13

OFFSCREEN GRAPHICS WORLDS, PICTURES, CURSORS, AND ICONS

Includes Demonstration Program GWorldPicCursIcon

Offscreen Graphics Worlds

Introduction

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

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

Creating an Offscreen Graphics World

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

```
type
GWorldPtr = CGrafPtr;
```

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

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

Setting the Colour Graphics Port for an Offscreen Graphics World

Before drawing into the offscreen graphics port, you should save the current colour graphics port and the current device's 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 colour graphics port as the current colour 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 attached to the offscreen graphics world.

Preparing to Draw Into an Offscreen Graphics World

After setting the offscreen graphics world as the current port, you should use the 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. If the base address for an offscreen pixel image has not been purged by the Memory Manager, or if its base address is not purgeable, LockPixels returns true. If LockPixels returns false, your application should either call the UpdateGWorld function to reallocate the offscreen pixel image and then reconstruct it, or draw directly into an onscreen graphics port.

As a related matter, note that the 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. You must use the GetPixBaseAddr function 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 restore the active window as the current graphics port.

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

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

Updating an Offscreen Graphics World

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

Disposing of an Offscreen Graphics World

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

Pictures

Introduction

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

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

Picture Format

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

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

The Picture Structure

The Picture structure is as follows:

```
Picture = RECORD
   picSize:  INTEGER;       { For a version 1 picture: its size. }
   picFrame:                Rect;        { Bounding rectangle for the picture. }
   ...
   END;

PicPtr = ^Picture;
PicHandle = ^PicPtr;
```

Field Descriptions

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

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

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

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

Opcodes: Drawing Commands and Picture Comments

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

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

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

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

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

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

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

The Picture Utilities

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

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

[1] See Chapter 18 — Scrap.

Creating Pictures

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

You pass information to OpenCPicture in the form of an OpenCPicParams structure:

```
OpenCPicParams = RECORD
   srcRect: Rect;              { Optimal bounding rectangle. }
   hRes:                 Fixed;        { Best horizontal resolution. }
   vRes:                 Fixed;        { Best vertical resolution. }
   version: INTEGER;     { Set to -2. }
   reserved1:            INTEGER;      { (Reserved.  Set to 0.) }
   reserved2:            LONGINT;      { (Reserved.  Set to 0.) }
   END;
```

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

Clipping Region

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

Opening and Drawing Pictures

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

When the picture is retrieved, you should call DrawPicture to draw the picture. The second parameter taken by DrawPicture 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

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

[2] The demonstration program at Chapter 16 — Files shows how to read pictures from, and save pictures to, files of type 'PICT'.

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 256-pixel image in a 16-by-16 pixel square defined in a black-and-white cursor ('CURS') or colour cursor ('crsr') resource.

Cursor Movement, Hot Spot, Visibility, and Shape

Cursor Movement

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

Cursor Hot Spot

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

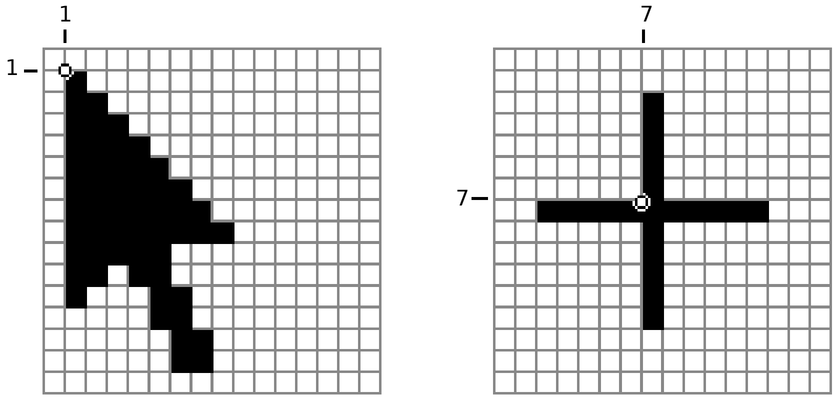

FIG 1 - HOT SPOTS IN CURSORS

Cursor Visibility

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

Cursor Shape

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

- To indicate that the user is over a certain area of the screen. For example, when the cursor is in the menu bar, it should usually have an arrow shape. When the user moves the cursor over a text document, your application should change the cursor to the I-beam shape.

- To provide feedback about the status of the computer system. For example, if an operation will take a second or two, you should provide feedback to the user by changing the cursor to the wristwatch cursor (see Fig 2). If the operation takes several seconds and the user can do nothing in your application but stop the operation, wait until it is completed, or switch to another application, you should display an animated cursor (see below).[3]

Non-Animated Cursors

System 'CURS' and 'crsr' Resources

The System file in the System Folder contains an number of 'CURS' 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 WRIS

The Mac OS 8.0 and later System file contains additional 'CURS' resources. The Mac OS 8.5 and later System file contains three 'crsr' resources. The following are the resource IDs for the additional cursors as shown as Fig 3:

Constant	Value	Description
-	-20488	Contextual menu arrow cursor.
-	-20487	Alias arrow cursor.
-	-20486	Copy arrow cursor.
-	-20452	Resize left cursor.
-	-20451	Resize right cursor.
-	-20450	Resize left/right cursor.
-	-20877	Pointing hand cursor.
-	-20876	Open hand pointer.
-	-20875	Close hand pointer.

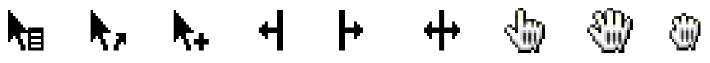

FIG 3 - ADDITIONAL CURSOR AND COLOUR CURS

Custom 'CURS' and 'crsr' Resources

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

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

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.

To change cursor shape, your application must get a handle to the relevant cursor (either a custom cursor or one of the system cursors shown at 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.

Changing Cursor Shape — Appearance-Compliant Methodology

Mac OS 8.5 (or, more specifically, Appearance Manager Version 1.1) introduced a new function (SetThemeCursor) for setting the cursor to a version of the specified cursor type that is consistent with the current appearance. You must pass one of the following constants, which are of type ThemeCursor, in the inCursor parameter of 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, for appearance-compliant cursors, SetThemeCursor, to change the cursor to the appropriate shape.[4]

[4] Note that your application may also have to accommodate the cursor shape changing requirements of modeless dialog boxes with edit text field items.as well as its main windows.

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

Appearance-Compliant Methodology

Mac OS 8.5 (or, more specifically, Appearance Manager Version 1.1) introduced a new function (SetThemeAnimatedCursor) for animating a version of the specified cursor type that is consistent with the current appearance. 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

Non-Appearance-Compliant Methodology

Non-appearance-compliant animated cursors require: 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 Mac OS 8.0 and later System file 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.

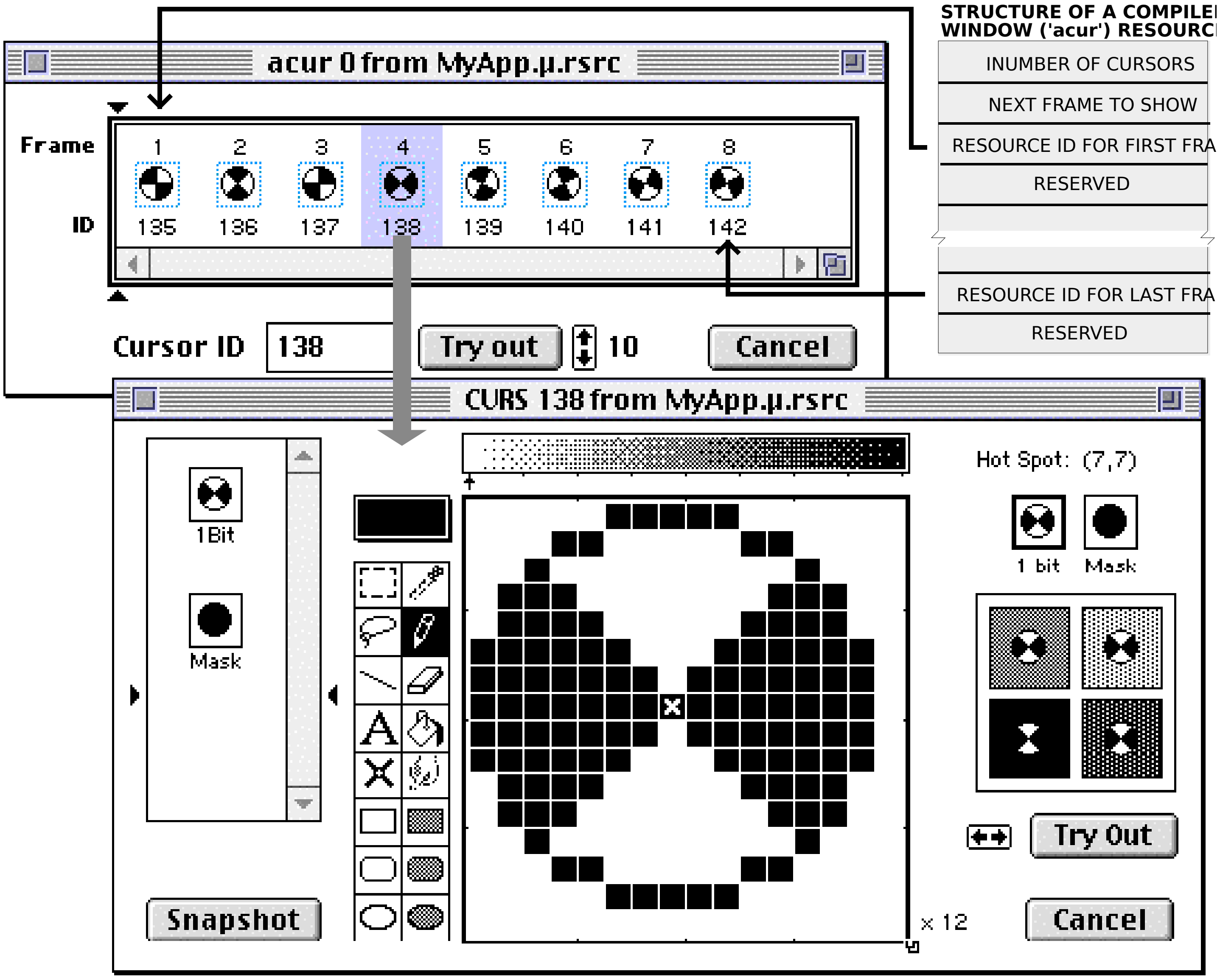

FIG 4 - CREATING AN 'acur' RESOURCE AND ASSOCIATED 'CURS' RESOURCES USING RESOR

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 an application-defined 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.

