGBBackColor(&gWhiteColour);
  RGBForeColor(&gBlackColour);

  CopyBits((BitMap *) *gworldPixMapHdl,
           (BitMap *) *windowPixMapHdl,
           &sourceRect,&destRect,srcCopy,NULL);

  if(QDError() != noErr)
    SysBeep(10);

  // ............................................................................... clean up

  UnlockPixels(gworldPixMapHdl);
  DisposeGWorld(gworldPortPtr);

  if(!gRunningOnX)
    SetThemeCursor(kThemeArrowCursor);

  SetWTitle(gWindowRef,"\pOffscreen Graphics Worlds, Pictures,  Cursors and Icons");
  QDFlushPortBuffer(GetWindowPort(gWindowRef),NULL);
}

// ********************************************************************** doOffScreenGWorld2

void  doOffScreenGWorld2(void)
{
  PicHandle    picture1Hdl,picture2Hdl;
  Rect         portRect;
  Rect         sourceRect, maskRect, maskDisplayRect, dest1Rect, dest2Rect, destRect;
  GrafPtr      windowPortPtr;
  GDHandle     deviceHdl;
  QDErr        qdErr;
  GWorldPtr    gworldPortPtr;
  PixMapHandle gworldPixMapHdl, windowPixMapHdl;
  RgnHandle    region1Hdl, region2Hdl, regionHdl;
  SInt16       a, sourceMode;

  RGBBackColor(&gBeigeColour);
  GetWindowPortBounds(gWindowRef,&portRect);
  EraseRect(&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);
  DrawPicture(picture1Hdl,&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

gworldPixMapHdl = GetGWorldPixMap(gworldPortPtr);

if(!(LockPixels(gworldPixMapHdl)))
{
  SysBeep(10);
  return;
}

GetPortBounds(gworldPortPtr,&portRect);
EraseRect(&portRect);

// ........................................................................ get mask picture and draw it in offscreen graphics port

if(!(picture2Hdl = GetPicture(rPicture + 1)))
  ExitToShell();
HNoPurge((Handle) picture2Hdl);
DrawPicture(picture2Hdl,&maskRect);

// ............................................................................................................ also draw it in the window

SetGWorld(windowPortPtr,deviceHdl);
SetRect(&maskDisplayRect,329,35,485,146);
DrawPicture(picture2Hdl,&maskDisplayRect);
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(GetWindowFromPort(windowPortPtr),"\pClick mouse to copy");
QDFlushPortBuffer(GetWindowPort(gWindowRef),NULL);
while(!Button()) ;

// ............................................................ get window port's pixel map

windowPixMapHdl = GetGWorldPixMap(windowPortPtr);

// ................... 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((BitMap *) *windowPixMapHdl,
                 (BitMap *) *gworldPixMapHdl,
                 (BitMap *) *windowPixMapHdl,
                 &sourceRect,&maskRect,&destRect,sourceMode + ditherCopy,regionHdl);

    if(QDError() != noErr)
      SysBeep(10);
  }

  // ................................................................................ clean up

  UnlockPixels(gworldPixMapHdl);
  DisposeGWorld(gworldPortPtr);

  ReleaseResource((Handle) picture1Hdl);
  ReleaseResource((Handle) picture2Hdl);
  DisposeRgn(region1Hdl);
  DisposeRgn(region2Hdl);

  SetWTitle(gWindowRef,"\pOffscreen Graphics Worlds, Pictures,  Cursors and Icons");
  QDFlushPortBuffer(GetWindowPort(gWindowRef),NULL);
}

// ******************************************************************************** doPicture

void  doPicture(void)
{
  Rect           portRect, pictureRect, theRect;
  OpenCPicParams picParams;
  RgnHandle      oldClipRgn;
  PicHandle      pictureHdl;
  SInt16         a, left, top, right, bottom, random;
  RGBColor       theColour;
  PictInfo       pictInfo;
  Str255         theString;

  RGBBackColor(&gWhiteColour);
  GetWindowPortBounds(gWindowRef,&portRect);
  EraseRect(&portRect);

  // ................................................................ define picture rectangle

  pictureRect = portRect;
  pictureRect.right = (portRect.right - portRect.left) / 2;
  InsetRect(&pictureRect,10,10);

