23

MISCELLANY

Includes Demonstration Program Miscellany

Notification From Applications in the Background

The Need for the Notification Manager

Applications running in the background cannot use the standard methods of communicating with the user, such as alert or dialog boxes, because such windows might easily be obscured by the windows of other applications. Furthermore, even if these windows are visible, the background application cannot be certain that the user has actually received the communication. Accordingly, some more reliable method must be used to manage communication between a background application and the user. The Notification Manager provides such a method.

Elements of a Notification

The Notification Manager creates **notifications**. A notification comprises one or more of five possible elements, which occur in the following sequence:

- An mark appears against the name of the target application in the Application menu.

  This mark is intended to prompt the user to switch the marked application to the foreground. The mark only appears while the application posting the notification remains in the background. It is replaced by the familiar mark when that application is brought to the foreground.

- The Application menu title begins alternating between the target application's icon and the foreground application's icon, or the Apple menu title begins alternating between the target application's icon and the Apple icon.

  The location of the icon alternation in the menu bar is determined by the posting application's mark (if any). If the application posting the notification is marked by either a mark or a mark in the Application menu, the Application menu title alternates; otherwise the Apple menu title alternates. Note that several applications might post notifications, so there might be a series of alternating icons.

- The Sound Manager plays a sound.

The application posting the notification can request that the system alert sound be used or it can specify its own sound by passing the Notification Manager a handle to a `'snd '` resource.

- In Mac OS 8.6 and earlier, a modal alert box appears, and the user dismisses it (by clicking on the Cancel button). In Mac OS 9.x, a floating window appears, allowing the current application's event loop to continue running while the notification alert is on the screen.

  The application posting the notification specifies the text for the modal alert box/floating window.

- A response function, if specified, executes.

  A response function can be used to remove the notification request from the notification queue (see below) or to perform other processing. For example, it can be used to set a global variable to record that the notification was received.

Suggested Notification Strategy

Apple's suggested notification strategy is to allow the user to set the desired level of notification at one of three levels, as follows:

- ***Level 1.*** Display the mark next to the name of the application in the Application menu.[1]

- ***Level 2.*** Display the mark next to the name of the application in the Application menu and alternate the icons. (This is the suggested default setting.)

- ***Level 3.*** Display the mark next to the name of the application in the Application menu, alternate the icons and invoke an alert box to notify the user that something needs to be done.

A sound might also be played at levels 2 and 3, but the user should have the option of turning the sound off. In addition, the user should be provided with the option of turning notification off altogether, except in cases where damage might occur or data would be lost.

That said, Apple accepts that this suggested strategy might not be appropriate for your application. (Indeed, notifications provided by the system software itself do not follow these guidelines.)

Notifications in Action

Overview

The Notification Manager is automatically initialised at system startup.

To issue a notification to the user, you need to create a **notification request** and install it into the **notification queue**, which is a standard Macintosh queue. The Notification Manager interprets the request and presents the notification to the user at the earliest possible time.

[1] Note that displaying the ◆ mark is only possible if the requesting software is listed in the Application Menu (and thus represents a process which is loaded into memory). The requesting software may not be an application. In addition to applications, other software that is largely invisible to the user can use the Notification Manager. Such software includes device drivers, vertical blanking (VBL) tasks, Time Manager tasks, and code which executes during the system startup sequence, such as code contained in extensions.

Eventually, you will need to remove the notification request from the notification queue. You can do this in the response function or when your application returns to the foreground.

Creating a Notification Request

The Notification Structure

When installing a request into the notification queue, your application must supply a pointer to a **notification structure**, a static and nonrelocatable structure of type NMRec which indicates the type of notification you require. Each entry in the notification queue is, in fact, a notification structure. The notification structure is as follows:

```
struct NMRec
{
   QElemPtr     qLink;          // Address of next element in queue. (Used internally.)
   short        qType;          // Type of data. (8 = nmType).
   short        nmFlags;        // (Reserved.)
   long         nmPrivate;      // (Reserved.)
   short        nmReserved;     // (Reserved.)
   short        nmMark;         // Application to identify with _ mark.
   Handle       nmIcon;         // Handle to small icon.
   Handle       nmSound;        // Handle to sound structure.
   StringPtr    nmStr;          // Pointer to string to appear in alert.
   NMUPP        nmResp;         // Pointer to response function.
   long         nmRefCon;       // Available for application use.
};
typedef struct NMRec NMRec;
typedef NMRec *NMRecPtr;
```

Field Descriptions:

To set up a notification request, you need to fill in at least the first six of the following fields:

qType	Indicates the type of operating system queue. Set to nmType (8).
nmMark	Indicates whether to place a ◆ mark next to the name of the application in the Application menu. If nmMark is 0, no mark appears. If nmMark is 1, the mark appears next to the name of the calling application. If nmMark is neither 0 nor 1, it is interpreted as the reference number of a desk accessory. An application should set nmMark to 1 and a driver or detached background task (such as a VBL task or Time Manager task) should set nmMark to 0.
nmIcon	A handle to an icon family containing a small colour icon, that is to alternate periodically in the menu bar. If nmIcon is set to NULL, no icon appears in the menu bar. This handle must be valid at the time the notification occurs. It does not need to be locked, but it must be non-purgeable.
nmSound	A handle to a sound resource to be played with SndPlay. If nmSound is set to NULL, no sound is produced. If nmSound is set to -1, the system alert sound is played. This handle does not need to be locked, but it must be non-purgeable.
nmStr	Points to a string which appears in the alert box. If nmStr is set to NULL, no alert box appears. Note that the Notification Manager does not make a copy of this string, so your application should not dispose of this storage until it removes the notification request.
nmResp	A universal procedure pointer to a response function. If nmResp is set to NULL, no response function executes when the notification is posted. If nmResp is set to -1, then a pre-defined function removes the notification request immediately after it has completed.

If you do not need to do any processing in response to the notification, you should set nmResp to NULL. If you supply the universal procedure pointer to your own response function, the Notification Manager passes it one parameter: a universal procedure pointer to your notification structure. For example, this is how you would declare a response function having the name theResponse:

```
pascal void  theResponse(NMUPP nmStructurePtr);
```

You can use response functions to remove notification requests from the notification queue, free any memory[2], or set a global variable in your application to record that the notification was posted[3]. If you are setting a global variable to enable you to determine that the user actually received the notification, you need to request an alert notification. This is because the response function executes only after the user has clicked the OK button in the alert box.

If you choose audible or alert notifications, you should probably set nmResp to -1 so that the notification structure is removed from the queue as soon as the sound has finished or the user has dismissed the alert box. However, if either nmMark or nmIcon is non-zero, do not set nmResp to -1, because the Notification Manager will remove the mark or the icon before the user sees it.

nmRefCon A long integer available for your application's own use.

Installing a Notification Request

NMInstall is used to add a notification request to the notification queue. The following is an example call:

```
osErr = NMInstall(&notificationStructure);
```

Before calling NMInstall, you should make sure that your application is running in the background. If your application is in the foreground, you simply use standard alert methods, rather than the Notification Manager, to gain the user's attention.

Removing a Notification Request

NMRemove is used to remove a notification request from the notification queue. The following is an example call:

```
osErr = NMRemove(&notificationStructure);
```

You can remove requests at any time, either before or after the notification actually occurs.

As previously stated, in Mac OS 9.x, notifications are non-blocking, meaning that the user can activate the posting application without dismissing the alert. For this reason, when your application is running on Mac OS 9.x, may wish to have it explicitly cancel an alert notification using NMRemove when the application becomes active.

[2] Note that an nmResp value of -1 does not free the memory block containing the queue element; it merely removes that element from the notification queue.

[3] When the Notification Manager calls your response function, it does not set up A5 or low-memory globals for you. If you need to access your application's global variables, you should save its A5 in the nmRefCon field.

Progress Bars and Scanning for Command-Period Key-Down Events and Mouse-Down Events

Progress Bars

Operations within an application which tie up the machine for relatively brief periods of time should be accompanied by a cursor shape change to the watch cursor, or perhaps to an animated cursor. On the other hand, lengthy operations should be accompanied by the display of a progress indicator.

The progress indicator control was described at Chapter 14 — More On Controls. A progress indicator created using this control may be determinate or indeterminate. Determinate progress indicators show how much of the operation has been completed. Indeterminate progress indicators show that an operation is occurring but does not indicate its duration. Ordinarily, progress indicators should be displayed within a dialog box.

As stated at Chapter 2 — Low and Operating System Events, your application should allow the user to cancel a lengthy operation using the Command-period key combination. You might also include a Stop push button in the dialog box in which the progress indicator is located.

Scanning for Command-Period Key-Down Events and Mouse-Down Events in a Stop Button

One way to satisfy this requirement is to periodically call an application-defined function which scans the event queue for Command-period key-down events and mouse-down events. This function should return true if:

- A Command-period keyboard event is found.
- A mouse-down event is found and the mouse-down was within the Stop button's rectangle.

The application-defined function should first get a pointer to the first queue element. It should then scan the queue for key-down and mouse-down events.

If a key-down event is found, the next step is to determine whether the Command key was down at the time of the key press. If it was, a check should be made as to whether the key pressed was the period key. If these checks reveal that a Command-period keyboard event has occurred, the function should return immediately, returning true to the calling function.

If a mouse-down event is found, the next step is to determine whether the mouse-down was within the Stop push button's rectangle and, if so, briefly highlight the push button before returning true to the calling function.

If true is returned to the calling function, that function should terminate the lengthy operation and close the progress indicator dialog box.

Soliciting a Colour Choice From the User — The Color Picker

The Color Picker Utilities provide your application with:

- A standard dialog box, called the **Color Picker**, for soliciting a colour choice from the user.
- Functions for converting colour specifications from one **colour model** to another.

Preamble - Colour Models

In the world of colour, three main colour models are used to specify a particular colour. These are the RGB (red, green, blue) model, the CYMK (cyan, magenta, yellow, black) model, and the HLS or HSV (hue, lightness, saturation, or hue, saturation, value) models.

RGB Model

The RGB model is used where light-produced colours are involved, as in the case of a television set, computer monitor, or stage lighting. In this model, the three primary colours involved (red, green, and blue) are said to be *additive* because, the more of each colour you add, the closer the resulting colour is to white.

CYMK Model

The CYMK model is closely associated with printing, that is, putting colour on a white page. In this model, the three primary colours (cyan, yellow, and magenta[4]) are said to be *subtractive* because, the more of each colour you add, the closer the resulting colour is to black. (The inclusion of black in the model accounts for the fact that the colours of printer's inks may vary slightly from true cyan, yellow, and magenta, meaning that a true black may not be achievable with just a CYM model.)

HLS and HSV Models

The HLS and HSV models separate colour (that is, hue) from saturation and brightness. Saturation is a measure of the amount of white in a colour (the less white, the more saturated the colour). Lightness is the measure of the amount of black in a colour. (The less black, the lighter the colour). The amount of black is specified by the lightness (L) value in the HLS model and by the value (V) value in the HSV model.

The HSL/HLV model may be represented diagrammatically by the HSL/HLV colour cone shown at Fig 1. In this colour cone, hue is represented by an angle between 0˚ and 360˚.

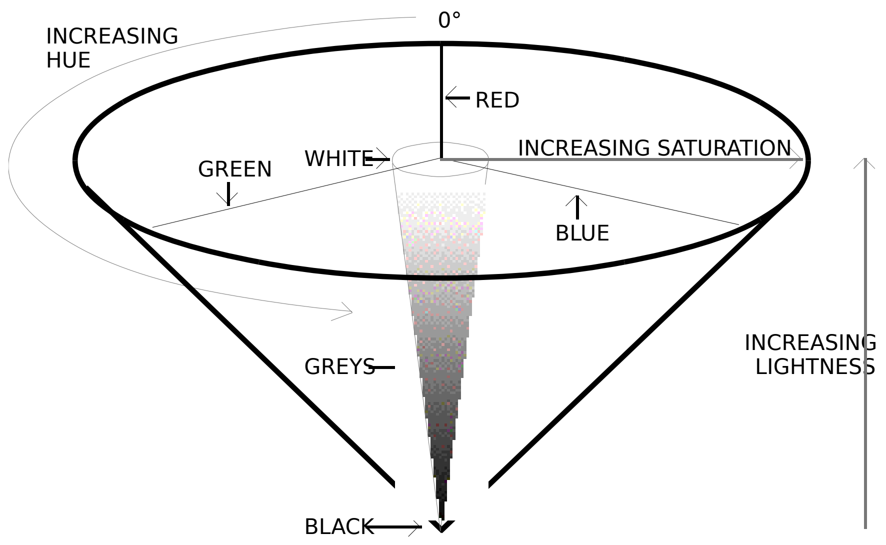


FIG 1 - HSL/HSV COLOUR CONE

[4] Cyan, magenta, and yellow are the complements of red, green, and blue.