Icons

Icons and the Finder

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

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

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

Creating Icon Family Resources Using Resorcerer

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

The 'icns' Resource

The 'icns' resource contains all of the data for four icon sizes, thus providing a single source for icon data as opposed to the collection of icon resources ('ics#', 'icl4', 'icm8', etc.) described above. This speeds up icon fetching and simplifies resource management.

Historical Note

The 'icns' resource was introduced with Mac OS 8.5.

The four icon sizes are mini, small, large, and huge, the latter being a new size of 48 by 48 pixels. Four colour depths (1-bit, 4-bit, 8-bit, and 32-bit) and two kind of masks (1-bit and 8-bit) are supported. The deep (8 bit) mask means that masks can have 256 different levels of transparency.

Icon Services checks for an 'icns' resource of the specified ID before it checks for the older resource types ('ics#', 'icl4', 'icm8', etc.) of the same ID. If an 'icns' resource is found, Icon Services obtains all icon data exclusively from that resource.

As of Version 2.2, Resorcerer had no pixel editor for 'icns' resources. However, a command available in the Icon Family editor will take all icons currently being shown and build an 'icns' resource with the same resource ID as those icons. (Choose Build/Update 'icns' from the IconFamily menu.)

Other Icons — Icons, Colour Icons and Small Icons

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

Icon ('ICON')

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

Colour Icon ('cicn')

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

Small Icon ('SICN')

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

Icons in Windows, Menus, and Alert and Dialog Boxes

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

Icons in Windows

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

Icons in Menus

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

- Create the icon resource with a resource ID between 257 and 511. Subtract 256 from the resource ID to get a value called the **icon number**. Specify the icon number in the Icon field of the menu item definition.

- For an icon ('ICON'), specify 0x1D in the keyboard equivalent field of the menu item definition to indicate to the Menu Manager that the icon should be reduced to fit into a 16-by-16 pixel rectangle. Otherwise, specify a value of $00, or a value greater than $20, 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 $00 or a value greater than $20 in the keyboard equivalent field. The Menu Manager automatically enlarges the enclosing rectangle of the menu item according to the rectangle specified in the 'cicn'

[5] Typically, only the Finder and the Standard File Package use small icons.

resource. (Colour icons, unlike icons, can be any height or width between 8 and 64 pixels.)

- For a small icon ('SICN'), specify $1E in the keyboard equivalent field. This indicates that the item has an icon defined by a 'SICN' resource. The Menu Manager plots the icon in a 16-by-16 pixel rectangle.

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

Icons in Alert and Dialog Boxes

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

To display the icon alone, the procedure is to define an item of type 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 box using Dialog Manager functions.

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

If you provide a colour icon ('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.

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

Drawing and Manipulating Icons

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

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

You need to use Icon Utilities functions only if:

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

Preamble - Icon Families, Suites, and Caches

Icon Families

You can define individual icons of resource types 'ICON', 'cicn', and 'SICN' that are not part of an icon family and use Icon Utilities functions to draw them as required. However, to display an icon effectively at a variety of sizes and bit depths, you should provide an icon family[7] in the same way that you provide icon families for the Finder. The advantage of providing an icon family is that you can then leave it to functions such as PlotIconID, which are used to draw icons, to automatically determine which icon in the icon family is best suited to the specified destination rectangle and current display bit depth.

Icon Suites

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

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

Icon Cache

An **icon cache** is like an icon suite except that it also contains a pointer to an application-defined **icon getter function** and a pointer to data that is associated with the icon suite. You can pass a handle to an icon cache to any of the Icon Utilities functions which accept a handle to an icon suite. An icon cache typically does not contain handles to the icon resources for all icon family members. Instead, if the icon cache does not contain an entry for a specific type of icon in an icon family, the Icon Utilities functions call your application's icon getter function to retrieve the data for that icon type.

Drawing an Icon Directly From a Resource

To draw an icon from an icon family without first creating an icon suite, use the PlotIconID function. PlotIconID determines, from the size of the specified destination rectangle and the current bit depth of the display device, which icon to draw. The icon drawn is as follows:

[7] Each icon in an icon family shares the same resource ID as other icons in the family but has its own resource type identifying the icon data it contains.

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

Icon Stretching and Shrinking

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

Icon Alignment and Transform

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

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 which contains only the icon of the desired resource type together with its corresponding mask. Constants such as kSelectorLarge4Bit (an icon selector mask for an '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 dialog boxes, and icons of resource type 'SICN' in menu items, you use Menu Manager and Dialog Manager functions such as SetItemIcon and SetDialogItem.

To draw resources of resource type 'ICON', 'cicn', and 'SICN' 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
'SICN'	GetResource	PlotSICNHandle

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

```
GWorldPtr = CGrafPtr;
GWorldFlags = UInt32;
```

Functions

Creating, Altering, and Disposing of Offscreen Graphics Worlds

```
FUNCTION  NewGWorld(VAR offscreenGWorld: GWorldPtr; PixelDepth: INTEGER;
                {CONST}VAR boundsRect: Rect; cTable: CTabHandle; aGDevice: GDHandle;
                flags: GWorldFlags): QDErr;
FUNCTION  UpdateGWorld(VAR offscreenGWorld: GWorldPtr; pixelDepth: INTEGER;
                CONST}VAR boundsRect: Rect; cTable: CTabHandle; aGDevice: GDHandle;
                flags: GWorldFlags): GWorldFlags;
PROCEDURE DisposeGWorld(offscreenGWorld: GWorldPtr);
```

Saving and Restoring Graphics Ports and Offscreen Graphics Worlds

```
PROCEDURE GetGWorld(VAR port: CGrafPtr; VAR gdh: GDHandle);
PROCEDURE SetGWorld(port: CGrafPtr; gdh: GDHandle);
```

Managing an Offscreen Graphics World's Pixel Image

```
FUNCTION  GetGWorldPixMap(offscreenGWorld: GWorldPtr): PixMapHandle;
FUNCTION  LockPixels(pm: PixMapHandle): BOOLEAN;
PROCEDURE UnlockPixels(pm: PixMapHandle);
PROCEDURE AllowPurgePixels(pm: PixMapHandle);
PROCEDURE NoPurgePixels(pm: PixMapHandle);
FUNCTION  GetPixelsState(pm: PixMapHandle): GWorldFlags;
PROCEDURE SetPixelsState(pm: PixMapHandle; state: GWorldFlags);
FUNCTION  GetPixBaseAddr(pm: PixMapHandle): Ptr;
FUNCTION  PixMap32Bit(pmHandle: PixMapHandle): BOOLEAN;
```

Main Constants, Data Types and Functions — Pictures

Constants

Verbs for the GetPictInfo, GetPixMapInfo, and NewPictInfo calls

returnColorTable	= $0001	Return a ColorTable structure.
returnPalette	= $0002	Return a Palette structure.
recordComments	= $0004	Return comment information.
recordFontInfo	= $0008	Return font information.
suppressBlackAndWhite	= $0010	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

```
Picture = RECORD
   picSize:   INTEGER;            { For a version 1 picture: its size. }
   picFrame:                    Rect;            { Bounding rectangle for the picture. }
   END;

PicPtr = ^Picture;
PicHandle = ^PicPtr;
```

OpenCPicParams

```
OpenCPicParams = RECORD
   srcRect:   Rect;                   { Optimal bounding rectangle. }
   hRes:                        Fixed;        { Best horizontal resolution. }
   vRes:                        Fixed;        { Best vertical resolution. }
   version:   INTEGER;            { Set to -2. }
   reserved1:                   INTEGER;   { (Reserved.  Set to 0.) }
   reserved2:                   LONGINT;      { (Reserved.  Set to 0.) }
   END;
```

PictInfo

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

CommentSpec

```
CommentSpec = RECORD
    count:                  INTEGER;                { Number of occurrances of this comment ID. }
    ID:                     INTEGER;                { ID for the comment in the picture. }
    END;

CommentSpecPtr = ^CommentSpec;
CommentSpecHandle = ^CommentSpecPtr;
```