  // ..................................................................... set clipping region

  oldClipRgn = NewRgn();
  GetClip(oldClipRgn);
  ClipRect(&pictureRect);

  // ........................................................ set up OpenCPicParams structure

  picParams.srcRect  = pictureRect;
  picParams.hRes    = 0x00480000;
  picParams.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,theRect.right);
    top = doRandomNumber(10,theRect.bottom);
    right = doRandomNumber(left,theRect.right);
    bottom = doRandomNumber(top,theRect.bottom);
    SetRect(&theRect,left,top,right,bottom);

    PenMode(doRandomNumber(addOver,adMin));

    random = doRandomNumber(0,5);

    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("\pPICTURE");
    }
  }

  // ............................... stop recording, draw picture, restore saved clipping region

  ClosePicture();

  DrawPicture(pictureHdl,&pictureRect);

  SetClip(oldClipRgn);
  DisposeRgn(oldClipRgn);

  // ...................................... display some information from the PictInfo structure

  RGBForeColor(&gBlueColour);
  RGBBackColor(&gBeigeColour);
  PenMode(patCopy);
  OffsetRect(&pictureRect,300,0);
  EraseRect(&pictureRect);
  FrameRect(&pictureRect);
  TextSize(10);
```

```
if(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((long) 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);

  QDFlushPortBuffer(GetWindowPort(gWindowRef),NULL);

  // ............................................................ release memory occupied by Picture structure

  KillPicture(pictureHdl);
}

// ********************************************************************************** doCursor

void  doCursor(void)
{
  Rect   portRect, cursorRect;
  SInt16 a;

  RGBBackColor(&gBlueColour);
  GetWindowPortBounds(gWindowRef,&portRect);
  EraseRect(&portRect);

  cursorRect = portRect;

  for(a=0;a<3;a++)
  {
    InsetRect(&cursorRect,40,40);

    if(a == 0 || a == 2)
      RGBBackColor(&gBeigeColour);
    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");

  gCursorRegion = NewRgn();
  doChangeCursor(gWindowRef,gCursorRegion);

  gCursorRegionsActive = true;
}

// **************************************************************************** doChangeCursor

void  doChangeCursor(WindowRef windowRef,RgnHandle cursorRegion)
{
  RgnHandle arrowCursorRgn;
  RgnHandle ibeamCursorRgn;
  RgnHandle crossCursorRgn;
  RgnHandle plusCursorRgn;
  Rect      cursorRect;
  GrafPtr   oldPort;
  Point     mousePosition;
  arrowCursorRgn = NewRgn();
  ibeamCursorRgn = NewRgn();
  crossCursorRgn = NewRgn();
  plusCursorRgn   = NewRgn();

  SetRectRgn(arrowCursorRgn,-32768,-32768,32766,32766);

  GetPort(&oldPort);
  SetPortWindowPort(windowRef);

  GetWindowPortBounds(windowRef,&cursorRect);
  LocalToGlobal(&topLeft(cursorRect));
  LocalToGlobal(&botRight(cursorRect));

  InsetRect(&cursorRect,40,40);
  RectRgn(ibeamCursorRgn,&cursorRect);
  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);

  GetGlobalMouse(&mousePosition);

  if(PtInRgn(mousePosition,ibeamCursorRgn))
  {
    SetThemeCursor(kThemeIBeamCursor);
    CopyRgn(ibeamCursorRgn,cursorRegion);
  }
  else if(PtInRgn(mousePosition,crossCursorRgn))
  {
    SetThemeCursor(kThemeCrossCursor);
    CopyRgn(crossCursorRgn,cursorRegion);
  }
  else if(PtInRgn(mousePosition,plusCursorRgn))
  {
    SetThemeCursor(kThemePlusCursor);
    CopyRgn(plusCursorRgn,cursorRegion);
  }
  else
  {
    SetThemeCursor(kThemeArrowCursor);
    CopyRgn(arrowCursorRgn,cursorRegion);
  }

  DisposeRgn(arrowCursorRgn);
  DisposeRgn(ibeamCursorRgn);
  DisposeRgn(crossCursorRgn);
  DisposeRgn(plusCursorRgn);