The Color Picker

The Color Picker allows the user to specify a colour using either the RGB, CMYK, HLS, or HSV, models.

Using the Color Picker RGB Mode

Fig 2 shows the Color Picker in RGB mode. The desired red, green and blue values may be set using the three slider controls or may be entered directly into the edit text fields on the right of the sliders.

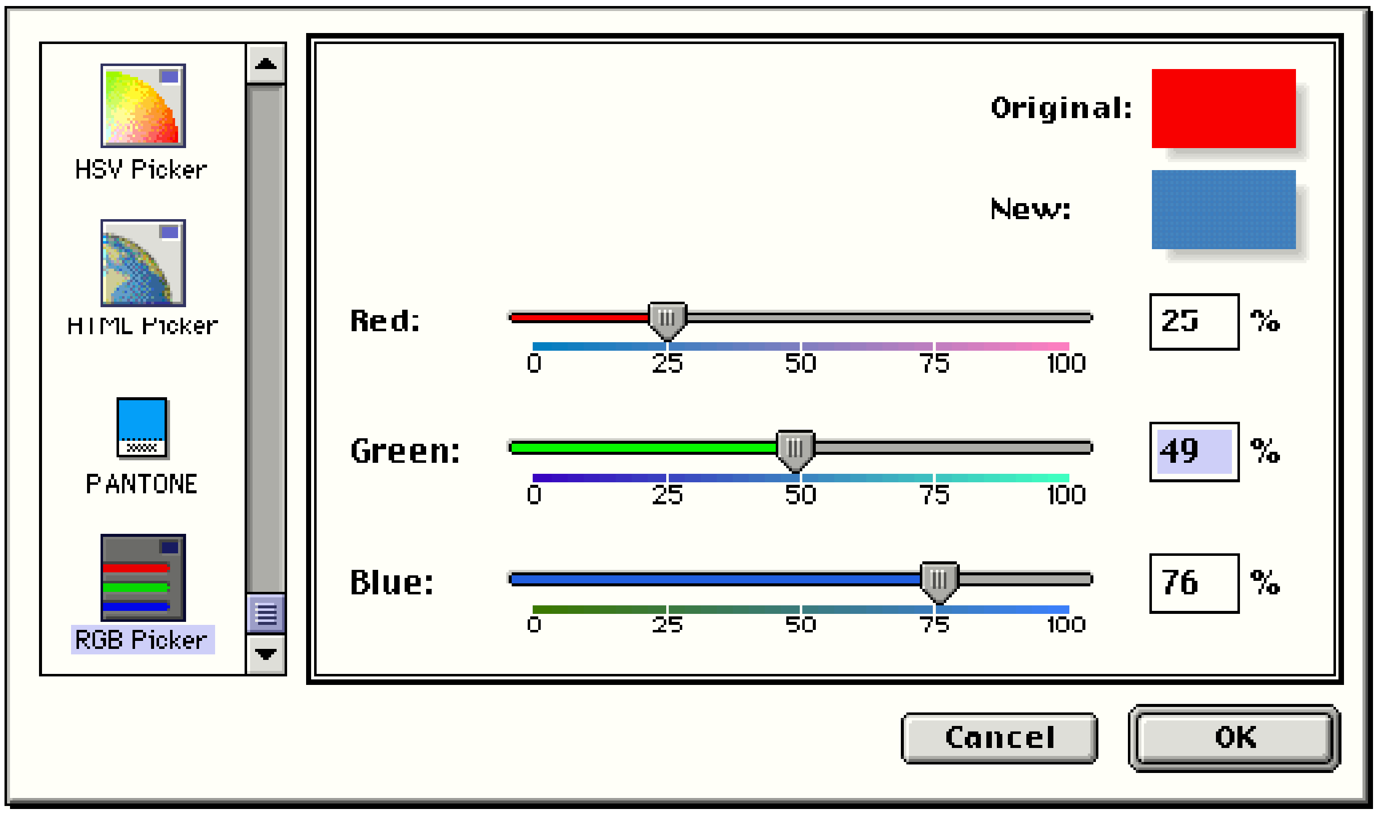
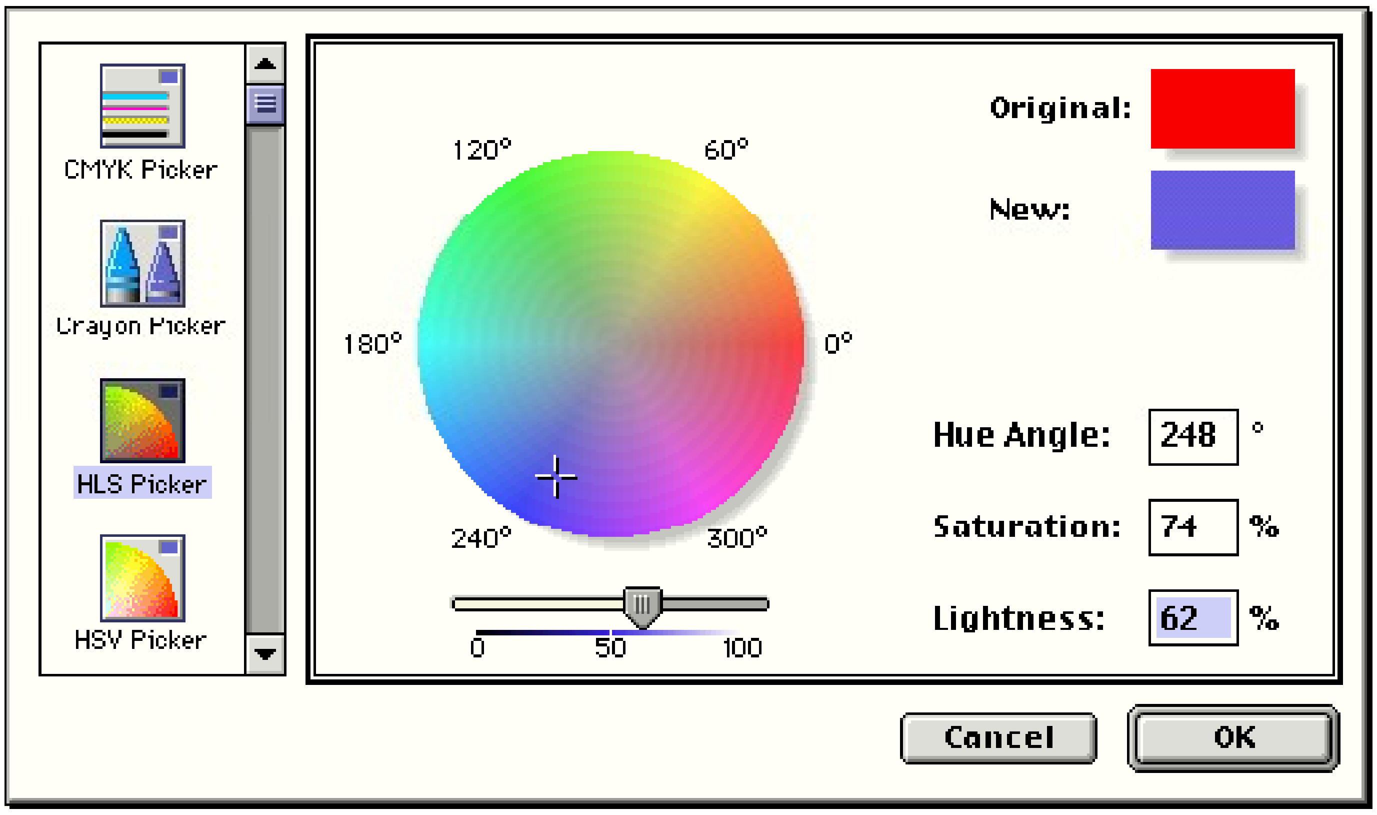


FIG 2 - COLOR PICKER DIALOG IN RGB MODE

Using the Color Picker in HLS Mode

Fig 3 shows the Color Picker in HLS mode. Hue is specified by an angle, which may be entered at Hue Angle:. Saturation is specified by percentage, which may be entered at Saturation:. Lightness is also specified by a percentage, which may be entered at Lightness: Alternatively, hue and saturation may be selected simultaneously by clicking at the desired point within the coloured disc, and lightness may be set with the slider control.

To relate Fig 3 to Fig 1, the coloured disc at Fig 3 may be considered as the HSL/HSV cone as viewed from above. The lightness slider control can then be conceived of as moving the disc up or down the axis of the cone from the apex (black) to the base (white).



Original:

New:

Hue Angle: 248 °

Saturation: 74 %

Lightness: 62 %

FIG 3 - COLOR PICKER IN HLS MODE

Invoking the Color Picker

The Color Picker is invoked using the GetColor function:

Boolean GetColor(Point *where*,ConstStr255Param *prompt*,const RGBColor **inColor*,
RGBColor **outColor*);

where	Dialog's upper-left corner. (0,0) causes the dialog box to positioned centrally on the main screen.
prompt	A prompt string, which is displayed in the upper left corner of the main pane in the dialog box.
inColor	The starting colour, which the user may want for comparison, and which is displayed against Original: in the top right corner of the dialog box.
outColor	Initially set to equal *inColor*. Assigned a new value when the user picks a colour. The colour stored in this parameter is displayed at the top right of the dialog box against New:.)

Returns: A Boolean value indicating whether the user clicked on the OK button or Cancel button.

If the user clicks the OK button in the Color Picker dialog, your application should adopt the outColor value as the colour chosen by the user. If the user clicks the Cancel button, your application should assume that the user has decided to make no colour change, that is, the colour should remain as that represented by the inColor parameter.

Coping With Multiple Monitors

Overview

Many Macintosh models can accommodate more than one monitor. In a multi-monitor system, the Monitors control panel allows the user to specify which of the attached monitors is to be the **main screen** (that is, the screen containing the menu bar) and to set the position of the other screen, or screens, relative to the main screen.

The maximum number of colours capable of being displayed by a given Macintosh at the one time is determined by the video capability of that particular Macintosh. The maximum number of colours capable of being displayed on a given screen at the one time depends on settings made by the user using the Monitors and Sound control panel. In a multi-monitor environment, therefore, it is possible for each screen to be set to a different pixel depth.

In more technical terms, a Monitors control panel colours/grays setting sets the pixel depth of a particular **video device**. A brief review of the subject of video devices is therefore appropriate at this point.

Video Devices Revisited

As stated at Chapter 11 — QuickDraw Preliminaries:

- A **graphics device** is anything into which QuickDraw can draw, a **video device** (such as a plug-in video card or a built-in video interface) is a graphics device that controls screens, Color QuickDraw stores information about video devices in GDevice structures, the system creates and initialises a GDevice structure for each video device found during start-up[5], all structures are linked together in a list called the **device list**, and the global variable DeviceList holds a handle to the first structure in the list.

- At any given time, one, and only one, graphics device is the **current device**[6], that is, the one in which the drawing is taking place. A handle to the current device's GDevice structure is placed in the global variable TheGDevice.

By default, the GDevice structure corresponding to the first video device found at start up is marked as the (initial) current device, and all other graphics devices in the list are initially marked as inactive. When the user moves a window to, or creates a window on, another screen, and your application draws into that window, Color QuickDraw automatically makes the video device for that screen the current device and stores that information in TheGDevice. As Color QuickDraw draws across a user's video devices, it keeps switching to the GDevice structure for the video device on which it is actively drawing.

Also recall from Chapter 11 — QuickDraw Preliminaries that two of the fields in a GDevice structure are:

- gdMap, which contains a handle to a pixel map which, in turn, contains a field (pixelSize) containing the device's pixel depth (that is, the number of bits per pixel).

- gdRect, which contains the device's global boundaries.