FontSpec

```
FontSpec = RECORD
    pictFontID:         INTEGER;                { ID of the font in the picture. }
    sysFontID:          INTEGER;                { ID of the same font in the current system file. }
    size:               ARRAY [0..3] OF LONGINT;  {  bit array of all the sizes found (1..127)
                                                         (bit 0 means > 127). }
    style:              INTEGER;                { Combined style of all occurrances of the font. }
    nameOffset:         LONGINT;                { Offset into the fontNamesHdl handle for the font name. }
    END;
FontSpecPtr = ^FontSpec;
FontSpecHandle = ^FontSpecPtr;
```

Functions

Creating and Disposing of Pictures

```
FUNCTION  OpenCPicture({CONST}VAR newHeader: OpenCPicParams): PicHandle;
PROCEDURE PicComment(kind: INTEGER; dataSize: INTEGER; dataHandle: Handle);
PROCEDURE ClosePicture;
PROCEDURE KillPicture(myPicture: PicHandle);
```

Drawing Pictures

```
PROCEDURE DrawPicture(myPicture: PicHandle; {CONST}VAR dstRect: Rect);
FUNCTION  GetPicture(pictureID: INTEGER): PicHandle;
```

Collecting Picture Information

```
FUNCTION  GetPictInfo(thePictHandle: PicHandle; VAR thePictInfo: PictInfo; verb: INTEGER; colorsRequested: INTEGER;
              colorPickMethod: INTEGER; version: INTEGER): OSErr;
FUNCTION  GetPixMapInfo(thePixMapHandle: PixMapHandle; VAR thePictInfo: PictInfo; verb: INTEGER; colorsRequested:
              INTEGER; colorPickMethod: INTEGER; version: INTEGER): OSErr;
FUNCTION  NewPictInfo(VAR thePictInfoID: PictInfoID; verb: INTEGER; colorsRequested: INTEGER; colorPickMethod:
              INTEGER; version: INTEGER): OSErr;
FUNCTION  RecordPictInfo(thePictInfoID: PictInfoID; thePictHandle: PicHandle): OSErr;
FUNCTION  RecordPixMapInfo(thePictInfoID: PictInfoID; thePixMapHandle: PixMapHandle): OSErr;
FUNCTION  RetrievePictInfo(thePictInfoID: PictInfoID; VAR thePictInfo: PictInfo; colorsRequested: INTEGER): OSErr;
FUNCTION  DisposePictInfo(thePictInfoID: PictInfoID): OSErr;
```

Main Constants, Data Types and Functions — Cursors

Constants

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

Data Types

Cursor

```
Cursor = RECORD
    data:              Bits16;
    mask:              Bits16;
    hotSpot:           Point;
    END;

CursorPtr = ^Cursor;
CursPtr = ^Cursor;
```

```
CursHandle = ^CursPtr;
```

CCrsr

```
    CCrsr = RECORD
    crsrType:       INTEGER;        { Type of cursor. }
    crsrMap:        PixMapHandle;   { The cursor's pixmap. }
    crsrData:       Handle;         { Cursor's data. }
    crsrXData:      Handle;         { Expanded cursor data. }
    crsrXValid:     INTEGER;        { Depth of expanded data (0 if none). }
    crsrXHandle:    Handle;         { Future use. }
    crsr1Data:      Bits16;         { One-bit cursor. }
    crsrMask:       Bits16;         { Cursor's mask. }
    crsrHotSpot:    Point;          { Cursor's hotspot. }
    crsrXTable:     LONGINT;        { Private. }
    crsrID:         LONGINT;        { Private. }
    END;

CCrsrPtr = ^CCrsr;
CCrsrHandle = ^CCrsrPtr;
```

Acur

```
Acur = RECORD
    n:              INTEGER;        { Number of cursors ("frames of film"). }
    index:          INTEGER;        { Next frame to show <for internal use>. }
    frame1:         INTEGER;        { 'CURS' resource id for frame #1. }
    fill1:          INTEGER;        { <for internal use>. }
    frame2:         INTEGER;        { 'CURS' resource id for frame #2. }
    fill2:          INTEGER;        { <for internal use>. }
    frameN:         INTEGER;        { 'CURS' resource id for frame #N. }
    fillN:          INTEGER;        { <for internal use>. }
    END;
acurPtr = ^Acur;
acurHandle = ^acurPtr;
```

Functions

Initialising Cursors

```
PROCEDURE InitCursor;
PROCEDURE InitCursorCtl(newCursors: UNIV acurHandle);
```

Changing Black-and-White Cursors

```
FUNCTION  GetCursor(cursorID: INTEGER): CursHandle;
PROCEDURE SetCursor({CONST}VAR crsr: Cursor);
```

Changing Colour Cursors

```
FUNCTION  GetCCursor(crsrID: INTEGER): CCrsrHandle;
PROCEDURE SetCCursor(cCrsr: CCrsrHandle);
PROCEDURE AllocCursor;
PROCEDURE DisposeCCursor(cCrsr: CCrsrHandle);
PROCEDURE DisposCCursor(cCrsr: CCrsrHandle);
```

Hiding, Showing , and Animating Cursors

```
PROCEDURE HideCursor;
PROCEDURE ShowCursor;
PROCEDURE ObscureCursor;
PROCEDURE ShieldCursor({CONST}VAR shieldRect: Rect; offsetPt: Point);
PROCEDURE RotateCursor(counter: LONGINT);
PROCEDURE SpinCursor(increment: INTEGER);
```

Appearance Manager Constants, Data Types and Functions — Cursors

The following constants, data types, and functions were introduced with Mac OS 8.5.

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

```
ThemeCursor = UInt32;
```

Functions

```
FUNCTION  SetThemeCursor(inCursor: ThemeCursor): OSStatus;
FUNCTION  SetAnimatedThemeCursor(inCursor: ThemeCursor; inAnimationStep: UInt32): OSStatus;
```

Main Constants, Data Types and Functions — Icons

Constants

Types for Icon Families

```
kLarge1BitMask              = 'ICN#'
kLarge4BitData              = 'icl4'
kLarge8BitData              = 'icl8'
kSmall1BitMask              = 'ics#'
kSmall4BitData              = 'ics4'
kSmall8BitData              = 'ics8'
kMini1BitMask               = 'icm#'
kMini4BitData               = 'icm4'
kMini8BitData               = 'icm8'
```

IconAlignmentType Values

```
kAlignNone                  = $00
kAlignVerticalCenter        = $01
kAlignTop                   = $02
kAlignBottom                = $03
kAlignHorizontalCenter      = $04
kAlignAbsoluteCenter        = kAlignVerticalCenter + kAlignHorizontalCenter
kAlignCenterTop             = kAlignTop            + kAlignHorizontalCenter
kAlignCenterBottom          = kAlignBottom         + kAlignHorizontalCenter
kAlignLeft                  = $08
kAlignCenterLeft            = kAlignVerticalCenter + kAlignLeft
kAlignTopLeft               = kAlignTop            + kAlignLeft
kAlignBottomLeft            = kAlignBottom         + kAlignLeft
kAlignRight                 = $0C
kAlignCenterRight           = kAlignVerticalCenter + kAlignRight
```

```
kAlignTopRight                = kAlignTop      + kAlignRight
kAlignBottomRight             = kAlignBottom      + kAlignRight
```

IconTransformType Values

```
kTransformNone                = $00
kTransformDisabled            = $01
kTransformOffline             = $02
kTransformOpen                = $03
kTransformLabel1              = $0100
kTransformLabel2              = $0200
kTransformLabel3              = $0300
kTransformLabel4              = $0400
kTransformLabel5              = $0500
kTransformLabel6              = $0600
kTransformLabel7              = $0700
kTransformSelected            = $4000
kTransformSelectedDisabled    = kTransformSelected + kTransformDisabled
kTransformSelectedOffline     = kTransformSelected + kTransformOffline
kTransformSelectedOpen        = kTransformSelected + kTransformOpen
```

IconSelectorValue Masks

```
kSelectorLarge1Bit            = $00000001
kSelectorLarge4Bit            = $00000002
kSelectorLarge8Bit            = $00000004
kSelectorSmall1Bit            = $00000100
kSelectorSmall4Bit            = $00000200
kSelectorSmall8Bit            = $00000400
kSelectorMini1Bit             = $00010000
kSelectorMini4Bit             = $00020000
kSelectorMini8Bit             = $00040000
kSelectorAllLargeData         = $000000FF
kSelectorAllSmallData         = $0000FF00
kSelectorAllMiniData          = $00FF0000
kSelectorAll1BitData          = kSelectorLarge1Bit + kSelectorSmall1Bit + kSelectorMini1Bit
kSelectorAll4BitData          = kSelectorLarge4Bit + kSelectorSmall4Bit + kSelectorMini4Bit
kSelectorAll8BitData          = kSelectorLarge8Bit + kSelectorSmall8Bit + kSelectorMini8Bit
kSelectorAllAvailableData     = $FFFFFFFF
```

Data Types

```
IconAlignmentType = SInt16;
IconTransformType = SInt16;
IconSelectorValue = UInt32;
IconSuiteRef = Handle;
IconCacheRef = Handle;
```

CIcon

```
CIcon = RECORD
    iconPMap:        PixMap;                    { the icon's pixMap }
    iconMask:        BitMap;                    { the icon's mask }
    iconBMap:        BitMap;                    { the icon's bitMap }
    iconData:        Handle;                    { the icon's data }
    iconMaskData:    ARRAY [0..0] OF SInt16;    { icon's mask and BitMap data }
    END;

CIconPtr = ^CIcon;
CIconHandle = ^CIconPtr;
```