  SetPort(oldPort);
}

// ************************************************************************ doAnimatedCursor1

void  doAnimatedCursor1(void)
{
  Rect    portRect;
  Pattern whitePattern;

  BackColor(whiteColor);
  GetWindowPortBounds(gWindowRef,&portRect);
  FillRect(&portRect,GetQDGlobalsWhite(&whitePattern));
  gAnimCursTickInterval = kCountingHandTickInterval;
  gSleepTime = gAnimCursTickInterval;
  gAnimatedCursor1Active = true;

}

// ************************************************************************ doAnimatedCursor2

void  doAnimatedCursor2(void)
{
  Rect    portRect;
  Pattern whitePattern;
  SInt16  animCursResourceID, animCursTickInterval;

  BackColor(whiteColor);
  GetWindowPortBounds(gWindowRef,&portRect);
  FillRect(&portRect,GetQDGlobalsWhite(&whitePattern));

  animCursResourceID   = rBeachBallCursor;
  animCursTickInterval = kBeachBallTickInterval;

  if(doGetAnimCursor(animCursResourceID,animCursTickInterval))
  {
    gSleepTime = animCursTickInterval;
    gAnimatedCursor2Active = true;
  }
  else
    SysBeep(10);
}

// ************************************************************************** doGetAnimCursor

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) HiWord((SInt32) (*gAnimCursHdl)->frame[a]);
      (*gAnimCursHdl)->frame[a] = GetCursor(cursorID);
      if((*gAnimCursHdl)->frame[a])
        a++;
      else
        noError = false;
    }
  }

  if(noError)
  {
    gAnimCursTickInterval = tickInterval;
    gAnimCursLastTick = TickCount();
    (*gAnimCursHdl)->whichFrame = 0;
  }

  return noError;
}

// ******************************************************************** doIncrementAnimCursor

void  doIncrementAnimCursor(void)
{
  SInt32        newTick;
  static UInt32 animationStep;

  newTick = TickCount();
  if(newTick < (gAnimCursLastTick + gAnimCursTickInterval))
    return;

  if(gAnimatedCursor1Active)
  {
    SetAnimatedThemeCursor(kThemeCountingUpAndDownHandCursor,animationStep);
      animationStep++;
  }
  else if(gAnimatedCursor2Active)
  {
    SetCursor(*((*gAnimCursHdl)->frame[(*gAnimCursHdl)->whichFrame++]));
    if((*gAnimCursHdl)->whichFrame == (*gAnimCursHdl)->numberOfFrames)
      (*gAnimCursHdl)->whichFrame = 0;
  }

  gAnimCursLastTick = newTick;
}

// ********************************************************************** doReleaseAnimCursor

void  doReleaseAnimCursor(void)
{
  SInt16 a;

  for(a=0;a<(*gAnimCursHdl)->numberOfFrames;a++)
    ReleaseResource((Handle) (*gAnimCursHdl)->frame[a]);

  ReleaseResource((Handle) gAnimCursHdl);
}

// ********************************************************************** doAnimatedCursorOSX

void  doAnimatedCursorOSX(void)
{
  if(!gAnimatedCursorOSXActive)
  {
    QDDisplayWaitCursor(true);
    gAnimatedCursorOSXActive = true;
  }
  else
  {
    QDDisplayWaitCursor(false);
    gAnimatedCursorOSXActive = false;
  }
}

// ************************************************************************************* doIcon

void  doIcon(void)
{
  Rect              portRect, theRect;
  SInt16            a, b, stringIndex = 1;
  IconTransformType transform = 0;
  Str255            theString;
  Handle            iconSuiteHdl;
  CIconHandle       ciconHdl;

  RGBForeColor(&gBlueColour);
  RGBBackColor(&gBeigeColour);
  GetWindowPortBounds(gWindowRef,&portRect);
  EraseRect(&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);
  LineTo(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");

  ciconHdl = GetCIcon(rColourIcon);

  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);
}

// *************************************************************************** doAboutDialog

void  doAboutDialog(void)
{
  DialogPtr dialogPtr;
  SInt16    itemHit;

  dialogPtr = GetNewDialog(rAboutDialog,NULL,(WindowRef)-1);
  ModalDialog(NULL,&itemHit);
  DisposeDialog(dialogPtr);
```