Requirements of the Application

Accommodating a multi-monitor environment requires that you address the following issues:

- ***Image Optimisation.*** To draw a particular graphic, your application may have to call different drawing functions for that graphic depending on the characteristics of the video device intersecting your window's drawing region, the aim being to

5 The Monitors and Sound control panel stores the pixel depth and other configuration information in a resource of type 'scrn' (resource ID 0). This resource contains an array of data structures which are analogous to GDevice records. Each element of this array contains information about a different video device. When InitGraf is called to initialize QuickDraw, it checks the System file for the 'scrn' resource. If the resource is found, and if it matches the hardware, InitGraf organises the video devices according to the resource's contents. If the resource is not found, QuickDraw uses only the video device of the startup screen.

6 The current device is sometimes referred to as the **active device**.

optimise the appearance of the image regardless of whether it is being displayed on, say, a grayscale device or a colour device. Recall from Chapter 11 — QuickDraw Preliminaries that when QuickDraw displays a colour on a grayscale screen, it computes the luminance, or intensity of light, of the desired colour and uses that value to determine the appropriate gray value to draw. It is thus possible that, for example, two overlapping objects drawn in two quite different colours on a colour screen may appear in the same shade of gray on a grayscale screen. In order for the user to differentiate between these two objects on a grayscale screen, you would need to provide an alternative drawing function which draws the two objects in different shades of gray on grayscale screens.

- ***Window Zooming.*** The second issue is window zooming. For example, if the user drags a window currently zoomed to the user state so that it spans two screens, and then clicks the zoom box to zoom the window to the standard state, your application will need to determine which screen contains the largest area of the window, calculate the standard state for that screen (which will depend, amongst other things, on whether that screen contains the menu bar), and finally zoom the window out to the standard state for that particular screen.

- ***Window Dragging and Sizing.*** In window dragging operations in a single-monitor environment, &qd.screenBits.bounds is typically passed in the limitRect parameter of DragWindow. (bounds is a rectangle which encloses the main screen.) Similarly, in window sizing operations in a single-monitor environment, the values in the bottom and right fields of bounds are typically assigned to the bottom and right fields of the rectangle passed in the sizeRect parameter of GrowWindow. For a multi-monitor environment, you should use the rectangle in the rgnBBox field of the Region structure filled in by a call to LMGetGrayRgn. This rectangle bounds the current desktop region, which spans multiple monitors.

Image Optimisation

The QuickDraw function DeviceLoop is central to the matter of optimising the appearance of your images. DeviceLoop searches for graphics devices which intersect your window's drawing region, informing your application of each graphics device it finds and providing your application with information about the current device's attributes. Armed with this information, your application can then invoke whichever of its drawing functions is optimised for those particular attributes.

DeviceLoop's second parameter is a pointer to an application-defined function. That function must be defined like this:

```
pascal void myDrawingFunction(short depth,short deviceFlags,GDHandle targetDevice,
                                        long userData)
```

DeviceLoop calls this function for each dissimilar video device it finds. If it encounters similar devices (that is, devices having the same pixel depth, colour table seeds, etc.) it will make only one call to myDrawingFunction, pointing to the first such device encountered. DeviceLoop's behaviour can, however, be modified by supplying the flags parameter with one of the following values:

Value	***Meaning***

singleDevices	Do not group similar devices when calling drawing function.
dontMatchSeeds	Do not consider ctSeed fields of ColorTable structures for graphics devices when comparing them.
allDevices	Ignore value of drawingRgn parameter and instead call drawing function for every screen.

Window Zooming

Handling window zooming in a multi-monitors environment requires that your application provide a special application-defined function. The user may have moved a window to a different screen, or to a position where it spans two separate screens, since it was last zoomed. When the user elects to zoom that window to the **standard state**[7], your application-defined function must first determine the screen on which the zoomed window is to appear and the appropriate standard state for that screen.

The screen on which the zoomed window should appear should be the screen on which the window is currently displayed or, if the window spans screens, the screen containing the largest area of the window. The appropriate standard state will depend on:

- The device's global boundaries, as contained in the gdRect field of the gDevice structure.
- The requirements of the application. (As stated at Chapter 4 — Windows, the standard state on the main screen is typically the gray area of the screen minus three pixels all round.)
- Whether the screen on which the zoomed window is to appear contains the menu bar.

After determining the screen on which the zoomed window is to appear and calculating the standard state, your application-defined function should call ZoomWindow to redraw the window frame in its new location and, finally, redraw the window's content region.

Vertical Blanking (VBL) Tasks

VBL Tasks and the Vertical Retrace Manager

The video circuitry in a Macintosh refreshes the screen at regular intervals, the exact interval depending on the video hardware. To refresh the screen, the monitor's electron beam draws in horizontal lines, starting at the upper left corner, finishing at the lower right corner, and then jumping back to the upper left corner. When the electron beam returns from the lower right corner to the upper left corner, the video circuitry generates a **vertical retrace interrupt** or **vertical blanking (VBL) interrupt**.

The Vertical Retrace Manager schedules tasks, known as **VBL tasks**, for execution during the vertical retrace interrupt. The Operating System itself uses the Vertical Retrace Manager to perform certain housekeeping operations, such as updating the global variable Ticks and the position of the cursor (every interrupt) and checking whether a disk has been inserted (every 30 interrupts).

You can also use the Vertical Retrace Manager to install your own recurrent tasks which, for some reason, you do not want to execute in your main event loop. Be aware, however, that:

- The Vertical Retrace Manager is useful only for small, repetitive tasks which do not allocate or release memory.

[7] See Chapter 4 — Windows for a description of standard state, user state, and the state data record.

- The Vertical Retrace Manager is not an absolute timing device. Its operations are always relative to the VBL interrupt, which is sometimes disabled — for example, during disk access. (This latter explains the jerky cursor movement experienced during disk operations.)

VBL tasks installed by the Operating System are not maintained in the same queue as that used by application-defined VBL tasks.

Types of VBL Tasks

There are two general types of VBL tasks:

- ***Slot-Based VBL Tasks.*** Slot-based VBL tasks are linked to an external video monitor. Because different monitors have different refresh rates, and hence execute VBL tasks at different intervals, a separate task queue is maintained for each attached video device. When a VBL interrupt occurs for one of these devices, the tasks in the queue relating to the slot holding that device's video card are executed. A slot-based VBL task is installed using SlotVInstall.

- ***System-Based VBL Tasks.*** System-based VBL tasks apply to Macintoshes which have only a built-in monitor. On such machines, there is no need to isolate VBL tasks into separate queues. System-based VBL tasks are installed using VInstall.

To maintain compatibility on modular Macintoshes for software which uses VInstall, the Operating System generates a special interrupt at a frequency identical to the retrace rate on compact Macintoshes. This ensures that application tasks installed using the VInstall function, as well as the periodic system tasks previously described, are performed as usual.

VBL Task Rules

A VBL task which violates any of the following rules may cause a system crash:

- A VBL task must not allocate, move, or purge memory, or call any Toolbox functions which may do so.

- Applicable to 680x0 code only, a VBL task cannot call a function from any other code segment (see Code Segmentation, below) unless it sets up the application's A5 world properly. In addition, that segment must already be loaded in memory.

- A VBL task cannot access your application's global variables unless it sets up the application's A5 world properly.

- A VBL task's code, and any data accessed during the execution of the task, must be locked into physical memory if virtual memory is in operation.

VBL Tasks and Foreground/Background Switching

Some VBL tasks may be intended to perform services which are useful only to the application, and which should therefore cease execution if the application is switched to the background. Others may be intended to continue to execute even when the application is no longer in the foreground.

System-Based VBL Tasks

If the address of a system-based VBL task (not the same thing as the address of the VBL task structure) is anywhere in the partition of the application that installed it, the Process Manager automatically disables that task when it is sent to the background. Then, when the application regains control of the processor (through either a minor or major switch), the task is re-enabled. This does not apply if the address of a system-based VBL task is in the system partition[8].

Note that, in the case of the address of the system-based task being in the application's partition, the task is re-enabled when the application receives processing time, which can occur without the application necessarily returning to foreground. For that reason, you may want to disable a system-based VBL task manually. This can be done using the same procedure as that applying to the disabling of a slot-based VBL task (see below).

Slot-Based VBL Tasks

By contrast, the Process Manager never disables a slot-based VBL task, no matter where the task is located. Accordingly, if you want a slot-based VBL task to be disabled when your application is in the background, you must do it yourself, either by removing the task structure from the VBL queue or by setting the vblCount field of the task structure (see below) to 0. You can do this in response to a suspend event. Then, when your application receives a resume event, you can re-enable the task by re-installing the task structure or by re-setting the vblCount field of the VBL task structure (see below) to the appropriate value.

Installing and Removing a VBL Task

You use the Vertical Retrace Manager to install and remove **VBL task structures** in and from system-based or slot-based vertical retrace queues. Before you call VInstall or SlotVInstall to install a task structure, you must first fill in the last four of the VBL task structure's fields.

The VBL Task Structure

The VBL task structure is defined by the VBLTask data type:

```
struct VBLTask
{
   QElemPtr    qLink;
   short       qType;
   VBLUPP      vblAddr;
   short       vblCount;
   short       vblPhase;
};
typedef struct VBLTask VBLTask;
typedef VBLTask *VBLTaskPtr;
```

Field Descriptions

qLink	Pointer to the next entry in the queue. (This field is not set by the application. It is set by the Vertical Retrace Manager.)
qType	The queue type. This must be set to vType.
vblAddr	Pointer to the function that the Vertical Retrace Manager is to execute.
vblCount	The number of interrupts before the function first executes. The Vertical Retrace Manager lowers this number by 1 during each interrupt. If the value in vblCount is 0, the task will not execute. If, when vblCount contains 0, you want the function to be executed again, you must reset the vblCount field to the required value.

[8] You load a system-based task's VBL task record into the system partition when you want the task to be a **persistent VBL task**, that is, a task that continues to be executed even when the application which installed it is no longer in control of the CPU. (Note that slot-based VBLs are always persistent no matter where you put the task record.)

Setting this field to 0 is one way of disabling a task. A more common approach is to remove the VBL task structure from its queue by calling VRemove or SlotVRemove, although this should not be done by the task itself.

vblPhase The phase count of the VBL task.

In most cases, you can set this field to 0 . However, if you install multiple tasks with the same vblCount at the same time, you can assign them different vblPhase values so that the tasks are not executed during the same interrupt. The value in the vblPhase field must be less than the value in the vblCount field.

Installing a VBL Task

For any particular VBL task, you must first decide whether to install it as a system-based VBL task or as a slot-based VBL task. The following considerations apply:

- ***Slot-Based VBL Tasks.*** You need to install a task as a slot-based VBL task only if the execution of the task needs to be synchronised with the retrace rate of a particular external monitor. This will be the case, for example, if you want the repetitive re-drawing of a moving image to occur only during that particular monitor's vertical blanking period.

- ***System-Based VBL Tasks.*** If the task performs no processing likely to affect the appearance of the screen, and no processing that depends on the state of an external monitor, you can install it as a system-based VBL task.

The next steps are to define the VBL task itself (so as be able to assign its address to the vblAddr field of the VBL task structure) and, in the case of slot-based VBL tasks, call LMGetMainDevice and GetDCtlEntry to find the slot number of the video device to whose retrace the VBL task is to be synchronised. The final step is to fill in a VBL task structure and install it into the appropriate queue.

VBL Task Structures Access — 680x0 Code

Recall that, if a VBL task is to be executed recurrently, it must reset the vblCount field of the VBL task structure each time it is executed. A repetitive VBL task must therefore be able to access its VBL task structure so that it can reset the vblCount field.

When the Vertical Retrace Manager executes the VBL task in a 680x0 environment, it places the address of the VBL task into the A0 register. The following defines an in-line function which moves that value onto the stack:

```
pascal SInt32  GetVBLRec(void) = 0x2E88;
```

This in-line function, which returns a long integer specifying the address of the VBL task structure, should be called only from a VBL task. It will not work if called from the main program. In addition, the call should be the first line of your VBL task, because other processing could change the value in A0.

VBL Task Structure Access — PowerPC Code

In the PowerPC environment, the address of the VBL task structure is passed to the to the VBL task as an explicit parameter.

Accessing Application Global Variables - 680x0 Code

Recall from Chapter 1 that the boundary between the current application's global variables and its application parameters are stored in the 680x0 microprocessor's A5 register. Since all 680x0 applications share this register, the Process Manager keeps track of the address of your application's A5 world when a major or minor switch yields control of the microprocessor to another application. Then, when your application regains access to the CPU, the Process Manager restores that address to the A5 register.