Functions

Drawing Icons From Resources

```
FUNCTION  PlotIconID({CONST}VAR theRect: Rect; align: IconAlignmentType;
                transform: IconTransformType; theResID: SInt16): OSErr;
PROCEDURE  PlotIcon({CONST}VAR theRect: Rect; theIcon: Handle);
FUNCTION  PlotIconHandle({CONST}VAR theRect: Rect; align: IconAlignmentType;
                transform: IconTransformType; theIcon: Handle): OSErr;
PROCEDURE PlotCIcon({CONST}VAR theRect: Rect; theIcon: CIconHandle);
FUNCTION  PlotCIconHandle({CONST}VAR theRect: Rect; align: IconAlignmentType;
                transform: IconTransformType; theCIcon: CIconHandle): OSErr;
FUNCTION  PlotSICNHandle({CONST}VAR theRect: Rect; align: IconAlignmentType;
                transform: IconTransformType; theSICN: Handle): OSErr;
```

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

```
FUNCTION   GetIcon(iconID: SInt16): Handle;
FUNCTION GetCIcon(iconID: SInt16): CIconHandle;
```

Disposing of Icons

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

Creating an Icon Suite

```
FUNCTION  GetIconSuite(VAR theIconSuite: IconSuiteRef; theResID: SInt16;
          selector: IconSelectorValue): OSErr;
FUNCTION  NewIconSuite(VAR theIconSuite: IconSuiteRef): OSErr;
FUNCTION  AddIconToSuite(theIconData: Handle; theSuite: IconSuiteRef;
          theType: ResType): OSErr;
```

Getting Icons From an Icon Suite

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

Drawing Icons From an Icon Suite

```
FUNCTION  PlotIconSuite({CONST}VAR theRect: Rect; align: IconAlignmentType;
          transform: IconTransformType; theIconSuite: IconSuiteRef): OSErr;
```

Performing Operations on Icons in an Icon Suite

```
FUNCTION  ForEachIconDo(theSuite: IconSuiteRef; selector: IconSelectorValue;
          action: IconActionUPP; yourDataPtr: UNIV Ptr): OSErr;
```

Disposing of Icon Suites

```
FUNCTION  DisposeIconSuite(theIconSuite: IconSuiteRef; disposeData: BOOLEAN): OSErr;
```

Converting an Icon Mask to a Region

```
FUNCTION  IconSuiteToRgn(theRgn: RgnHandle; {CONST}VAR iconRect: Rect;
          align: IconAlignmentType; theIconSuite: IconSuiteRef): OSErr;
FUNCTION  IconIDToRgn(theRgn: RgnHandle; {CONST}VAR iconRect: Rect;
          align: IconAlignmentType; iconID: SInt16): OSErr;
```

Determining Whether a Point or Rectangle is Within an Icon

```
FUNCTION  PtInIconID(testPt: Point; {CONST}VAR iconRect: Rect;
          align: IconAlignmentType; iconID: SInt16): BOOLEAN;
FUNCTION  PtInIconSuite(testPt: Point; {CONST}VAR iconRect: Rect;
          align: IconAlignmentType; theIconSuite: IconSuiteRef): BOOLEAN;
FUNCTION  RectInIconID({CONST}VAR testRect: Rect; {CONST}VAR iconRect: Rect;
          align: IconAlignmentType; iconID: SInt16): BOOLEAN;
FUNCTION  RectInIconSuite({CONST}VAR testRect: Rect; {CONST}VAR iconRect: Rect;
          align: IconAlignmentType; theIconSuite: IconSuiteRef): BOOLEAN;
```

Working With Icon Caches

```
FUNCTION  MakeIconCache(VAR theCache: IconCacheRef; makeIcon: IconGetterUPP;
          yourDataPtr: UNIV Ptr): OSErr;
FUNCTION  LoadIconCache({CONST}VAR theRect: Rect; align: IconAlignmentType;
          transform: IconTransformType; theIconCache: IconCacheRef): OSErr;
```