```
}

// ******************************************************************************* doDrawStuff

void  doDrawStuff(void)
{
  Rect     portRect, theRect;
  RGBColor theColour;
  SInt16   a, left, top, right, bottom, random;

  RGBBackColor(&gBlueColour);
  GetWindowPortBounds(gWindowRef,&portRect);
  EraseRect(&portRect);

  for(a=0;a<900;a++)
  {
    theRect = portRect;

    theColour.red   = doRandomNumber(0,65535);
    theColour.green = doRandomNumber(0,65535);
    theColour.blue  = doRandomNumber(0,65535);
    RGBForeColor(&theColour);

    left = doRandomNumber(0,theRect.right);
    top = doRandomNumber(0,theRect.bottom);
    right = doRandomNumber(left,theRect.right);
    bottom = doRandomNumber(top,theRect.bottom);
    SetRect(&theRect,left,top,right,bottom);

    PenMode(doRandomNumber(addOver,adMin));

    random = doRandomNumber(0,3);

    if(random == 0)
      PaintRect(&theRect);
    else if(random == 1)
      PaintRoundRect(&theRect,doRandomNumber(10,100),doRandomNumber(10,100));
    else if(random == 2)
      PaintOval(&theRect);
    else if(random == 3)
      PaintArc(&theRect,0,doRandomNumber(5,330));

    QDFlushPortBuffer(GetWindowPort(gWindowRef),NULL);
  }
}

// **************************************************************************** doRandomNumber

UInt16  doRandomNumber(UInt16 minimum, UInt16 maximum)
{
  UInt16 randomNumber;
  SInt32 range, t;

  randomNumber = Random();
  range = maximum - minimum + 1;
  t = (randomNumber * range) / 65536;
  return (t + minimum);
}