Because VBL tasks are interrupt functions, they could well execute when the value in the A5 register does not point to your application's A5 world. As a result, if you need to access your application's global variables in a VBL task, you need to set the A5 register to its correct value when your VBL task begins executing and restore the previous value upon exit.

To achieve this, your 680x0 application should save its A5 using SetCurrentA5. Then, at interrupt time, the VBL task can begin by calling SetA5 to, firstly, set the A5 register to this saved value and, secondly, save the value that was in the A5 register immediately prior to the call. The VBL task should end with another call to SetA5, this time to restore the initial value.

The only memory location that a VBL task has access to is the address of the VBL task structure. Accordingly, if your application stores its A5 directly following the VBL task structure, it can locate this value by first locating the VBL task structure. To store the A5 value directly following the VBL task structure, define a new data type whose first field contains the VBL task structure and whose second field will hold the value in the A5 register retrieved by a call to SetCurrentA5:

```
typedef struct
{
   VBLTask     vblTaskStruc;      // The VBL task structure.
   long        vblA5              // Saved value of A5.
} VBLStructure, *VBLStructurePtr;
```

You can think of this new data type as an expanded VBL task structure.

Accessing Application Global Variables - PowerPC Code

Setting and restoring the A5 register has no relevance in PowerPC code. In the PowerPC environment, the table of contents register always points to the table of contents for the currently executing code, through which the application's global variables can be addressed. As a result, your application's global variables are transparently available to any code compiled into your application.

Ensuring Compatibility with the Operating Environment

If your application is to run successfully in the software and hardware environments that may be present in a wide range of Macintosh models, it must be able to acquire information about a number of machine-dependent features and, where appropriate, act on that information.

Getting Operating Environment Information - The Gestalt Function

The Gestalt function may be used to acquire a wide range of information about the operating environment.

```
OSErr  Gestalt(OSType selector,long *response);
```

selector	Selector code.
response	4-byte return result which provides the requested information. When all four bytes are not needed, the result is expressed in the low-order byte.
Returns:	Error code. (0 = no error.)

The types of information capable of being retrieved by Gestalt are as follows:

- The type of machine.
- The version of the System file currently running.
- The type of CPU.
- The type of keyboard attached to the machine.
- The type of floating-point unit (FPU) installed, if any.
- The type of memory management unit (MMU).
- The size of the available RAM.
- The amount of available virtual memory.
- The versions and features of various drivers and managers.

Gestalt Selectors

To use Gestalt, you pass it a **selector**, which specifies exactly what information your application is seeking. Of those selectors which are pre-defined by the Gestalt Manager, there are two sub-types:

- ***Environmental Selectors.*** Environmental selectors are those which return information about the existence, or otherwise, of a feature. This information can be used by your application to guide its actions. Some examples of the many available environmental selectors, and the information returned in the reponse parameter, are as follows:

Selector	***Information Returned***
gestaltFPUType	FPU type.
gestaltKeyboardType	Keyboard type.
gestaltLogicalRAMSize	Logical RAM size.
gestaltPhysicalRAMSize	Physical RAM size.
gestaltQuickdrawVersion	QuickDraw version.
gestaltTextEditVersion	TextEdit version.

- ***Informational Selectors.*** Informational selectors are those which provide information which should be used for the user's enlightenment only. This information should never be used as proof positive of some feature's existence, nor should it be used to guide your application's actions. Some example of informational selectors, and the information they return, are as follows:

Selector	***Information Returned***
gestaltMachineType	Machine type.
gestaltROMVersion	ROM version.
gestaltSystemVersion	System file version.

Gestalt Responses

In almost all cases, the last few characters in the selector's name form a suffix which indicates the type of value that will be returned in the response parameter. The following shows the meaningful suffixes:

Suffix	Returned Value
Attr	A range of 32 bits, the meaning of which must be determined by comparison with a list of constants.
Count	A number indicating how many of the indicated type of items exist.
Size	A size, usually in bytes.
Table	Base address of a table.
Type	An index describing a particular type of feature.
Version	A version number. Implied decimal points may separate digits of the returned value. For example, a value of 0x0750 returned in response to the gestaltSystemVersion selector means that system software version 7.5.0 is present.

Using Gestalt — Examples

The header file Gestalt.h defines and describes Gestalt Manager selectors, together with the many constants which may be used to test the response parameter.

Example 1

For example, when Gestalt is used to check whether Version 1.3 or later of Color QuickDraw is present, the value returned in the response parameter may be compared with gestalt32BitQD13 as follows:

```
OSErr	osErr
SInt32	response;
Boolean	colorQuickDrawVers13Present = true;

osErr = Gestalt(gestaltQuickdrawVersion,&response);
if(osErr == noErr)
{
  if(response < gestalt32BitQD13)
    colorQuickDrawVers13Present = false;
}
```

Example 2

Many constants in Gestalt.h represent bit numbers. In this example, the value returned in the response parameter is tested to determine whether bit number 5 (gestaltHasSoundInputDevice) is set:

```
OSErr	osErr;
SInt32	response;
Boolean	hasSoundInputDevice = false;

osErr = Gestalt(gestaltSoundAttr,&response);
if(osErr == noErr)
  gHasSoundInputDevice = BitTst(&response,31 - gestaltHasSoundInputDevice);
```

Note that the function BitTst is used to determine whether the specified bit is set. Bit numbering with BitTst is the opposite of the usual MC680x0 numbering scheme used by Gestalt. Thus the bit to be tested must be subtracted from 31. This is illustrated in the following:

```
Bit numbering as used in BitTst
0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
Bit as numbered in MC69000 CPU operations, and used by Gestalt
15 14 13 12 11 10 9  8  7  6  5  4  3  2  1  0  15 14 13 12 11 10 9  8  7  6  5  4  3  2  1  0

gestaltHasSoundInputDevice = 5
31 - 5 = 26
```

Code Segmentation and Heap Space Optimisation - 680x0 Code

680x0 Macintosh programs may be divided into several **segments**. The Macintosh system software limits segments to 32K; accordingly, if you are writing a large program, you must segment your code.

Observing the 32K limit is, however, not the only reason for segmenting your 680x0 code. Segments equate, in the built application, to units of executable code which are stored in resources of type 'CODE' and which are loaded into your application's heap as relocatable blocks. Because these resources are loaded into memory only when required, and because your application can cause them to be marked as purgeable when no longer needed, segmentation allows you to optimise your 680x0 application's heap space. Put another way, segmentation allows you to provide the user with the maximum possible heap space to accommodate the windows and user data, etc., created while the 680x0 application is running.

The main segment (that is, the segment containing the main function) is loaded and locked by the system when the application is launched. Thereafter, when the application makes a call to a function in one of the remaining segments, the Segment Loader, with no help from the application, automatically loads that segment, moves it high in the application's heap, locks it, and passes control to the called function.

Ultimately, of course, all code segments will be brought into memory and locked, creating the same memory-hogging situation as would obtain if the application had not been segmented. To prevent that situation, your application should, at the appropriate time, unlock these blocks and make them purgeable. Note that this applies to all but the main code segment, which must never be unlocked or made purgeable. The following describes an appropriate methodology for unlocking and marking as purgeable the other code segments of your application:

- Create a new **stub**, or “do nothing” function, for each of the code segments you want to unload. For example, this is a stub for a code segment called updateSegment:

  ```
  void  updateSegment(void) {}
  ```

- Include each stub in its associated code segment.

- Write a function called, say, doUnloadSegments which calls the Segment Loader function UnloadSeg for each of the stubs. The following is an example:

  ```
  void  doUnloadSegments(void)
  {
     UnloadSeg(updateSegment);
     UnloadSeg(activateSegment);
     // Other UnloadSeg calls here as required.
  }
  ```

  Note that each UnloadSeg call looks up the code segment that contains the stub function in its input parameter, unlocks that segment, and makes it purgeable. Note also that you could pass any of the segment's functions as the parameter to the UnloadSeg call; however, it is preferable to use stubs dedicated to this purpose because the other functions in the segment could well be moved to another segment during future updating of the code.

- Place the doUnloadSegments function in the main code segment and call it at the bottom of the main event loop (which should also be located in the main code segment) so that all code segments specified in the function will be unlocked and marked as purgeable after a received event has been handled to completion. The following is an example:

  ```
  void  main(void)
  ```

```
{
   ...
   while(!gDone)
   {
      if(WaitNextEvent(everyEvent,&eventRec,MAXLONG,NULL))
         doEvents(&eventRec);

      doUnloadSegments();
   }
}
```

One or more of the unlocked and purgeable code segments may then be purged by the Memory Manager if this becomes necessary in order to satisfy a memory allocation request. When a call is subsequently made to a function contained in one of the purged segments, the Segment Loader once again loads that segment into your application's heap as a relocatable block.

PowerPC Considerations

There is no need to include conditional compilation directives in source code containing segmentation directives before that code is compiled for the PowerPC. Compilers which produce PowerPC code ignore segmentation directives, and any calls to the Segment Managers's `UnloadSeg` function are simply ignored.

Main Notification Manager Data Types and Functions

Data Types

Notification Structure

```
struct NMRec
{
   QElemPtr     qLink;          // Next queue entry.
   short        qType;          // Queue type.
   short        nmFlags;        // (Reserved.)
   long         nmPrivate;      // (Reserved.)
   short        nmReserved;     // (Reserved.)
   short        nmMark;         // Item to mark in Apple menu.
   Handle       nmIcon;         // Handle to small icon.
   Handle       nmSound;        // Handle to sound structure.
   StringPtr    nmStr;          // String to appear in alert.
   NMUPP        nmResp;         // Pointer to response function.
   long         nmRefCon;       // For application use.
};
typedef struct NMRec NMRec;
typedef NMRec *NMRecPtr;
```

Functions

Add Notification Request to the Notification Queue

```
OSErr    NMInstall(NMRecPtr nmReqPtr);
```

Remove Notification Request from the Notification Queue

```
OSErr    NMRemove(NMRecPtr nmReqPtr);
```

Relevant Process Manager Data Types and Functions

Data Types

Process Serial Number

```
struct ProcessSerialNumber
{
    unsigned long          highLongOfPSN;
    unsigned long          lowLongOfPSN;
};
```

Functions

Get Process Serial Number of a Particular Process

```
OSErr      GetCurrentProcess(ProcessSerialNumber *PSN);
```

Get Process Serial Number of Foreground Process

```
OSErr      GetFrontProcess(ProcessSerialNumber *PSN);
```

Compare Two Process Serial Numbers

```
OSErr      SameProcess(const ProcessSerialNumber *PSN1,const ProcessSerialNumber *PSN2,
           Boolean *result);
```

Relevant Event Manager Data Types and Functions

Data Types

QHdr (Defines the Queue Header)

```
struct QHdr
{
    short          qFlags;
    QElemPtr       qHead;
    QElemPtr       qTail;
};
typedef struct QHdr QHdr;
typedef QHdr *QHdrPtr;
```

QElem

```
struct QElem
{
    QElemPtr       qLink;
    short          qType;
    short          qData[1];
};
typedef struct QElem QElem;
typedef QElem *QElemPtr;
```

EvQEl (Defines an Entry in the Operating System Event Queue)

```
struct EvQEl
{
    QElemPtr                qLink;
    short                   qType;
    EventKind               evtQWhat;
    Uint32                  evtQMessage;
    Uint32                  evtQWhen;
    Point                   evtQWhere;
    EventModifiers          evtQModifiers;
};
typedef struct EvQEl EvQEl;
typedef EvQEl *EvQElPtr;
```

Functions

Get Address of Event Queue Header

```
QHdrPtr    LMGetEventQueue(void);
```

Relevant Color Picker Utilities Function

```
Boolean        GetColor(Point where,ConstStr255Param prompt,const RGBColor *inColor,
               RGBColor *outColor)
```

Relevant QuickDraw Constants and Functions

Constants

Flag Bits for gdFlags Field of GDevice Structure

```
mainScreen        = 11   // Graphics device is main screen.
screenDevice      = 13   // Graphics device is a screen device.
screenActive      = 15   // Graphics device is current device.
```

Functions

Getting Available Graphics Devices

```
GDHandle     LMGetDeviceList(void);
GDHandle     LMGetMainDevice(void);
GDHandle     GetNextDevice(void);
```