Demonstration Program

```
{  ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊
// GWorldPicCursIcon.p
// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊
//
// This program:
//
// •   Opens a window in which the results of various drawing, copying, and cursor shape
//     change operations are  displayed.
//
// •   Demonstrates offscreen graphics world, picture, cursor, cursor shape change,
//     animated cursor, and icon    operations as a result of the user choosing items from
//     a Demonstration menu.
//
// •   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:
//
// •   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 and is32BitCompatible flags
//     set.
//
// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ }

program GWorldPicCursIcon;

//
.................................................................................................................................................................................................................................
....................................................... includes

uses

    { Universal Interfaces. }
    Appearance, Devices, Fonts, GestaltEqu, Menus, PictUtils, Processes, Sound, TextUtils,
    ToolUtils, Resources;

//
.................................................................................................................................................................................................................................
....................................................... constants

const

rMenubar = 128;
rWindow = 128;
mApple = 128;
iAbout = 1;
mFile = 129;
iQuit = 11;
mDemonstration = 131;
iOffScreenGWorld1 = 1;
iOffScreenGWorld2 = 2;
iPicture = 3;
iCursor = 4;
iAnimatedCursor = 5;
iIcon = 6;
```

```
rBeachBallCursor = 128;
rPicture = 128;
rTransformStrings = 128;
rIconFamily1 = 128;
rIconFamily2 = 129;
rColourIcon = 128;
rAboutDialog = 128;
kSleepTime = 1;
kBeachBallTickInterval = 5;
kCountingHandTickInterval   = 30;

MAXLONG = $7FFFFFFF;

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

type

AnimCurs = record
    numberOfFrames : SInt16;
    whichFrame : SInt16;
    frame : array [0..0] of CursHandle;
    end;

AnimCursPtr = ^AnimCurs;
AnimCursHandle = ^AnimCursPtr;

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

var

gPreAllocatedBlockPtr : Ptr;
gMacOS85Present : Boolean;
gWindowPtr : WindowPtr;
gDone : boolean;
gInBackground : boolean;
gSleepTime : SInt32;
gCursorRegion : RgnHandle;
gCursorRegionsActive : boolean;
gAnimCursActive : boolean;
gAnimCursResourceHdl : Handle;
gAnimCursHdl : AnimCursHandle;
gAnimCursTickInterval : SInt16;
gAnimCursLastTick : SInt32;
gAnimationStep : UInt32;
gBlackColour : RGBColor;
gWhiteColour : RGBColor;
gBeigeColour : RGBColor;
gBlueColour : RGBColor;

// ......................................................................................................................................................... main
program block variables

osError : OSErr;
response : SInt32;
mainMenubarHdl : Handle;
mainMenuHdl : MenuHandle;
mainErr : OSErr;

//
.......................................................................................................................................................................
........... routine prototypes

procedure DoInitManagers; forward;
procedure EventLoop; forward;
procedure DoEvents({const} var theEvent : EventRecord); forward;
procedure DoMenuChoice(menuChoice : SInt32); forward;
procedure DoOffScreenGWorld1; forward;
procedure DoOffScreenGWorld2; forward;
procedure DoPicture; forward;
procedure DoCursor; forward;
procedure DoChangeCursor(theWindowPtr : WindowPtr; cursorRegion : RgnHandle); forward;
procedure DoAnimCursor; forward;
function  DoGetAnimCursor(int1, int2 : SInt16) : boolean; forward;
procedure DoIncrementAnimCursor; forward;
procedure DoReleaseAnimCursor; forward;
procedure DoIdle; forward;
procedure DoIcon; forward;
procedure DoAboutDialog; forward;
procedure DoGWorldDrawing; forward;
function  DoRandomNumber(minimum, maximum : UInt16) : UInt16; forward;

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoInitManagers

procedure DoInitManagers;
   var
   osError : OSErr;

   begin
   MaxApplZone;
   MoreMasters;

   InitGraf(@qd.thePort);
   InitFonts;
   InitWindows;
   InitMenus;
   TEInit;
   InitDialogs(nil);

   InitCursor;
   FlushEvents(everyEvent, 0);

   osError := RegisterAppearanceClient;

   end;
      { of procedure DoInitManagers }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ EventLoop

procedure EventLoop;
   var
   theEvent : EventRecord;
   gotEvent : Boolean;

   begin
   gDone := false;
   gSleepTime := MAXLONG;
   gCursorRegion := nil;

   while not gDone do
      begin
      gotEvent := WaitNextEvent(everyEvent, theEvent, gSleepTime, gCursorRegion);
      if gotEvent then
         begin
         DoEvents(theEvent);
         end
      else begin
         DoIdle;
         end;
      end;
   end;
      { of procedure EventLoop }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoEvents

procedure DoEvents({const} var theEvent : EventRecord);
   var
   theWindowPtr : WindowPtr;
   partCode : SInt16;
   charCode : SInt8;
   ignored : OSStatus;

   begin

   case (theEvent.what) of

      mouseDown: begin
         partCode := FindWindow(theEvent.where, theWindowPtr);
         case partCode of

            inMenuBar: begin
               DoMenuChoice(MenuSelect(theEvent.where));
               end;

            inContent: begin
               if (theWindowPtr <> FrontWindow) then
                  begin
```

```
                        SelectWindow(theWindowPtr);
                        end;
                    end;

                inDrag: begin
                    DragWindow(theWindowPtr, theEvent.where, qd.screenBits.bounds);
                    if gCursorRegionsActive then
                        begin
                        DoChangeCursor(gWindowPtr, gCursorRegion);
                        end;
                    end;
                end;
            end;

        keyDown, autoKey: begin
            charCode := BAnd(theEvent.message, charCodeMask);
            if (BAnd(theEvent.modifiers, cmdKey) <> 0) then
                begin
                DoMenuChoice(MenuEvent(theEvent));
                end;
            end;

        updateEvt: begin
            BeginUpdate(WindowPtr(theEvent.message));
            EndUpdate(WindowPtr(theEvent.message));
            end;

        osEvt: begin
            case BAnd(BSR(theEvent.message, 24), $000000FF) of

                suspendResumeMessage: begin
                    if (BAnd(theEvent.message, resumeFlag) = 1) then
                        begin
                        gInBackground := false;
{$ifc TARGET_CPU_PPC}
                        if (gMacOS85Present = true) then
                            begin
                            ignored := SetThemeCursor(kThemeArrowCursor);
                            end
                        else begin
                            SetCursor(qd.arrow);
                            end;
{$elsec}
                        SetCursor(qd.arrow);
{$endc}
                        end
                    else begin
                        gInBackground := true;
                        end;
                    end;

                mouseMovedMessage: begin
                    DoChangeCursor(FrontWindow, gCursorRegion);
                    end;

                otherwise begin
                    end;
                end;
                    { of case statement }
            end;

        otherwise begin
            end;
        end;
            { of case statement }
    end;
        { of procedure DoEvents }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoMenuChoice

procedure DoMenuChoice(menuChoice : SInt32);
    var
    menuID, menuItem : SInt16;
    itemName : Str255;
    daDriverRefNum : SInt16;
    ignored : OSStatus;

    begin
    menuID := HiWord(menuChoice);
    menuItem := LoWord(menuChoice);
    if (menuID = 0) then
        begin
        Exit(DoMenuChoice);
        end;

    if (gAnimCursActive = true) then
        begin
        gAnimCursActive := false;
{$ifc TARGET_CPU_PPC}
        if (gMacOS85Present = true) then
            begin
            ignored := SetThemeCursor(kThemeArrowCursor);
            end
        else begin
            SetCursor(qd.arrow);
            DoReleaseAnimCursor;
            end;
{$elsec}
        SetCursor(qd.arrow);
        DoReleaseAnimCursor;
{$endc}
        gSleepTime := MAXLONG;
        end;

    if gCursorRegionsActive then
        begin
        gCursorRegionsActive := false;
        DisposeRgn(gCursorRegion);
        gCursorRegion := nil;
        end;

    case menuID of

        mApple: begin
            if (menuItem = iAbout) then
                begin
                DoAboutDialog;
                end
            else begin
                GetMenuItemText(GetMenuHandle(mApple), menuItem, itemName);
                daDriverRefNum := OpenDeskAcc(itemName);
                end;
            end;

        mFile: begin
            if (menuItem = iQuit) then
                begin
                gDone := true;
                end;
            end;

        mDemonstration: begin
            case menuItem of

                iOffScreenGWorld1: begin
                    DoOffScreenGWorld1;
                    end;

                iOffScreenGWorld2: begin
                    DoOffScreenGWorld2;
                    end;

                iPicture: begin
                    DoPicture;
                    end;

                iCursor: begin
                    DoCursor;
                    end;

                iAnimatedCursor: begin
                    DoAnimCursor;
                    end;

                iIcon: begin
                    DoIcon;
                    end;

                otherwise begin
                end;
            end;
                { of case statement }
        end;

    otherwise begin
        end;
    end;
        { of case statement }

    HiliteMenu(0);
    end;
        { of procedure DoMenuChoice }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoOffScreenGWorld1

procedure DoOffScreenGWorld1;
    var
    windowPortPtr : CGrafPtr;
    deviceHdl : GDHandle;
    qdErr : QDErr;
    gworldPortPtr : GWorldPtr;
    gworldPixMapHdl : PixMapHandle;
    sourceRect, destRect : Rect;

    begin

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

    RGBBackColor(gBlueColour);
    EraseRect(gWindowPtr^.portRect);

    SetWTitle(gWindowPtr, 'Simulated time-consuming drawing operation');

    DoGWorldDrawing;

    SetWTitle(gWindowPtr, 'Click mouse to repeat in offscreen graphics port');

    while not Button do
        begin
        end;

    RGBBackColor(gBlueColour);
    EraseRect(gWindowPtr^.portRect);
    RGBForeColor(gWhiteColour);
    MoveTo(190, 180);
    DrawString('Please Wait.  Drawing in offscreen graphics port.');

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

    SetCursor(GetCursor(watchCursor)^^);

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

    GetGWorld(windowPortPtr, deviceHdl);

    qdErr := NewGWorld(gworldPortPtr, 0, gWindowPtr^.portRect, nil, nil, 0);
    if ((gworldPortPtr = nil) or (qdErr <> noErr)) then
        begin
        SysBeep(10);
        Exit(DoOffScreenGWorld1);
        end;

    SetGWorld(gworldPortPtr, nil);

    // ............. lock pixel image for duration of drawing and erase offscreen to white

    gworldPixMapHdl := GetGWorldPixMap(gworldPortPtr);

    if not LockPixels(gworldPixMapHdl) then
        begin
        SysBeep(10);
        Exit(DoOffScreenGWorld1);
        end;

    EraseRect(gworldPortPtr^.portRect);

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

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

    SetGWorld(windowPortPtr, deviceHdl);

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

    sourceRect := gworldPortPtr^.portRect;