// *******************************************************************************************
```

Demonstration Program GWorldPicCursIcn Comments

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 animated cursor demonstrations have been invoked.
- Choose the About… item in the Apple menu to display the About… dialog.
- Note that the Icons item in the Demonstration menu contains an icon.

On Mac OS 8/9, if the first offscreen graphics world demonstration does not work when the monitor colour depth is set to Millions, increase the Minimum Heap Size set in the CodeWarrior project.

defines

Constants are established for the resource IDs of 'acur', 'PICT', 'STR#', icon family, and 'cicn' resources, and a 'DLOG' resource.

kSleeptime and MAX_UINT32 will be assigned to WaitNextEvent's sleep parameter at various points in the program. kBeachBallTickInterval represents the interval between frame changes for the first of three animated cursors. kCountingHandTickInterval represents the interval between frame changes for the second animated cursor.

typedefs

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

Global Variables

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

gAnimCursHdl will be assigned a handle to the animCurs structure used during the first animated cursor demonstration. gAnimCursTickInterval and gAnimCursLastTick also relate to the animated cursor demonstration.

main

Random numbers will be used in the function doPicture. The call to SetQDGlobalsRandomSeed seeds the random number generator with the value returned by the call to GetDateTime.

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().

eventLoop

Before the event loop is entered, gSleepTime is set to MAX_UINT32. Initially, therefore, the sleep parameter in the WaitNextEvent call is set to the maximum possible UInt32 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.

When WaitNextEvent returns 0 with a null event, the function doIdle is called.

doIdle

doIdle is called from the main event loop when WaitNextEvent returns 0 with a null event. If the active demonstration is one of the first two animated cursor demonstrations, the function doIncrementAnimCursor is called.

doEvents

In the inDrag case, after the call to DragWindow, and provided the cursor shape changing demonstration is currently under way, the 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 redefined. The call to doChangeCursor re-defines these regions for the new window location and copies the reference 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, SetThemeCursor is called to ensure that the cursor is set to the arrow shape.

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

doMenuChoice

The purpose of the code prior to the switch is to cancel any cursor demonstrations that may be currently under way.

If the second of the first two animated cursor demonstrations is currently under way, doReleaseAnimCursor is called to release the memory associated with that animated cursor.

If either of the first two animated cursor demonstrations is currently under way, SetThemeCursor is called to set the cursor shape to the arrow shape and WaitNextEvent's sleep parameter is then set to the maximum possible value.

if the third animated cursor demonstration is currently under way, the function doAnimatedCursorOSX is called to terminate the Mac OS X wait cursor.

If the cursor shape changing demonstration is currently under way, gCursorRegionsActive is set to false, 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.

Note that, if the user chooses the About... item in the Mac OS 8/9Apple or Mac OS X Application menu, doAboutDialog is called.

doOffScreenGWorld1

doWithoutOffScreenGWorld is the first demonstration.

Draw in Window

As a prelude for what is to come, the function doDrawStuff is called to repeatedly paint some shapes in the window.

Draw in Offscreen Graphics Port and Copy to Window

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

The call to NewGWorld creates an offscreen graphics world. The gworldPortPtr parameter receives a pointer to the offscreen graphics world's graphics port. 0 in the pixelDepth parameter means that the offscreen world's pixel depth will be set to the deepest device intersecting the rectangle passed as the boundsRect parameter. This rectangle becomes the offscreen port's port port rectangle, the offscreen pixel map's bounding rectangle, and the offscreen device's bounding rectangle. 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. 0 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 is called to prevent the base address of the pixel image from being moved when the pixel image is drawn into or copied from.

The call to EraseRect clears the offscreen graphics port before the 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 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.

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 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 references 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 are 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 allocated for 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 of 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 function doChangeCursor to be called from within the main event loop provided the application is not in the background. In addition, it draws 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 reference to a cursor region will be copied to gCursorRegion in the 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 five 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 GetGlobalMouse gets the point, in global coordinates, representing the mouse's current position.

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 reference 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.

doAnimatedCursor1

doAnimatedCursor1 responds to the user's selection of the Animated Cursor 1 item in the Demonstration menu.

In this first animated cursor demonstration, the Appearance Manager function SetAnimatedThemeCursor will be used in the function doIncrementAnimCursor to increment the cursor frame. As preparatory measures, an appropriate frame change tick interval is assigned to gAnimCursTickInterval, the sleep parameter in the WaitNextEvent call is set to the same value (causing null events to be generated at that tick interval), and gAnimCurs1Active is set to true so that doIncrementAnimCursor will be called from the doIdle function.

doAnimatedCursor2

doAnimatedCursor2 responds to the user's selection of the Animated Cursor 2 item in the Demonstration menu.

In this second animated cursor demonstration, functions are utilised to retrieve 'acur' and 'CURS' resources, animate the cursor, and deallocate the memory associated with the animated cursor when the cursor is no longer required. DoAnimatedCursor2's major role is simply to call doGetAnimCursor with a "beach-ball" 'acur' resource as a parameter.

After the screen has been cleared, the resource ID of the "beach-ball" 'acur' resource is assigned to the variable used as the first parameter in the later call to doGetAnimCursor. The next line assigns a value to the second parameter in the doGetAnimCursor 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 doGetAnimCursor is successful, the sleep parameter in the WaitNextEvent call is set to the same ticks value as that used to control the cursor's frame rate (causing null events to be generated at that tick interval), and the flag gAnimCurs2Active is set to true so that doIncrementAnimCursor will be called from the doIdle function.

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

doGetAnimCursor

doGetAnimCursor 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 reference s 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. 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.

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

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

doIncrementAnimCursor

doIncrementAnimCursor is called whenever null events are received.

The first line assigns the number of ticks since system startup to newTick. The next line checks whether the specified number of ticks have elapsed since the previous call to doIncrementAnimCursor. If the specified number of ticks have not elapsed, the function simply returns. Otherwise, the following occurs:

- If the first animated cursor demonstration is under way, the Appearance Manager function SetThemeAnimatedCursor is called to increment the theme-compliant cursor frame.

- If the second animated cursor demonstration is under way, 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 gAnimatedCursor1Active and gAnimatedCursor2Active flags are set to false, the cursor is reset to the standard arrow shape, and WaitNextEvent's sleep parameter is reset to the maximum possible value.

doAnimatedCursorOSX

doAnimatedCursorOSX utilises the function QDDisplayWaitCursor to turn the Mac OS X "spinning wheel" cursor on and off. This function is only available on OS X, so the stub library CarbonFrameworkLib has been added to the CodeWarrior project.

doIcon

doIcon demonstrates the drawing of icons in a window using PlotIconID, PlotIconSuite, 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.

GetCIcon and PlotCIcon

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.

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