Determining the Characteristics of a Video Device

```
void         DeviceLoop(RgnHandle drawingRgn,DeviceLoopDrawingUP drawingProc,
             long userData,DeviceLoopFlags flags);
Boolean      TestDeviceAttribute(GDHandle gdh,short attribute);
```

Getting the Intersection Between Two Rectangles and Determining the Overlap

```
Boolean      SectRect(Rect rect1,Rect rect2,Rect resultRect);
```

Vertical Retrace Manager Data Types and Functions

Data Types

VBL Task Structure

```
struct VBLTask
{
   QElemPtr    qLink;
   short       qType;
   VBLUPP      vblAddr;
   short       vblCount;
   short       vblPhase;
};

typedef struct VBLTask VBLTask,*VBLTaskPtr;
```

Functions

Slot-Based Installation and Removal Routines

```
OSErr      SlotVInstall(QElemPtr vblBlockPtr,short theSlot);
OSErr      SlotVRemove(QElemPtr vblBlockPtr,short theSlot);
```

System-Based Installation and Removal Routines

```
OSErr      VInstall(QElemPtr vblTaskPtr);
OSErr      VRemove(QElemPtr vblTaskPtr);
```

Utility Routines

```
OSErr      AttachVBL(short theSlot);
OSErr      DoVBLTask(short theSlot);
QHdrPtr    GetVBLQHdr(void);
```

Relevant Gestalt Manager Function

```
OSErr   Gestalt(OSType selector,long *response);
```

Relevant Segment Loader Functions

Unlock Code Segments and Make Purgeable

```
void  UnloadSeg(void * routineAddr);
```

Terminate Caller, Release Heap, and Launch Finder

```
void ExitToShell(void);
```