    destRect := windowPortPtr^.portRect;

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

    RGBBackColor(gWhiteColour);
    RGBForeColor(gBlackColour);

    CopyBits(GrafPtr(gworldPortPtr)^.portBits,
                 GrafPtr(windowPortPtr)^.portBits,
                 sourceRect, destRect, srcCopy, nil);

    if (QDError <> noErr) then
        begin
        SysBeep(10);
        end;

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

    UnlockPixels(gworldPixMapHdl);
    DisposeGWorld(gworldPortPtr);

    SetCursor(qd.arrow);

    SetWTitle(WindowPtr(windowPortPtr),
                    'Offscreen Graphics Worlds, Pictures,   Cursors and Icons');
    end;
        { of procedure DoOffScreenGWorld1 }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoOffScreenGWorld2

procedure DoOffScreenGWorld2;
    var
    picture1Hdl, picture2Hdl : PicHandle;
    sourceRect, maskRect, maskDisplayRect, dest1Rect, dest2Rect, destRect : Rect;
    windowPortPtr : CGrafPtr;
    deviceHdl : GDHandle;
    qdErr : QDErr;
    gworldPortPtr : GWorldPtr;
    gworldPixMapHdl : PixMapHandle;
    region1Hdl, region2Hdl, regionHdl : RgnHandle;
    a, sourceMode : SInt16;

    begin
    RGBBackColor(gBeigeColour);
    EraseRect(gWindowPtr^.portRect);

    // ............................................................................................. get the source picture and draw it in the window

    picture1Hdl := GetPicture(rPicture);
    if (picture1Hdl = nil) then
        begin
        ExitToShell;
        end;
    HNoPurge(Handle(picture1Hdl));
    SetRect(sourceRect, 116, 35, 273, 147);
    DrawPicture(picture1Hdl, sourceRect);
    HPurge(Handle(picture1Hdl));
    MoveTo(116, 32);
    DrawString('Source image');

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

    GetGWorld(windowPortPtr, deviceHdl);

    SetRect(maskRect, 0, 0, 157, 112);

    qdErr := NewGWorld(gworldPortPtr, 0, maskRect, nil, nil, 0);

    if ((gworldPortPtr = nil) or (qdErr <> noErr)) then
        begin
        SysBeep(10);
        Exit(DoOffScreenGWorld2);
        end;

    SetGWorld(gworldPortPtr, nil);

    // ......................................... lock pixel image for duration of drawing and erase offscreen to white

    gworldPixMapHdl := GetGWorldPixMap(gworldPortPtr);

    if not LockPixels(gworldPixMapHdl) then
        begin
        SysBeep(10);
        Exit(DoOffScreenGWorld2);
        end;

    EraseRect(gworldPortPtr^.portRect);

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

    picture2Hdl := GetPicture(rPicture + 1);
    if (picture2Hdl = nil) then
        begin
        ExitToShell;
        end;
    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('Copy of offscreen mask');

    // ........................................................ define an oval-shaped region and a round rectangle-shaped region

    SetRect(dest1Rect, 22, 171, 296, 366);
    region1Hdl := NewRgn;
    OpenRgn;
    FrameOval(dest1Rect);
    CloseRgn(region1Hdl);

    SetRect(dest2Rect, 308, 171, 582, 366);
    region2Hdl := NewRgn;
    OpenRgn;
    FrameRoundRect(dest2Rect, 100, 100);
    CloseRgn(region2Hdl);

    SetWTitle(WindowPtr(windowPortPtr), 'Click mouse to copy');
    while not Button do
        begin
        end;

    // ......... set background and foreground colour, then copy source to destination using mask

    RGBForeColor(gBlackColour);
    RGBBackColor(gWhiteColour);

    for a := 0 to 1 do
        begin
        if (a = 0) then
            begin
            regionHdl := region1Hdl;
            destRect := dest1Rect;
            sourceMode := srcCopy;
            MoveTo(22, 168);
            DrawString('Boolean source mode srcCopy');
            end
        else begin
            regionHdl := region2Hdl;
            destRect := dest2Rect;
            sourceMode := srcXor;
            MoveTo(308, 168);
            DrawString('Boolean source mode srcXor');
            end;
        CopyDeepMask(GrafPtr(windowPortPtr)^.portBits,
                        GrafPtr(gworldPortPtr)^.portBits,
                        GrafPtr(windowPortPtr)^.portBits,
                        sourceRect, maskRect, destRect, sourceMode + ditherCopy, regionHdl);

        if (QDError <> noErr) then
            begin
            SysBeep(10);
            end;
        end;

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

    UnlockPixels(gworldPixMapHdl);
    DisposeGWorld(gworldPortPtr);

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

    SetWTitle(WindowPtr(windowPortPtr),
                    'Offscreen Graphics Worlds, Pictures,   Cursors and Icons');
    end;
        { of procedure DoOffScreenGWorld2 }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoPicture

procedure DoPicture;
    var
    pictureRect, theRect : Rect;
    picParams : OpenCPicParams;
    oldClipRgn : RgnHandle;
    pictureHdl : PicHandle;
    a, left, top, right, bottom, random : SInt16;
    theColour : RGBColor;
    osErr : OSErr;
    pictInfo : PictInfo;
    theString : Str255;

    begin
    RGBBackColor(gWhiteColour);
    EraseRect(gWindowPtr^.portRect);

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

    pictureRect := gWindowPtr^.portRect;
    pictureRect.right := (gWindowPtr^.portRect.right - gWindowPtr^.portRect.left) div 2;
    InsetRect(pictureRect, 10, 10);

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

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

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

    picParams.srcRect := pictureRect;
    picParams.hRes := $00480000;
    picParams.vRes := $00480000;
    picParams.version := -2;

    //
..........................................................................................................................................................................................................................................
..................... record picture

    pictureHdl := OpenCPicture(picParams);

    RGBBackColor(gBlueColour);
    EraseRect(pictureRect);

    for a := 0 to 299 do
        begin
    theRect := pictureRect;

    theColour.red := DoRandomNumber(0, 65535);
    theColour.green := DoRandomNumber(0, 65535);
    theColour.blue := DoRandomNumber(0, 65535);
    RGBForeColor(theColour);

    left := DoRandomNumber(10, UInt16(theRect.right + 1));
    top := DoRandomNumber(10, UInt16(theRect.bottom + 1));
    right := DoRandomNumber(UInt16(left), UInt16(theRect.right + 1));
    bottom := DoRandomNumber(UInt16(top), UInt16(theRect.bottom + 1));
    SetRect(theRect, left, top, right, bottom);

    PenMode(DoRandomNumber(addOver, adMin + 1));

    random := DoRandomNumber(0, 6);

    if (random = 0) then
        begin
        MoveTo(left, top);
        LineTo(right - 1, bottom - 1);
        end
    else if (random = 1) then
        begin
        PaintRect(theRect);
        end
    else if (random = 2) then
        begin
        PaintRoundRect(theRect, 30, 30);
        end
    else if (random = 3) then
        begin
        PaintOval(theRect);
        end
    else if (random = 4) then
        begin
        PaintArc(theRect, 0, 300);
        end
    else if (random = 5) then
        begin
        TextSize(DoRandomNumber(10, 70));
        MoveTo(left, right);
        DrawString('GWorldPicCursIcon');
        end;
    end;

// ………………………………………………………………… 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);

osErr := GetPictInfo(pictureHdl, pictInfo, recordFontInfo + returnColorTable, 1,
                                systemMethod, 0);
if (osErr <> noErr)then
    begin
    SysBeep(10);
    end;

MoveTo(380, 70);
DrawString('Some Picture Information:');

MoveTo(380, 100);
DrawString('Lines: ');
NumToString(pictInfo.lineCount, theString);
DrawString(theString);
```

```
   MoveTo(380, 115);
   DrawString('Rectangles: ');
   NumToString(pictInfo.rectCount, theString);
   DrawString(theString);

   MoveTo(380, 130);
   DrawString('Round rectangles: ');
   NumToString(pictInfo.rRectCount, theString);
   DrawString(theString);

   MoveTo(380, 145);
   DrawString('Ovals: ');
   NumToString(pictInfo.ovalCount, theString);
   DrawString(theString);

   MoveTo(380, 160);
   DrawString('Arcs: ');
   NumToString(pictInfo.arcCount, theString);
   DrawString(theString);

   MoveTo(380, 175);
   DrawString('Polygons: ');
   NumToString(pictInfo.polyCount, theString);
   DrawString(theString);

   MoveTo(380, 190);
   DrawString('Regions: ');
   NumToString(pictInfo.regionCount, theString);
   DrawString(theString);

   MoveTo(380, 205);
   DrawString('Text strings: ');
   NumToString(pictInfo.textCount, theString);
   DrawString(theString);

   MoveTo(380, 220);
   DrawString('Unique fonts: ');
   NumToString(pictInfo.uniqueFonts, theString);
   DrawString(theString);

   MoveTo(380, 235);
   DrawString('Unique colours: ');
   NumToString(pictInfo.uniqueColors, theString);
   DrawString(theString);

   MoveTo(380, 250);
   DrawString('Frame rectangle left: ');
   NumToString(pictInfo.sourceRect.left, theString);
   DrawString(theString);

   MoveTo(380, 265);
   DrawString('Frame rectangle top: ');
   NumToString(pictInfo.sourceRect.top, theString);
   DrawString(theString);

   MoveTo(380, 280);
   DrawString('Frame rectangle right: ');
   NumToString(pictInfo.sourceRect.right, theString);
   DrawString(theString);

   MoveTo(380, 295);
   DrawString('Frame rectangle bottom: ');
   NumToString(pictInfo.sourceRect.bottom, theString);
   DrawString(theString);

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

   KillPicture(pictureHdl);
   end;
      { of procedure DoPicture }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoCursor

procedure DoCursor;
   var
   cursorRect : Rect;
   a : SInt16;

   begin
```

```
    RGBBackColor(gBlueColour);
    EraseRect(gWindowPtr^.portRect);

    cursorRect := gWindowPtr^.portRect;

    for a := 0 to 2 do
        begin
        InsetRect(cursorRect, 40, 40);

        if ((a = 0) or (a = 2)) then
            begin
            RGBBackColor(gBeigeColour);
            end
        else begin
            RGBBackColor(gBlueColour);
            end;

        EraseRect(cursorRect);
        end;

    RGBForeColor(gBeigeColour);
    MoveTo(10, 20);
    DrawString('Arrow cursor region');
    RGBForeColor(gBlueColour);
    MoveTo(50, 60);
    DrawString('IBeam cursor region');
    RGBForeColor(gBeigeColour);
    MoveTo(90, 100);
    DrawString('Cross cursor region');
    RGBForeColor(gBlueColour);
    MoveTo(130, 140);
    DrawString('Plus cursor region');

    gCursorRegionsActive := true;
    gCursorRegion := NewRgn;
    end;
        { of procedure DoCursor }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoChangeCursor

procedure DoChangeCursor(theWindowPtr : WindowPtr; cursorRegion : RgnHandle);
    var
    cursorRect : Rect;
    arrowCursorRgn : RgnHandle;
    ibeamCursorRgn : RgnHandle;
    crossCursorRgn : RgnHandle;
    plusCursorRgn : RgnHandle;
    mousePosition : Point;
    ignored : OSStatus;

    begin
    arrowCursorRgn := NewRgn;
    ibeamCursorRgn := NewRgn;
    crossCursorRgn := NewRgn;
    plusCursorRgn := NewRgn;

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

    cursorRect := theWindowPtr^.portRect;
    LocalToGlobal(cursorRect.topLeft);
    LocalToGlobal(cursorRect.botRight);

    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);

    GetMouse(mousePosition);
    LocalToGlobal(mousePosition);

    if PtInRgn(mousePosition, ibeamCursorRgn) then
        begin
{$ifc TARGET_CPU_PPC}
```

```
        If (gMacOS85Present = true) then
            begin
            ignored := SetThemeCursor(kThemeIBeamCursor);
            end
        else begin
{$endc}
        SetCursor(GetCursor(iBeamCursor)^^);
{$ifc TARGET_CPU_PPC}
        end;
{$endc}
        CopyRgn(ibeamCursorRgn, cursorRegion);
        end

    else if PtInRgn(mousePosition, crossCursorRgn) then
        begin
{$ifc TARGET_CPU_PPC}
        If (gMacOS85Present = true) then
            begin
            ignored := SetThemeCursor(kThemeCrossCursor);
            end
        else begin
{$endc}
        SetCursor(GetCursor(crossCursor)^^);
{$ifc TARGET_CPU_PPC}
        end;
{$endc}
        CopyRgn(crossCursorRgn, cursorRegion);
        end

    else if PtInRgn(mousePosition, plusCursorRgn) then
        begin
{$ifc TARGET_CPU_PPC}
        If (gMacOS85Present = true) then
            begin
            ignored := SetThemeCursor(kThemePlusCursor);
            end
        else begin
{$endc}
        SetCursor(GetCursor(plusCursor)^^);
{$ifc TARGET_CPU_PPC}
        end;
{$endc}
        CopyRgn(plusCursorRgn, cursorRegion);
        end

    else begin
{$ifc TARGET_CPU_PPC}
        If (gMacOS85Present = true) then
            begin
            ignored := SetThemeCursor(kThemeArrowCursor);
            end
        else begin
{$endc}
        SetCursor(qd.arrow);
{$ifc TARGET_CPU_PPC}
        end;
{$endc}
        CopyRgn(arrowCursorRgn, cursorRegion);
        end;

    DisposeRgn(arrowCursorRgn);
    DisposeRgn(ibeamCursorRgn);
    DisposeRgn(crossCursorRgn);
    DisposeRgn(plusCursorRgn);
    end;
        { of procedure DoChangeCursor }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoAnimCursor

procedure DoAnimCursor;
    var
    animCursResourceID, animCursTickInterval : SInt16;

    begin
    BackColor(whiteColor);
    FillRect(gWindowPtr^.portRect, qd.white);

{$ifc TARGET_CPU_PPC}
    if (gMacOS85Present) then
        begin
        gAnimCursTickInterval := kCountingHandTickInterval;
        gSleepTime := gAnimCursTickInterval;
        gAnimCursActive := true;
        end
    else begin
{$endc}
        animCursResourceID := rBeachBallCursor;
        animCursTickInterval := kBeachBallTickInterval;

        if DoGetAnimCursor(animCursResourceID, animCursTickInterval) then
            begin
            gAnimCursActive := true;
            gSleepTime := animCursTickInterval;
            end
        else begin
            SysBeep(10);
            end;
{$ifc TARGET_CPU_PPC}
        end;
{$endc}
    end;
        { of procedure DoAnimCursor }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoGetAnimCursor

function DoGetAnimCursor(resourceID, tickInterval : SInt16) : boolean;
    var
    cursorID, a : SInt16;
    noError : boolean;

    begin
    a := 0;
    noError := false;

    gAnimCursHdl := AnimCursHandle(GetResource('acur', resourceID));
    if (gAnimCursHdl <> nil) then
        begin
        noError := true;
        while ((a < gAnimCursHdl^^.numberOfFrames) and noError) do
            begin
            cursorID := SInt16(HiWord(SInt32(gAnimCursHdl^^.frame[a])));
            gAnimCursHdl^^.frame[a] := GetCursor(cursorID);
            if (gAnimCursHdl^^.frame[a] <> nil) then
                begin
                a := a + 1;
                end
            else begin
                noError := false;
                end;
            end;
        end;

    if noError then
        begin
        gAnimCursTickInterval := tickInterval;
        gAnimCursLastTick := TickCount;
        gAnimCursHdl^^.whichFrame := 0;
        end;

    DoGetAnimCursor := noError;
    end;
        { of procedure DoGetAnimCursor }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoIncrementAnimCursor

procedure DoIncrementAnimCursor;
    var
    newTick : SInt32;
    ignored : OSStatus;

    begin
    newTick := TickCount;
    if (newTick < (gAnimCursLastTick + gAnimCursTickInterval)) then
        begin
        Exit(DoIncrementAnimCursor);
        end;

{$ifc TARGET_CPU_PPC}
    if (gMacOS85Present) then
        begin
        ignored := SetAnimatedThemeCursor(kThemeCountingUpAndDownHandCursor, gAnimationStep);
        gAnimationStep := gAnimationStep + 1;
        end
    else begin
{$endc}
        SetCursor(gAnimCursHdl^^.frame[gAnimCursHdl^^.whichFrame]^^);
        gAnimCursHdl^^.whichFrame := gAnimCursHdl^^.whichFrame + 1;
        if (gAnimCursHdl^^.whichFrame = gAnimCursHdl^^.numberOfFrames) then
            begin
            gAnimCursHdl^^.whichFrame := 0;
            end;
{$ifc TARGET_CPU_PPC}
        end;
{$endc}
        gAnimCursLastTick := newTick;
    end;
        { of procedure DoIncrementAnimCursor }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoReleaseAnimCursor

procedure DoReleaseAnimCursor;
    var
    a : SInt16;

    begin
    for a := 0 to gAnimCursHdl^^.numberOfFrames - 1 do
        begin
        ReleaseResource(Handle(gAnimCursHdl^^.frame[a]));
        end;

    ReleaseResource(Handle(gAnimCursHdl));
    end;
        { of procedure DoReleaseAnimCursor }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoIdle

procedure DoIdle;
    begin
    if gAnimCursActive then
        begin
        DoIncrementAnimCursor;
        end;
    end;
        { of procedure DoIdle }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoIcon

procedure DoIcon;
    var
    a, b, stringIndex : SInt16;
    theRect : Rect;
    transform : IconTransformType;
    theString : Str255;
    iconSuiteHdl : Handle;
    ciconHdl : CIconHandle;
    ignoredErr : OSErr;

    begin
    stringIndex := 1;
    transform := 0;

    RGBForeColor(gBlueColour);
    RGBBackColor(gBeigeColour);
    EraseRect(gWindowPtr^.portRect);

    // ..............................................................................................................................................................................................
PlotIconID with transforms

    MoveTo(50, 28);
    DrawString('PlotIconID with transforms');

    for a := 50 to 471 by 140 do
        begin
        if (a = 190) then
            begin
            transform := 16384;
            end
        else if (a = 330) then
            begin
            transform := 256;
            end;

        for b := 0 to 3 do
            begin
            if ((a <> 470) or (b <> 3)) then
                begin
                GetIndString(theString, rTransformStrings, stringIndex);
                stringIndex := stringIndex + 1;
                MoveTo(a, b * 60 + 47);
                DrawString(theString);

                SetRect(theRect, a, b * 60 + 50, a + 32, b * 60 + 82);
                ignoredErr := PlotIconID(theRect, 0, transform, rIconFamily1);
                SetRect(theRect, a + 40, b * 60 + 50, a + 56, b * 60 + 66);
                ignoredErr := PlotIconID(theRect, 0, transform, rIconFamily1);
                SetRect(theRect, a + 64, b * 60 + 50, a + 80, b * 60 + 62);
                ignoredErr := PlotIconID(theRect, 0, transform, rIconFamily1);

                if (a >= 330) then
                    begin
                    transform := transform + 256;
                    end
                else begin
                    transform := transform + 1;
                    end;
                end;
            end;
        end;

    // ..................................................................................................................................................... GetIconSuite
and PlotIconSuite

    MoveTo(50, 275);
    LineTo(550, 275);
    MoveTo(50, 299);
    DrawString('GetIconSuite and PlotIconSuite');

    ignoredErr := GetIconSuite(iconSuiteHdl, rIconFamily2, kSelectorAllLargeData);

    SetRect(theRect, 50, 324, 82, 356);
    ignoredErr := PlotIconSuite(theRect, kAlignNone, kTransformNone, iconSuiteHdl);
    SetRect(theRect, 118, 316, 166, 364);
    ignoredErr := PlotIconSuite(theRect, kAlignNone, kTransformNone, iconSuiteHdl);
    SetRect(theRect, 202, 308, 266, 372);
    ignoredErr := PlotIconSuite(theRect, kAlignNone, kTransformNone, iconSuiteHdl);

    //
..................................................................................................................................................................
GetCIcon and PlotCIcon

    MoveTo(330, 299);
    DrawString('GetCIcon 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);
    end;
        { of procedure DoIcon }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoAboutDialog

procedure DoAboutDialog;
    var
    dialogPtr : DialogPtr;
    itemHit : SInt16;

    begin
    dialogPtr := GetNewDialog(rAboutDialog, gPreAllocatedBlockPtr, WindowPtr(-1));
    ModalDialog(nil, itemHit);
    DisposeDialog(dialogPtr);
    end;
        { of procedure DoAboutDialog }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoGWorldDrawing
```

```
procedure DoGWorldDrawing;
    var
    a : SInt32;
    b, c : SInt16;
    theRect : Rect;
    theColour : RGBColor;

    begin
    RGBForeColor(gBeigeColour);
    PaintRect(gWindowPtr^.portRect);

    for a :=0 to 55000 by 300 do
        begin
        theColour.red := a;
        theColour.blue := 55000 - a;
        theColour.green := 0;
        RGBForeColor(theColour);

        for b := 10 to 531 by 65 do
            begin
            for c := 10 to 331 by 64 do
                begin
                SetRect(theRect, b, c, b + 62, c + 61);
                PaintRect(theRect);
                end;
            end;
        end;
    end;
        { of procedure DoGWorldDrawing }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ DoRandomNumber

function DoRandomNumber(minimum, maximum : UInt16) : UInt16;
    {
    var
    randomNumber : UInt16;
    range, t : SInt32;
    }
    begin
    {randomNumber := Random;
    range := maximum - minimum;
    t := (randomNumber * range) / 65535;
    Exit()(t + minimum);}
    DoRandomNumber := minimum + Abs(Random) mod (maximum - minimum + 1);
    end;
        { of function DoRandomNumber }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ main program block

begin

    gMacOS85Present := false;
    gCursorRegionsActive := false;
    gAnimCursActive := false;
    gAnimCursResourceHdl := nil;

    gBlackColour.red := $0000;
    gBlackColour.green := $0000;
    gBlackColour.blue := $0000;

    gWhiteColour.red := $FFFF;
    gWhiteColour.green := $FFFF;
    gWhiteColour.blue := $FFFF;

    gBeigeColour.red := $F000;
    gBeigeColour.green := $E300;
    gBeigeColour.blue := $C200;

    gBlueColour.red := $4444;
    gBlueColour.green := $4444;
    gBlueColour.blue := $9999;

    // ...................................................... get nonrelocatable block low in heap for About… dialog structure

    gPreAllocatedBlockPtr := NewPtr(sizeof(DialogRecord));
    if (gPreAllocatedBlockPtr = nil) then
        begin
        ExitToShell;
        end;
```

```
  //
...............................................................................................................................................................................
...... initialise managers

  DoInitManagers;

  // ........................................................................................................... check whether Mac OS 8.5 or later is
present

  osError := Gestalt(gestaltSystemVersion, response);
  if ((osError = noErr) and (response >= $00000850)) then
    begin
    gMacOS85Present := true;
    end;

  // ........................................................................................................................................................ see
random number generator

  GetDateTime(UInt32(qd.randSeed));

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

  mainMenubarHdl := GetNewMBar(rMenubar);
  if (mainMenubarHdl = nil) then
    begin
    ExitToShell;
    end;
  SetMenuBar(mainMenubarHdl);
  DrawMenuBar;
  mainMenuHdl := GetMenuHandle(mApple);
  if (mainMenuHdl = nil) then
    begin
    ExitToShell;
    end
  else begin
    AppendResMenu(mainMenuHdl, 'DRVR');
    end;

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

  gWindowPtr := GetNewCWindow(rWindow, nil, WindowPtr(-1));
  if (gWindowPtr = nil) then
    begin
    ExitToShell;
    end;

  SetPort(gWindowPtr);
  TextSize(10);

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

  EventLoop;

end.
  { of main program block }

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊
```