Demonstration Program

```
// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊
// Miscellany.h
// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊
//
// This program demonstrates:
//
// •   The use of the Notification Manager to allow an application running in the
//     background to communicate with the foreground application.
//
// •   The use of the determinate progress indicator control to show progress during a
//     time-consuming operation, together with scanning the event queue for Command-period
//     key-down events for the purpose of terminating the lengthy operation before it
//     concludes of its own accord.
//
// •   Image drawing optimisation and window zooming in a multi-monitors environment.
//
// •   The use of the Color Picker to solicit a choice of colour from the user.
//
// •   Slot-based VBL tasks.

// •   The use of stubs in 68K code segments, together with a function which uses those
//     stubs    to unlock code segments and make them purgeable.
//
// The program utilises the following resources:
//
// •   An 'MBAR' resource, and 'MENU' resources for Apple, File, Edit and Demonstration
//     menus (preload, non-purgeable).
//
// •   A 'WIND' resource (purgeable) (initially visible) for a window in which graphics
//     and information relevant to the demonstrations is displayed.
//
```

```
// •   A 'DLOG' resource (purgeable), and associated 'DITL', 'dlgx', and 'dftb' resources
//      (purgeable), for a dialog box in which the progress indicator is displayed.
//
// •   'CNTL' resources (purgeable) for the progress indicator dialog.
//
// •   'icn#', 'ics4', and 'ics8' resources (non-purgeable) which contain the application
//      icon shown in the Application menu during the Notification Manager demonstration.
//
// •   A 'snd ' resource (non-purgeable) used in the Notification Manager demonstration.
//
// •   A 'STR ' resource (non-purgeable) containing the text displayed in the alert box
//      invoked by the Notification Manager.
//
// •   A 'STR#' resource (purgeable) containing the label and narrative strings for the
//      notification-related alert displayed by Miscellany.
//
// •   A 'PICT' resource (non-purgeable) used in the slot-based VBL task demonstration.
//
// •   A 'SIZE' resource with the acceptSuspendResumeEvents, doesActivateOnFGSwitch,
//      canBackgound, and is32BitCompatible flags set.
//
// Miscellany source code is contained in six files.  In the 68K project, each source
// code file is assigned to a different segment.  (Note that this small program does not
// really require such segmentation.  The code is segmented only to facilitate the
// demonstration of the Segment Loader aspects, which apply only to the 68K version.)
//
// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊

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

#include <Appearance.h>
#include <ColorPicker.h>
#include <ControlDefinitions.h>
#include <Devices.h>
#include <LowMem.h>
#include <Resources.h>
#include <Retrace.h>
#include <SegLoad.h>
#include <Sound.h>
#include <ToolUtils.h>

// ..................................................................................................................................................................
............................................ defines

#define mApple                  128
#define  iAbout                1
#define mFile                   129
#define  iQuit                 11
#define mDemonstration         131
#define  iNotification        1
#define  iProgress             2
#define  iColourPicker        3
#define  iMultiMonitors       4
#define  iSlotVertBlank       5
#define rMenubar               128
#define rWindow                 128
#define rAlert                 128
#define rDialog                 128
#define  iProgressIndicator   1
#define rIconFamily            128
#define rBarkSound             8192
#define rString                128
#define rAlertStrings          128
#define  indexLabel            1
#define  indexNarrative        2
#define rPicture               128
#define topLeft(r)            (((Point *) &(r))[0])
#define botRight(r)            (((Point *) &(r))[1])

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

typedef struct
{
  VBLTask       vblTaskStruc;
```

```
    SInt32       thisApplicationsA5;
    Boolean        inVBlankPeriod;
} VBLStructure, *VBLStructurePtr;

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

void            main                                       (void);
void            doInitManagers                         (void);
void            doEvents                                   (EventRecord *);
void            doMenuChoice                             (SInt32 menuChoice);
void            unloadSegments                          (void);

void            notificationSegment                    (void);
void            doSetUpNotification                     (void);
void            doPrepareNotificationStructure (void);
void            doIdle                                        (void);
void            doOSEvent                                     (EventRecord *);
void            doDisplayMessageToUser              (void);

void            progressBarSegment                      (void);
void            doProgressIndicator                     (void);
Boolean            doCheckForCancel                           (DialogPtr);

void            multiMonitorSegment                     (void);
pascal void   doDeviceLoopDraw                       (SInt16,SInt16,GDHandle,SInt32);
void            doZoomWindowMultiMonitors            (WindowPtr,SInt16);
void            doRedoWindowContent                   (WindowPtr);

void            colourPickerSegment                     (void);
void            doColourPicker                          (void);
void            doDrawColourPickerChoice            (void);
char            *doDecimalToHexadecimal                (UInt16 n);

void            doSlotVBLTask                            (void);
OSErr             doInstallSlotVBLTask                 (void);
#if TARGET_CPU_68K
void            theSlotVBLTask                          (void);
#else
void            theSlotVBLTask                          (VBLStructurePtr);
#endif
void            doStopSlotVBLTask                        (void);

// ........................................................................................................................................................................ in-
line glue for GetVBLRec

#if TARGET_CPU_68K
pascal SInt32  GetVBLRec (void) = 0x2E88;
#endif

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊
// Miscellany.c
// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊

#include "Miscellany.h"

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

DeviceLoopDrawingUPP    doDeviceLoopDrawUPP;
Boolean                            gDone;
WindowPtr                          gWindowPtr;
ProcessSerialNumber       gProcessSerNum;
Rect                              gMultiMonDragBounds, gMultiMonGrowBounds;
Boolean                            gMultiMonitorsDrawDemo = false;
Boolean                            gColourPickerDemo  = false;
RGBColor                        gWhiteColour = { 0xFFFF, 0xFFFF, 0xFFFF };
RGBColor                        gBlueColour  = { 0x4444, 0x4444, 0x9999 };

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ main

void  main(void)
{
    Handle             menubarHdl;
    MenuHandle     menuHdl;
    RgnHandle           grayRgnHdl;
    EventRecord     eventStructure;
    //
..........................................................................................................................................................................................
... initialise managers

    doInitManagers();

    // ....................................................................................................................................................................
create routine descriptor

    doDeviceLoopDrawUPP = NewDeviceLoopDrawingProc((ProcPtr) doDeviceLoopDraw);

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

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

    menuHdl = GetMenuHandle(mApple);
    if(menuHdl == NULL)
        ExitToShell();
    else
        AppendResMenu(menuHdl,'DRVR');

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

    if(!(gWindowPtr = GetNewCWindow(rWindow,NULL,(WindowPtr)-1)))
        ExitToShell();

    SetPort(gWindowPtr);
    TextSize(10);

    // .................................................................................................................... get process serial number of this
process

    GetCurrentProcess(&gProcessSerNum);

    // ..................................................................................... get window drag and sizing limits for multiple monitors

    grayRgnHdl = LMGetGrayRgn();
    gMultiMonDragBounds = (*grayRgnHdl)->rgnBBox;
    SetRect(&gMultiMonGrowBounds,445,302,
                ((gMultiMonDragBounds.right) - (gMultiMonDragBounds.left)),
                ((gMultiMonDragBounds.bottom) - (gMultiMonDragBounds.top)));

    //
..........................................................................................................................................................................................
.............. enter eventLoop

    gDone = false;

    while(!gDone)
    {
        if(WaitNextEvent(everyEvent,&eventStructure,0,NULL))
            doEvents(&eventStructure);
        else
            doIdle();

        unloadSegments();
    }
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doInitManagers

void  doInitManagers(void)
{
    MaxApplZone();
    MoreMasters();

    InitGraf(&qd.thePort);
    InitFonts();
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs(NULL);

    InitCursor();
    FlushEvents(everyEvent,0);

    RegisterAppearanceClient();
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doEvents

void  doEvents(EventRecord *eventStrucPtr)
{
    SInt16       partCode;
    WindowPtr    windowPtr;
    SInt8        charCode;
    SInt32       newSize, userData;

    switch(eventStrucPtr->what)
    {
        case mouseDown:
            partCode = FindWindow(eventStrucPtr->where,&windowPtr);

            switch(partCode)
            {
                case inMenuBar:
                    doMenuChoice(MenuSelect(eventStrucPtr->where));
                    break;

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

                case inDrag:
                    DragWindow(windowPtr,eventStrucPtr->where,&gMultiMonDragBounds);
                    break;

                case inGrow:
                    newSize = GrowWindow(windowPtr,eventStrucPtr->where,&gMultiMonGrowBounds);
                    if(newSize != 0)
                        SizeWindow(windowPtr,LoWord(newSize),HiWord(newSize),true);
                    InvalRect(&windowPtr->portRect);
                    break;

                case inZoomIn:
                case inZoomOut:
                    if(TrackBox(windowPtr,eventStrucPtr->where,partCode))
                        doZoomWindowMultiMonitors(windowPtr,partCode);
                    break;
            }
            break;

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

        case updateEvt:
            windowPtr = (WindowPtr) eventStrucPtr->message;

            BeginUpdate(windowPtr);

            if(gMultiMonitorsDrawDemo)
            {
                RGBBackColor(&gWhiteColour);
                userData = (SInt32) windowPtr;
                DeviceLoop(windowPtr->visRgn,doDeviceLoopDrawUPP,userData,0);
            }
            else if(gColourPickerDemo )
            {
                RGBBackColor(&gBlueColour);
                EraseRect(&windowPtr->portRect);
                doDrawColourPickerChoice();
            }
            else
            {
                RGBBackColor(&gBlueColour);
            EraseRect(&windowPtr->portRect);
        }

        EndUpdate(windowPtr);
        break;

      case osEvt:
        doOSEvent(eventStrucPtr);
        HiliteMenu(0);
        break;
  }
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doMenuChoice

void  doMenuChoice(SInt32 menuChoice)
{
  SInt16  menuID, menuItem;
  Str255  itemName;
  SInt16  daDriverRefNum;

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

  if(menuID == 0)
    return;

  switch(menuID)
  {
    case mApple:
      if(menuItem == iAbout)
        SysBeep(10);
      else
      {
        GetMenuItemText(GetMenuHandle(mApple),menuItem,itemName);
        daDriverRefNum = OpenDeskAcc(itemName);
      }
      break;

    case mFile:
      if(menuItem == iQuit)
        ExitToShell();
      break;

    case mDemonstration:

      gMultiMonitorsDrawDemo = gColourPickerDemo = false;

      switch(menuItem)
      {
        case iNotification:
          RGBBackColor(&gBlueColour);
          EraseRect(&gWindowPtr->portRect);
          doSetUpNotification();
          break;

        case iProgress:
          RGBBackColor(&gBlueColour);
          EraseRect(&gWindowPtr->portRect);
          doProgressIndicator();
          break;

        case iColourPicker:
          gColourPickerDemo  = true;
          doColourPicker();
          break;

        case iMultiMonitors:
          gMultiMonitorsDrawDemo = true;
          InvalRect(&gWindowPtr->portRect);
          break;

        case iSlotVertBlank:
          RGBBackColor(&gBlueColour);
          EraseRect(&gWindowPtr->portRect);
          doSlotVBLTask();
          break;
      }
```

```
            break;
    }

    HiliteMenu(0);
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ unloadSegments

void  unloadSegments(void)
{
    UnloadSeg(notificationSegment);
    UnloadSeg(progressBarSegment);
    UnloadSeg(multiMonitorSegment);
    UnloadSeg(colourPickerSegment);
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊
// Notification.c
// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊

#include "Miscellany.h"

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

NMRec                                   gNotificationStructure;
long                                 gStartingTickCount;
Boolean                                 gNotificationDemoInvoked;
Boolean                                 gNotificationInQueue;
Boolean                                 gInBackground;
extern WindowPtr                  gWindowPtr;
extern ProcessSerialNumber gProcessSerNum;
extern RGBColor                      gWhiteColour;
extern RGBColor                      gBlueColour;

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ notificationSegment

void  notificationSegment(void) {}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doSetUpNotification

void  doSetUpNotification(void)
{
    doPrepareNotificationStructure();
    gNotificationDemoInvoked = true;

    gStartingTickCount = TickCount();

    RGBForeColor(&gWhiteColour);
    MoveTo(10,279);
    DrawString("\pPlease click on the desktop now to make the Finder ");
    DrawString("\pthe frontmost application.");
    MoveTo(10,292);
    DrawString("\p(This application will post a notification 10 seconds from now.)");
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doPrepareNotificationStructure

void  doPrepareNotificationStructure(void)
{
    Handle          iconSuiteHdl;
    Handle          soundHdl;
    StringHandle    stringHdl;

    GetIconSuite(&iconSuiteHdl,rIconFamily,kSelectorAllSmallData);
    soundHdl = GetResource('snd ',rBarkSound);
    stringHdl = GetString(rString);

    gNotificationStructure.qType        = nmType;
    gNotificationStructure.nmMark       = 1;
    gNotificationStructure.nmIcon       = iconSuiteHdl;
    gNotificationStructure.nmSound      = soundHdl;
    gNotificationStructure.nmStr        = *stringHdl;
    gNotificationStructure.nmResp       = NULL;
    gNotificationStructure.nmRefCon     = 0;
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doIdle
```

```
void  doIdle(void)
{
   ProcessSerialNumber   frontProcessSerNum;
   Boolean                       isSameProcess;

   if(gNotificationDemoInvoked)
   {
      if(TickCount() > gStartingTickCount + 600)
      {
         GetFrontProcess(&frontProcessSerNum);
         SameProcess(&frontProcessSerNum,&gProcessSerNum,&isSameProcess);

         if(!isSameProcess)
         {
            NMInstall(&gNotificationStructure);
            gNotificationDemoInvoked = false;
            gNotificationInQueue = true;
         }
         else
         {
            doDisplayMessageToUser();
            gNotificationDemoInvoked = false;
         }

         EraseRect(&gWindowPtr->portRect);
      }
   }
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doOSEvent

void  doOSEvent(EventRecord *eventStrucPtr)
{
   switch((eventStrucPtr->message >> 24) & 0x000000FF)
   {
      case suspendResumeMessage:
         gInBackground = (eventStrucPtr->message & resumeFlag) == 0;
         if((!gInBackground) && gNotificationInQueue)
            doDisplayMessageToUser();
         break;
   }
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doDisplayMessageToUser

void  doDisplayMessageToUser(void)
{
   AlertStdAlertParamRec paramRec;
   Str255                        labelText;
   Str255                        narrativeText;
   SInt16                        itemHit;

   if(gNotificationInQueue)
   {
      NMRemove(&gNotificationStructure);
      gNotificationInQueue = false;
   }

   EraseRect(&gWindowPtr->portRect);

   paramRec.movable             = true;
   paramRec.helpButton          = false;
   paramRec.filterProc          = NULL;
   paramRec.defaultText         = (StringPtr) kAlertDefaultOKText;
   paramRec.cancelText          = NULL;
   paramRec.otherText           = NULL;
   paramRec.defaultButton       = kAlertStdAlertOKButton;
   paramRec.cancelButton        = 0;
   paramRec.position            = kWindowDefaultPosition;

   GetIndString(labelText,rAlertStrings,indexLabel);
   GetIndString(narrativeText,rAlertStrings,indexNarrative);

   StandardAlert(kAlertNoteAlert,labelText,narrativeText,&paramRec,&itemHit);

   DisposeIconSuite(gNotificationStructure.nmIcon,false);
   ReleaseResource(gNotificationStructure.nmSound);
   ReleaseResource((Handle) gNotificationStructure.nmStr);
```

```
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊
// ProgressIndicator.c
// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊

#include "Miscellany.h"

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

extern WindowPtr gWindowPtr;
extern RGBColor       gWhiteColour;

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ progressBarSegment

void  progressBarSegment(void) {}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doProgressBar

void  doProgressIndicator(void)
{
   DialogPtr          dialogPtr;
   ControlHandle progressBarHdl;
   SInt16             statusMax, statusCurrent;
   SInt16             a, b, c;
   Handle             soundHdl;
   Rect               theRect;
   RGBColor           redColour = { 0xFFFF, 0x0000, 0x0000 };

   if(!(dialogPtr = GetNewDialog(rDialog,NULL,(WindowPtr) -1)))
      ExitToShell();

   SetPort(dialogPtr);
   UpdateControls(dialogPtr,((GrafPtr) dialogPtr)->visRgn);
   SetPort(gWindowPtr);

   GetDialogItemAsControl(dialogPtr,iProgressIndicator,&progressBarHdl);

   statusMax = 3456;
   statusCurrent = 0;
   SetControlMaximum(progressBarHdl,statusMax);

   for(a=0;a<9;a++)
   {
      for(b=8;b<423;b+=18)
      {
         for(c=8;c<286;c+=18)
         {
            if(doCheckForCancel(dialogPtr))
            {
               soundHdl = GetResource('snd ',rBarkSound);
               SndPlay(NULL,(SndListHandle) soundHdl,false);
               ReleaseResource(soundHdl);
               DisposeDialog(dialogPtr);

               EraseRect(&gWindowPtr->portRect);
               MoveTo(10,292);
               RGBForeColor(&gWhiteColour);
               DrawString("\pOperation cancelled at user request");

               return;
            }

            SetRect(&theRect,b+a,c+a,b+17-a,c+17-a);
            if(a < 3)                              RGBForeColor(&gWhiteColour);
            else if(a > 2 && a < 6)     RGBForeColor(&redColour);
            else if(a > 5)                   RGBForeColor(&gWhiteColour);
            FrameRect(&theRect);

            SetControlValue(progressBarHdl,statusCurrent++);
         }
      }
   }

   DisposeDialog(dialogPtr);
   EraseRect(&gWindowPtr->portRect);
   MoveTo(10,292);
```

```
    RGBForeColor(&gWhiteColour);
    DrawString("\pOperation completed");
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doCheckForCancel

Boolean  doCheckForCancel(DialogPtr dialogPtr)
{
    GrafPtr     oldPort;
    Boolean     foundCommandPeriod;
    QHdrPtr     eventQHdrPtr;
    EvQElPtr    eventQElPtr;
    SInt32      charCode;
    SInt32      commandKeyDown;

    GetPort(&oldPort);
    SetPort(dialogPtr);

    foundCommandPeriod = false;

    eventQHdrPtr = LMGetEventQueue();
    eventQElPtr = (EvQElPtr) eventQHdrPtr->qHead;

    while(eventQElPtr && !foundCommandPeriod)
    {
        if(eventQElPtr->evtQWhat == keyDown)
        {
            charCode = eventQElPtr->evtQMessage & charCodeMask;
            commandKeyDown = eventQElPtr->evtQModifiers & cmdKey;

            if(commandKeyDown)
                if(charCode == 0x2e)
                    foundCommandPeriod = true;
        }

        if(!foundCommandPeriod)
            eventQElPtr = (EvQElPtr) eventQElPtr->qLink;
    }

    SetPort(oldPort);

    return(foundCommandPeriod);
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊
// ColourPicker.c
// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊

#include "Miscellany.h"

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

RGBColor            gInColour = { 0xCCCC, 0x0000, 0x0000 };
RGBColor            gOutColour;
Boolean             gColorPickerButton;
extern WindowPtr    gWindowPtr;
extern RGBColor     gWhiteColour;
extern RGBColor     gBlueColour;

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ colourPickerSegment

void  colourPickerSegment(void) {}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doColourPicker

void  doColourPicker(void)
{
    Rect        theRect;
    Point       where;
    Str255      prompt = "\pChoose a rectangle colour:";

    theRect = gWindowPtr->portRect;
    RGBBackColor(&gBlueColour);
    EraseRect(&theRect);
    InsetRect(&theRect,55,55);
    RGBForeColor(&gInColour);
    PaintRect(&theRect);
    where.v = where.h = 0;

    gColorPickerButton = GetColor(where,prompt,&gInColour,&gOutColour);

    InvalRect(&gWindowPtr->portRect);
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doDrawColorPickerChoice

void  doDrawColourPickerChoice(void)
{
    Rect theRect;
    char *cString;

    theRect = gWindowPtr->portRect;
    InsetRect(&theRect,55,55);

    if(gColorPickerButton)
    {
        RGBForeColor(&gOutColour);
        PaintRect(&theRect);

        RGBForeColor(&gWhiteColour);

        MoveTo(55,22);
        DrawString("\pRequested Red Value: ");
        cString = doDecimalToHexadecimal(gOutColour.red);
        MoveTo(170,22);
        DrawText(cString,0,6);

        MoveTo(55,35);
        DrawString("\pRequested Green Value: ");
        cString = doDecimalToHexadecimal(gOutColour.green);
        MoveTo(170,35);
        DrawText(cString,0,6);

        MoveTo(55,48);
        DrawString("\pRequested Blue Value: ");
        cString = doDecimalToHexadecimal(gOutColour.blue);
        MoveTo(170,48);
        DrawText(cString,0,6);
    }
    else
    {
        RGBForeColor(&gInColour);
        PaintRect(&theRect);

        RGBForeColor(&gWhiteColour);
        MoveTo(55,48);
        DrawString("\pCancel button was clicked.");
    }
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doDecimalToHexadecimal

char  *doDecimalToHexadecimal(UInt16 decimalNumber)
{
    static char cString[] = "0xXXXX";
    char          *hexCharas = "0123456789ABCDEF";
    SInt16         a;

    for (a=0;a<4;decimalNumber >>= 4,++a)
        cString[5 - a] = hexCharas[decimalNumber & 0xF];

    return cString;
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊
// MultiMonitor.c
// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊

#include "Miscellany.h"

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ multiMonitorSegment

void  multiMonitorSegment(void) {}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doDeviceLoopDraw
```

```
pascal void  doDeviceLoopDraw(SInt16 depth,SInt16 deviceFlags,GDHandle targetDeviceHdl,
                                                   SInt32 userData)
{
   RGBColor  oldForeColour;
   WindowPtr windowPtr;
   Rect      theRect;
   RGBColor  greenColour  = { 0x0000, 0xAAAA, 0x1111 };
   RGBColor  redColour    = { 0xAAAA, 0x4444, 0x3333 };
   RGBColor  blueColour   = { 0x5555, 0x4444, 0xFFFF };
   RGBColor  ltGrayColour = { 0xDDDD, 0xDDDD, 0xDDDD };
   RGBColor  grayColour   = { 0x9999, 0x9999, 0x9999 };
   RGBColor  dkGrayColour = { 0x4444, 0x4444, 0x4444 };

   GetForeColor(&oldForeColour);

   windowPtr = (WindowPtr) userData;
   theRect = windowPtr->portRect;
   EraseRect(&windowPtr->portRect);

   if(BitTst(&deviceFlags,15 - gdDevType))
   {
      InsetRect(&theRect,50,50);
      RGBForeColor(&greenColour);
      PaintRect(&theRect);
      InsetRect(&theRect,40,40);
      RGBForeColor(&redColour);
      PaintRect(&theRect);
      InsetRect(&theRect,40,40);
      RGBForeColor(&blueColour);
      PaintRect(&theRect);
   }
   else
   {
      InsetRect(&theRect,50,50);
      RGBForeColor(&ltGrayColour);
      PaintRect(&theRect);
      InsetRect(&theRect,40,40);
      RGBForeColor(&grayColour);
      PaintRect(&theRect);
      InsetRect(&theRect,40,40);
      RGBForeColor(&dkGrayColour);
      PaintRect(&theRect);
   }

   RGBForeColor(&oldForeColour);
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doZoomWindow

void doZoomWindowMultiMonitors(WindowPtr windowPtr,SInt16 zoomInOrOut)
{
   GrafPtr     oldPort;
   Rect        windRect, intersectRect, zoomRect;
   SInt16      titleBarHeight, windowFrameWidth;
   WStateData  *winStateDataPtr;
   GDHandle    deviceHdl, zoomDeviceHdl;
   SInt32      intersectArea, greatestArea;

   GetPort(&oldPort);
   SetPort(windowPtr);

   EraseRect(&windowPtr->portRect);

   if(zoomInOrOut == inZoomOut)
   {
      windRect = windowPtr->portRect;
      LocalToGlobal(&topLeft(windRect));
      LocalToGlobal(&botRight(windRect));
      titleBarHeight = windRect.top -
                             (*((WindowPeek) windowPtr)->strucRgn)->rgnBBox.top - 1;

      windRect.top = windRect.top - titleBarHeight;

      deviceHdl = LMGetDeviceList();
      greatestArea = 0;

      while(deviceHdl != NULL)
      {
```

```
      if(TestDeviceAttribute(deviceHdl,screenDevice) &&
         TestDeviceAttribute(deviceHdl,screenActive))
      {
        SectRect(&windRect,&(*deviceHdl)->gdRect,&intersectRect);

        intersectArea = (long) (intersectRect.right - intersectRect.left) *
                                         (intersectRect.bottom - intersectRect.top);
        if(intersectArea > greatestArea)
        {
          greatestArea = intersectArea;
          zoomDeviceHdl = deviceHdl;
        }

        deviceHdl = GetNextDevice(deviceHdl);
      }
    }

    if(zoomDeviceHdl == LMGetMainDevice())
      titleBarHeight = titleBarHeight + LMGetMBarHeight();

    windowFrameWidth = (*(((WindowPeek) windowPtr)->strucRgn))->rgnBBox.right -
                                    (*(((WindowPeek) windowPtr)->contRgn))->rgnBBox.right;

    SetRect(&zoomRect,(*zoomDeviceHdl)->gdRect.left + 3 + windowFrameWidth,
                                   (*zoomDeviceHdl)->gdRect.top + titleBarHeight + 3,
                                   (*zoomDeviceHdl)->gdRect.right - 3 - windowFrameWidth,
                                   (*zoomDeviceHdl)->gdRect.bottom - 3 - windowFrameWidth);

    winStateDataPtr = (WStateData *) *((WindowPeek) windowPtr)->dataHandle;
    winStateDataPtr->stdState = zoomRect;
  }

  ZoomWindow(windowPtr,zoomInOrOut,windowPtr == FrontWindow());
  doRedoWindowContent(windowPtr);
  SetPort(oldPort);
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doRedoWindowContent

void doRedoWindowContent(WindowPtr windowPtr)
{
  // Do scroll bar and TextEdit, etc, adjustments here as appropriate.

  InvalRect(&windowPtr->portRect);
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊
// VerticalBlank.c
// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊

#include "Miscellany.h"

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

VBLUPP            gSlotVBLTaskUPP;
VBLStructure      gSlotVBLStructure;
SInt16            gMainSlotNumber;

extern RGBColor   gWhiteColour;

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doSlotVBLTask

void  doSlotVBLTask(void)
{
  GDHandle     mainDeviceHdl;
  SInt16       mainDeviceRefNum;
  DCtlHandle   deviceCtlEntryHdl;
  OSErr        osErr;
  PicHandle    pictureHdl;
  Rect         theRect;
  SInt16       h = 1, v = 1, hh = 0, vv = 0, index = 0;

  gSlotVBLTaskUPP = NewVBLProc((ProcPtr) theSlotVBLTask);

  mainDeviceHdl       = LMGetMainDevice();
  mainDeviceRefNum  = (*mainDeviceHdl)->gdRefNum;
  deviceCtlEntryHdl = GetDCtlEntry(mainDeviceRefNum);
```

```
    gMainSlotNumber     = (SInt16) (*((AuxDCEHandle) deviceCtlEntryHdl))->dCtlSlot;

    RGBForeColor(&gWhiteColour);

    osErr = doInstallSlotVBLTask();
    if(osErr != noErr)
    {
        MoveTo(10,292);
        DrawString("\pCould not install slot-based VBL task.");
        DisposeRoutineDescriptor(gSlotVBLTaskUPP);
        return;
    }

    MoveTo(10,292);
    DrawString("\pClick mouse to remove slot-based VBL task task");
    pictureHdl = GetPicture(rPicture);
    theRect = (*pictureHdl)->picFrame;

    while(!Button())
    {
        while(!gSlotVBLStructure.inVBlankPeriod)
            ;

        OffsetRect(&theRect,h,v);
        DrawPicture(pictureHdl,&theRect);

        gSlotVBLStructure.inVBlankPeriod = false;

        hh = hh + h;
        if(hh == 0 || hh == 350)
            h = -h;
        vv = vv + v;
        if(vv == - 18 || vv == 165)
            v = -v;
    }

    doStopSlotVBLTask();
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doInstallSlotVBLTask

OSErr doInstallSlotVBLTask(void)
{
    OSErr   osErr;

    gSlotVBLStructure.vblTaskStruc.qType      = vType;
    gSlotVBLStructure.vblTaskStruc.vblAddr    = gSlotVBLTaskUPP;
    gSlotVBLStructure.vblTaskStruc.vblCount   = 1;
    gSlotVBLStructure.vblTaskStruc.vblPhase   = 0;
    gSlotVBLStructure.inVBlankPeriod = false;

    osErr = SlotVInstall((QElemPtr) &gSlotVBLStructure.vblTaskStruc,gMainSlotNumber);

    return osErr;
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ theSlotVBLTask

#if TARGET_CPU_68K
void  theSlotVBLTask(void)
#else
void  theSlotVBLTask(VBLStructurePtr vblStructurePtr)
#endif
{
#if TARGET_CPU_68K
    VBLStructurePtr   vblStructurePtr;

    vblStructurePtr = (VBLStructurePtr) GetVBLRec();
#endif

    vblStructurePtr->inVBlankPeriod = true;
    vblStructurePtr->vblTaskStruc.vblCount = 1;
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doStopSlotVBLTask

void  doStopSlotVBLTask(void)
{
    SlotVRemove((QElemPtr) &gSlotVBLStructure.vblTaskStruc,gMainSlotNumber);
    DisposeRoutineDescriptor(gSlotVBLTaskUPP);
}

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

Demonstration Program Comments

When this program is run, the user should make choices from the Demonstration menu, taking the following actions and making the following observations:

- Choose the Notification item and, observing the instructions in the window, click the desktop immediately to make the Finder the foreground application. A notification will be posted by Miscellany about 10 seconds after the Notification item choice is made. Note that, when about 10 seconds have elapsed, the Notification Manager invokes an alert box (Mac OS 8.6 and earlier) or floating window (Mac OS 9.x) and alternates the Finder and Miscellany icons in the Application menu title. Observing the instructions in the alert box/floating window, dismiss the alert (Mac OS 8.6 and earlier only) and then choose the Miscellany item in the Application menu, noting the ◆ mark to the left of the item name. When Miscellany comes to the foreground, note that the icon alternation concludes and that an alert (invoked by Miscellany) appears. Dismiss this second alert box.

- Choose the Notification item again and, this time, leave Miscellany in the foreground. Note that only the alert box invoked by Miscellany appears on this occasion.

- Choose the Notification item again and, this time, click on the desktop and then in the Miscellany window before 10 seconds elapse. Note again that only the alert box invoked by Miscellany appears.

- Choose the Determinate Progress Indicator item, noting that the progress indicator dialog box is automatically disposed of when the (simulated) time-consuming task concludes.

- Choose the Determinate Progress Indicator item again, and this time press the Command-period key combination before the (simulated) time-consuming task concludes. Note that the progress indicator dialog box is disposed of when the Command-period key combination is pressed.

- Choose the Colour Picker item and make colour choices using the various available modes. Note that, when the Colour Picker is dismissed by clicking the OK button, the requested RGB colour values for the chosen colour are displayed in hexadecimal, together with a rectangle in that colour, in the Miscellany window.

- Choose the Multiple Monitors Draw item, noting that the drawing of the simple demonstration image is optimised as follows:

  - On a monitor set to display 256 or more colours, the image is drawn in three distinct colours. The luminance of the three colours is identical, meaning that, if these colours are drawn on a grayscale screen, they will all appear in the same shade of gray.

  - On a monitor set to display 256 shades of gray, the image is drawn in three distinct shades of gray.

- Choose the Slot-Based VBL Task item, bearing in mind that the successive re-drawing of the picture is delayed until the monitor is in the vertical blank period.

In addition, if the user's system has more than one monitor, the user should zoom the window in and out when the window is on the main monitor, when it has been dragged to the second monitor, and when it has been dragged to a position where it is partially displayed on both monitors, noting the standard state, and the monitor zoomed to, in each case.

Miscellany.h

#typedef

The VBLStructure data type, which can be regarded as an expanded VBL task structure, will be used by the slot-based VBL task. The first field is a VBL task structure. The second field will be used to save the A5 register (680x0 code only).

Miscellany.c

Global Variables

doDeviceLoopDrawUPP will be assigned a universal procedure pointer to the application-defined image-optimising drawing function called by DeviceLoop. gProcessSerNum will be assigned the process serial number of the Miscellany application. gMultiMonDragBounds will be passed in the bounds parameter of the DragWindow function. gMultiMonGrowBounds will be passed in the bBox parameter of the GrowWindow function.

main

The call to NewDeviceLoopDrawingProc creates a routine descriptor for the image-optimising drawing function doDeviceLoopDraw.

GetCurrentProcess gets the process serial number of this process.

The next block defines two rectangles. gMultiMonDragBounds is set to equate to the current desktop region, which potentially crosses multiple devices and consists of everything below the menu bar. This establishes the limits within which the user will be able to drag the window. gMultiMonGrowBounds establishes the minimum and maximum heights and widths to apply to window resizing.

Within the event loop, note that:

- The call to doIdle is relevant to the notification demonstration only.