Demonstration Program Comments

If monitor colour depth is set to Thousands (or less), the Minimum Heap Size of 680K set in the CodeWarrior project will be adequate. If monitor depth is set to Millions, the Minimum Heap Size will need to be increased. The determinant is the memory required by the offscreen graphics world created within the function DoOffScreenGWorld1.

When this program is run, the user should:

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

- Note that the Icons item in the Demonstration menu contains an icon.

When the PowerPC target is run, and Mac OS 8.5 or later is present, the new Appearance Manager Version 1.1 functions for setting cursor shape and animating the cursor are used. When Mac OS 8.5 or later is not present (PowerPC target), or when the 68K target is run, the older cursor shape changing and animating functions and methodologies are used.

constants

In addition to the usual window and menu-related constants, constants are established for the resource IDs of 'acur', 'PICT', 'STR#', icon family, and 'cicn' resources, and a 'DLOG' resource. KSleeptime and MAXLONG will be assigned WaitNextEvent's sleep parameter at various points in the program. kBeachBallTickInterval represents the interval between frame changes for the animated cursor (68K target, or PowerPC target when Mac OS 8.5 or later is not present). kCountingHandTickInterval represents the interval between frame changes for the animated cursor (PowerPC target with Mac OS 8.5 or later present).

data types

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

global Variables

gPreAllocatedBlock will be assigned a pointer to a nonrelocatable block for the dialog structure associated with the About… dialog. gMacOS85Present will be assigned true if Mac OS 8.5 or later is present. gWindowPtr will be assigned a pointer to the window's window structure. gDone controls exit from the main event loop and thus program termination. gInBackground relates to foreground/background switching.

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

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

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

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

EventLoop

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

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

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

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

DoEvents

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

In the case of a resume event:

- If the target is the PowerPC target and Mac OS 8.5 or later is not present, or the target is the 68K target, SetCursor is called to ensure that the cursor is set to the arrow shape. The QuickDraw global variable arrow, which is of type Cursor, and which contains the arrow shaped cursor image, is passed in the newCursor parameter.

- If the target is the PowerPC target and Mac OS 8.5 or later is present SetThemeCursor is called to ensure that the cursor is set to the appearance-compliant arrow shape.

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

DoMenuChoice

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

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

Firstly, if the animated cursor demonstration is currently under way, the flag which indicates this condition is set to false. Then:

- If the target is the PowerPC target and Mac OS 8.5 or later is not present, or the target is the 68K target, SetCursor is called to set the cursor shape to the arrow shape and memory associated with the animated cursor is released.
- If the target is the PowerPC target and Mac OS 8.5 or later is present SetThemeCursor is called to set the cursor to the appearance-compliant arrow shape.

WaitNextEvent's sleep parameter is then set to the maximum possible value.

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

DoOffScreenGWorld1

DoWithoutOffScreenGWorld is the first demonstration.

Draw in Window

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

Draw in Offscreen Graphics Port and Copy to Window

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

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

The call to NewGWorld creates an offscreen graphics world. The offscreenGWorld parameter receives a pointer to the offscreen graphics world's graphics port. 0 in the pixelDepth means that the offscreen world's pixel depth will be set to the deepest device intersecting the rectangle passed as the boundsRect parameter. This Rect passed in the boundsRect parameter becomes the offscreen port's portRect, the offscreen pixel map's bounds and the offscreen device's gdRect. nil in the cTable parameter causes the default colour table for the pixel depth to be used. The aGDevice parameter is set to nil 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 called to prevent the base address of the pixel image from being moved when the pixel image it is drawn into or copied from.

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

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

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

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

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

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

DoOffScreenGWorld2

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

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

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

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

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

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

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

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

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

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

DoPicture

DoPicture demonstrates the recording and playing back a picture.

Define Picture Rectangle and Set Clipping Region

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

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

Set up OpenCPicParams Structure

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

Record Picture

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

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

Stop Recording, Draw Picture, Restore Saved Clipping Region

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

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

Display Some Information From The Pictinfo Structure

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

Release Memory Occupied By Picture Structure

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

DoCursor

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

The last two lines sets the gCursorRegionsActive flag to true and create an empty region for the last parameter of the WaitNextEvent call in the main event loop. A handle to a cursor region will be copied to gCursorRegion in the application-defined routine 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 lines establish a rectangle equivalent to the window's port rectangle and change this rectangle's coordinates from local to global coordinates so that the regions calculated from it will be in the required global coordinates. The call to InsetRect insets this rectangle by 40 pixels all round and the call to RectRgn establishes this as the I-beam region. The call to DiffRgn, in effect, cuts the rectangle represented by the I-beam region from the arrow region, leaving a hollow arrow region.

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

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

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

- If the target is the PowerPC target and Mac OS 8.5 or later is not present, or if the target is the 68K target, SetCursor is used to change the cursor shape.
- If the target is the PowerPC target and Mac OS 8.5 or later is present, SetThemeCursor is used to change the appearance-compliant cursor shape.

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

DoAnimCursor

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

If the target is the PowerPC target and Mac OS 8.5 or later is present, the Appearance Manager function SetAnimatedThemeCursor will be used in the application-defined function DoIncrementAnimCursor to increment the appearance-compliant cursor frame. In this case, DoAnimCursor simply assigns the appropriate frame change tick interval 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 gAnimCursActive is set to true so that DoIncrementAnimCursor will be called from the DoIdle function.

If the target is the PowerPC target and Mac OS 8.5 or later is not present, or if the target is the 68K target, the following applies:

- In this demonstration, application-defined 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. These functions are generic in that they may be used to initialise, animate and release any animated cursor passed to the getAnimCursor function as a formal parameter. A spinning "beach-ball" cursor is utilised in this demonstration. DoAnimCursor's major role is simply to call getAnimCursor with the beach-ball 'acur' resource as a parameter.
- The first line after #endif assigns the resource ID of the beach-ball 'acur' resource to the variable used as the first parameter in the later call to DoGetAnimCursor. The next line assigns a value represented by a constant 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 gAnimCursActive 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.

DoGetAnimatedCursor

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

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

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

The first line within the if block assigns the ticks value passed to DoGetAnimCursor to a global variable which will be utilised in the function 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 target is the PowerPC target and Mac OS 8.5 or later is present, the new Appearance Manager function SetThemeAnimatedCursor is called to increment the appearance-compliant cursor frame.

- If the target is the PowerPC target and Mac OS 8.5 or later is not present, or if the target is the 68K target, SetCursor sets the cursor shape to that represented by the handle stored in the specified element of the frame[] field of the animCurs structure. This line also increments the frame counter field (whichFrame) of the animCurs structure. If whichFrame has been incremented to the last cursor in the series, the frame counter is re-set to 0.

The last line retrieves and stores the tick count at exit for use at the first line the next time the function is called.

DoReleaseAnimCursor

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

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

DoIdle

DoIdle is called from the main event loop whenever a null event is received. If the active demonstration is the animated cursor demonstration, the application-defined routine DoSpinCursor is called.

DoIcon

DoIcon demonstrates the drawing of icons in a window using PlotIconID, PoltIconSuite, and PlotCIcon.

PlotIconID With Transforms

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

GetIconSuite and PlotIconSuite

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

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. Note that a pointer to a pre-allocated nonrelocatable memory block is passed in the dStorage parameter

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

DoGWorldDrawing and doRandomNumber

DoGWorldDrawing and DoRandomNumber are both incidental to the demonstration, DoGWorldDrawing is called from DoOffScreenGWorld1 to execute a drawing operation which will take a short but nonetheless perceptible period of time. DoRandomNumber generates the random numbers used within the DoPicture function.

main program block

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

If Mac OS 8.5 or later is present, gMacOS85Present is assigned true.

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

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