- The application-defined function unloadSegments is called at the bottom of the event loop after the event received by WaitNextEvent has been handled to completion.

doEvents

Note that, within the mouseDown case, gMultiMonDragBounds is be passed in the bounds parameter of DragWindow, gMultiMonGrowBounds is passed in the bBox parameter of GrowWindow, and the application-defined window-zooming function doZoomWindowMultiMonitors is called at the inZoomIn/inZoomOut case.

Within the updateEvt case, if the Multiple Monitors Draw item in the Demonstration menu has been chosen (gMultiMonitorsDrawDemo is true), a call is made to DeviceLoop and the universal procedure pointer to the application-defined drawing function doDeviceLoopDraw is passed as the second parameter.

doMenuChoice

When the Multiple Monitors Draw item in the Demonstration menu is chosen, the window's port rectangle is invalidated so as to force an update event and consequential call to DeviceLoop.

unloadSegments

unloadSegments unlocks, and marks as purgeable, the specified 68K code segments, that is, the segment in which a stub ("do nothing" function) resides. Note that a stub has not been included in the segment containing the VBL task demonstration code, and that that segment is preloaded and locked. This is because a VBL task's code must be locked into physical memory if virtual memory is in operation.

Recall that there is no need to include conditional compilation directives in source code containing segmentation directives before that code is compiled for the PowerPC. Compilers which produce PowerPC code ignore segmentation directives, and any calls to the Segment Manager's UnloadSeg function are simply ignored.

Notification.c

notificationSegment

notificationSegment is the stub, or "do nothing" function, called by unloadSegments at the bottom of the main event loop.

doSetUpNotification

doSetUpNotification is called when the user chooses Notification from the Demonstration menu.

The first line calls an application-defined function which fills in the relevant fields of a notification structure. The next line assigns true to a global variable which records that the Notification item has been chosen by the user.

The next line saves the system tick count at the time that the user chose the Notification item. This value is used later to determine when 10 seconds have elapsed following the execution of this line.

doPrepareNotificationStructure

doPrepareNotificationStructure fills in the relevant fields of the notification structure.

First, however, GetIconSuite creates an icon family based on the specified resource ID and the third parameter, which limits the family to 'ics#', 'ics4' and 'ics8' icons. The GetIconSuite call returns the handle to the suite in its first parameter. The call to GetResource loads the specified 'snd ' resource. GetString loads the specified 'STR ' resource.

The first line of the main block specifies the type of operating system queue. The next line specifies that the ◆ mark is to appear next to the application's name in the Application menu. The next three lines assign the icon suite, sound and string handles previously obtained. The next line specifies that no response function is required to be executed when the notification is posted.

doIdle

doIdle is called from the main event loop when a null event is received. Recall that the canBackground flag is set in the application's 'SIZE' resource, meaning that the application will receive null events when it is in the background.

If the user has not just chosen the Notification item in the Demonstration menu (gNotificationDemoInvoked is false), doIdle simply returns immediately.

If, however, that item has just been chosen, and if 10 seconds (600 ticks) have elapsed since that choice was made, the following occurs:

- The calls to GetFrontProcess and SameProcess determine whether the current foreground process is Miscellany. If it is not, the notification request is installed in the notification queue by NMInstall and a global variable is set to indicate that a request has been placed in the queue by Miscellany. Also, gNotificationDemoInvoked is set to false so as to ensure that the main if block only executes once after the Notification item is chosen.

- If, however, the current foreground process is Miscellany, the application-defined function doDisplayMessageToUser is called to present the required message to the user, via an alert box, in the normal way. Once again gNotificationDemoInvoked is reset to false so as to ensure that the main if block only executes once after the Notification item is chosen.

doOSEvent

doOSEvent handles operating system events.

If the event is a resume event (that is, Miscellany is coming to the foreground) and if the notification request is still in the notification queue (gNotificationInQueue is true), the application-defined function doDisplayMessageToUser is called to remove the notification request from the queue and have Miscellany convey the required message to the user via an alert box.

doDisplayMessageToUser

doDisplayMessageToUser is called by doOSEvent and doIdle in the circumstances previously described.

If a Miscellany notification request is in the queue, NMRemove removes it from the queue and gNotificationInQueue is set to false to reflect this condition. (Recall that, if the nmResp field of the notification structure is not assigned -1, the application itself must remove the queue element from the queue.)

Regardless of whether there was a notification in the queue or not, Miscellany then presents its alert. When the alert is dismissed, the notification's icon suite, sound and string resources are released/disposed of.

ProgressIndicator.c

doProgressIndicator

doProgressIndicator is called when the user chooses Determinate Progress Indicator from the Demonstration menu.

GetNewDialog creates a modal dialog box. The call to UpdateControls draws the dialog box's controls. The call to GetDialogItemAsControl retrieves the handle to the dialog's progress indicator control. SetControlMaximum sets the control's maximum value to equate to the number of steps in a simulated time-consuming task.

The main for loop performs the simulated time-consuming task, represented to the user by the drawing of a large number of coloured rectangles in the window. The task involves 3456 calls to FrameRect.

Within the inner for loop, the application-defined function doCheckForCancel is called to check whether the user has pressed the Command-period key combination. If so, a 'snd ' resource is loaded, played, and released, the dialog is disposed of, an advisory message in drawn in the window, and the function returns.

Within the inner for loop, the rectangles are drawn. Each time round this inner loop, a progress indicator control's value is incremented.

When the outer loop exits (that is, when the Command-period key combination is not pressed before the simulated time-consuming task completes), the dialog is disposed of.

doCheckForCancel

doCheckForCancel scans the event queue for Command-period keyboard events.

The first line sets a variable so as to begin by assuming that such an event is not in the queue.

LMGetEventQueue gets a pointer to the event queue header. The next line gets a pointer to first queue element. The while loop scans the whole of the event queue, exiting only when a Command-period key event is found in the queue or the entire queue has been scanned.

Inside the loop, if a key-down event is found, the first two lines in the if block get the character code and the third line checks whether the Command key was down. If the Command key was down, and if the character code is the code for the period character (charCode 0x2e), the variable foundCommandPeriod is set to true, causing the loop to exit and doCheckForCancel to return true.

If a Command-period keyboard event was not found, the last two lines in the while loop call up the next queue for examination.

The last line returns the result of the search.

ColourPicker.c

doColourPicker

doColourPicker is called when the user chooses Colour Picker from the Demonstration menu.

The first block erases the window's content area and paints a rectangle in the colour which will be passed in GetColor's inColor parameter.

The next line assigns 0 to the fields of the Point variable to be passed in GetColor's where parameter. ((0,0) will cause the Colour Picker dialog box to be centred on the main screen.)

The call to GetColor displays the Colour Picker's dialog box. GetColor retains control until the user clicks either the OK button or the Cancel button, at which time the port rectangle is invalidated, causing the function doDrawColourPickerChoice to be called.

doDrawColourPickerChoice

If the user clicked the OK button, a filled rectangle is painted in the window in the colour returned in GetColor's outColor parameter, and the values representing the red, green, and blue components of this colour are displayed at the top of the window in hexadecimal. Note that the application-defined function doDecimalToHexadecimal is called to convert the decimal (UInt32) values in the fields of the RGBColor variable outColor to hexadecimal.

If the user clicks the Cancel button, a filled rectangle is painted in the window in the colour passed in GetColor's inColor parameter.

doDecimalToHexadecimal

doDecimalToHexadecimal converts a UInt16 value to a hexadecimal string.

MultiMonitor.c

doDeviceLoopDraw

doDeviceLoopDraw is the image-optimising drawing function the universal procedure pointer to which is passed in the second parameter in the DeviceLoop call in the function doEvents. (Recall that the DeviceLoop call is made whenever the Multiple Monitors Draw item in the Demonstration menu has been selected and an update event is received.) DeviceLoop scans all active video devices, calling doDeviceLoopDraw whenever it encounters a device which intersects the drawing region, and passing certain information to doDeviceLoopDraw.

The second line casts the SInt32 value received in the userData parameter to a WindowPtr. The window's content area is then erased.

If an examination of the device's attributes, as received in the deviceFlags formal parameter, reveals that the device is a colour device, three rectangles are painted in the window in three different colours. (The luminance value of these colours is the same, meaning that the rectangles would all be the same shade of gray if they were drawn on a monochrome (grayscale) device.)

If the examination of the device's attributes reveals that the device is a monochrome device, the rectangles are painted in three distinct shades of gray.

doZoomWindowMultiMonitors

doZoomWindowMultiMonitors is called when the user clicks in the window's zoom box.

The first two lines save and set the current graphics port. The third line erases the window's port rectangle prior to the zoom so as to avoid flicker.

The main if block executes only if the direction of the zoom is "out" to the standard state. The purpose of this block of code is to determine the standard state rectangle and, in a multi-monitors environment, the monitor on which the zoomed window is to be displayed.

The first block inside the if block converts the window's port rectangle to global coordinates and gets the height of the window's title bar. The next line establishes a rectangle equal to the window's port rectangle, plus the window's title bar, in global coordinates.

LMGetDeviceList gets a handle to the first gDevice structure in the device list and the variable greatestArea is assigned 0.

The while loop then walks the device list. For each active video device, the associated gDevice structure's gdRect field is compared to the window's rectangle by a call to SectRect. If the two rectangles intersect, the coordinates of the intersection are assigned to the intersectRect variable, otherwise an empty rectangle ((0,0)(0,0)) is assigned to intersectRect. The area of the intersection rectangle is calculated and stored in the variable intersectArea. If the new value in intersectArea is greater than that calculated during any previous pass through the loop, the variable zoomDeviceHdl is assigned the GDHandle of the device currently being examined.

GetNextDevice gets the handle to the next device in the device list. The while loop exits when this call returns NULL. When the loop exits, the contents of the variable zoomDeviceHdl represents the video device on which the window should be zoomed to the standard state, that is, the device on which the largest area of the window currently appears.

If the call to LMGetMainDevice reveals that this device is the main device, the height of the menu bar is added to the value in the variable which holds the window's title bar height.

The next two lines determine the width of the window frame and the following four lines establish the standard state rectangle. This latter is three pixels inside the rectangle contained in the gdRect field of the device's gDevice structure, but with the top adjusted to account for the height of the title bar (and the menu bar if the device is the main device) and the sides and bottom adjusted for the width of the window frame. The last two lines in the main if block then assign this rectangle to the stdState field of the window's state data structure.

Below the main if block, ZoomWindow is called to zoom the window in the appropriate direction, following which an application-defined function is called to redraw the window contents as appropriate. Finally, the saved graphics port is restored.

doRedoWindowContent

doRedoWindowContent is called by doZoomWindowMultiMonitors to redraw the content region of a newly-zoomed window. In this demonstration the window's content area is simply invalidated, forcing an update event.

VerticalBlank.c

doSlotVBLTask

doSlotVBLTask is called when the user chooses Slot-Based VBL Task from the Demonstration menu. In this demonstration, the slot-based VBL task is used to delay the drawing of a picture at a new location until the monitor enters the vertical blank period.

The first line creates a routine descriptor for the slot-based VBL task.

The next block gets the slot number of the main graphics device. This process involves getting a handle to the startup gDevice structure, extracting from that structure the device driver's reference number, getting a handle to the DCtlEntry structure, and then getting the slot number.

doInstallSlotVBLTask is then called to install the slot-based VBL task. If this call is not successful, the function returns.

The call to GetPicture loads the specified 'PICT' resource, following which the picFrame field in the resource is copied to a local variable. Within the outer while loop, this rectangle is continually offset between successive calls to DrawPicture, causing the picture to appear to move around the window. The inner (empty) while loop continues to cycle until the inBlankPeriod field of the expanded VBL task structure is set to true by the VBL task, at which time the Picture is drawn and the inBlankPeriod field is set back to false.

When the user clicks the mouse, the outer while loop exits and doStopSlotVBLTask is called to remove the slot-based VBL task.

doInstallSlotVBLTask

doInstallSlotVBLTask installs the slot-based VBL task.

The first four lines fill in the appropriate fields of the VBL task record. Note that the vblCount field is set to 1 so that the VBL task will execute at the first interrupt.

The fifth line sets a field in the expanded VBL task structure to false. This field will be set to true by the slot-based VBL task. Note that there is no need to save the application's A5 in this case because the slot-based VBL task does not need to access an application global variable.

SlotVInstall attempts to install the task in the slot-based queue. The success, or otherwise, of the attempted installation is returned to the calling function.

theSlotVBLTask

theSlotVBLTask is the slot-based VBL task itself. It is similar to theSystemVBLTask except that no measures are required to facilitate access to the application's global variables.

When the task executes, it sets to true the flag in the expanded VBL task structure which indicates that the vertical blanking period has just been entered. When the task executes, the value in the vblCount field of the VBL task record will be 0. So that the task will execute again at the next interrupt, the last line sets the vblCount field of the VBL task record back to 1.

doStopSlotVBLTask

doStopSlotVBLTask is called when the user clicks the mouse button. It removes the slot-based VBL task from the relevant queue and disposes of the routine descriptor.

Recall that, at the Demonstration Program Controls2 Comments section at Chapter 7 — Introduction to Controls, mention was made of horizontal "tearing" of the picture in the lower section of the window when it was being scrolled using the scroll arrows, and that this would be addressed in the Demonstration Program Comments section of this chapter.

This "tearing" can be eliminated by delaying the drawing of the picture until the vertical blanking period is entered, using the same approach as is used in the slot-based VBL task demonstration, above. Simply install a slot-based VBL task using the relevant source code in the Miscellany demonstration and change the bottom of the actionFuncLive function in Controls2.c to the following:

```
    doMoveScrollBox(controlHdl,scrollDistance);
  }

  SetOrigin(GetControlValue((*docStrucHdl)->scrollbarLiveHdl),0);

  if(!gVBLInstallFail)    // to true if VBL task is not successfully installed.
  {
    while(!gSlotVBLStructure.inVBlankPeriod)           // Wait for vertical blanking period.
      ;
    DrawPicture(gPictHandleLive,&gPictRectLive);
    gSlotVBLStructure.inVBlankPeriod = false;
  }
  else
    DrawPicture(gPictHandleLive,&gPictRectLive);

  SetOrigin(0,0);
}
```