22 Version 1.1

MISCELLANY

Includes Demonstration Program Miscellany

Code Segmentation and Heap Space Optimisation

As stated in the CodeWarrior manual Targetting Mac OS, 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 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 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 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 routine 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 routine.

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" routine, 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 routine called, say, `doUnloadSegments` which calls the Segment Loader routine `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 routine in its input parameter, unlocks that segment, and makes it purgeable. Note also that you could pass any of the segment's routines as the parameter to the `UnloadSeg` call; however, it is preferable to use stubs dedicated to this purpose because the other routines in the segment could well be moved to another segment during future updating of the code.

- Place the `doUnloadSegments` routine 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 routine 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);

    unloadSegments();
  }
}
```

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 routine contained in one of the purged segments, the Segment Loader once again loads that segment into your application's heap as a relocatable block.

Status Bars and Scanning for a Command-Period Event

Status 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 status bar, which should indicate visually to the user the current state of progress in that operation.

Ordinarily, status bars should be displayed within a modal dialog box. Static text within the dialog box should advise the user how to terminate the operation (ordinarily by using the Command-period combination) before it completes of its own accord.

Scanning for a Command-Period Event

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. One way to satisfy this requirement is to periodically call an application-defined function which scans the event queue for a Command-period keyboard event. This function should return `true` if a Command-period keyboard event is found.

The application-defined function should first get a pointer to the first queue element. It should then scan the queue for a key-down event. 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. The calling function should, in turn, terminate the lengthy operation.

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.

Examples of Notifications - PrintMonitor and Alarm Clock

You may have noticed that, if you are attempting to print in the background and the printer is not turned on, the printer cable is disconnected, or the printer is out of paper:

- An alert box is presented advising you that there is a printing problem.
- The Finder or PrintMonitor icon begins alternating with the current application's icon on the right of the menu bar.
- A ◆ mark appears on the left of the Finder or PrintMonitor item in the Application menu[1].

You may also have noticed that, when the Alarm Clock alarm goes off, the system alert sound plays and the Alarm Clock icon begins alternating with the Apple menu icon at the left of the menu bar.

These are two instances of the Notification Manager at work. The Notification Manager allows applications running in the background (in these examples, the Finder, PrintMonitor, and Alarm Clock applications) to communicate with the user.

The Notification Manager provides a one-way communications path from the application to the user. There is no provision for carrying information back from the user to the application.

Elements of a Notification

In addition to the alert box, the icon rotation, the ◆ mark, and the playing of the system alert sound, the Notification Manager also provides for the playing of a sound from a specified `'snd '` resource and for the specification of a **response procedure**, which is a procedure executed as the final step in a notification. In short, a notification comprises one or more of five possible elements.

The elements of a notification, assuming they have been specified, occur in the following sequence:

- The ◆ mark appears. (Note that the ◆ mark only appears while the application posting the notification remains in the background. The ◆ mark is replaced by the familiar ✓ mark when that application is brought to the foreground.)
- The icon alternation begins. (Typically, the icon which alternates with the foreground application's icon is the posting application's small icon. Note that several applications might post notifications, so there might be a series of alternating icons. Note also that the location of each icon 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 icon flashes above the Application menu; otherwise the icon flashes above the Apple menu.)
- The Sound Manager plays the 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.)

---

[1]The ◆ mark is intended to prompt the user to switch the marked application to the foreground.

- The alert box appears, and the user dismisses it. (The application posting the notification specifies the text for the alert box.)

- The response procedure executes. (The response procedure 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.[2]

- **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.

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

Creating a Notification Request

The Notification Record

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

[2]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.

```
  Handle     nmSound;     // Handle to sound record.
  StringPtr  nmStr;       // Pointer to string to appear in alert.
  NMUPP      nmResp;      // Pointer to response routine.
  long       nmRefCon;    // Available for application use.
};

typedef struct NMRec 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 a small icon, or 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. If `nmIcon` is not `NULL`, the Notification Manager determines whether it is a handle to a small icon or to an icon family containing a small colour icon. 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` Pointer to a response procedure. If `nmResp` is set to `NULL`, no response procedure executes when the notification is posted. If `nmResp` is set to `-1`, then a pre-defined procedure 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 address of your own response procedure, the Notification Manager passes it one parameter, a pointer to your notification record. For example, this is how you would declare a response procedure having the name `theResponse`:

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

You can use response procedures to remove notification requests from the notification queue, free any memory[3], or set a global variable in your application to record that the notification was posted[4]. 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 procedure executes only after the user has clicked the OK button in the alert box.

---

[3]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.

[4]When the Notification Manager calls your response procedure, 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.

If you choose audible or alert notifications, you should probably set `nmResp` to `-1` so that the notification record 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(&notificationRecord);
```

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(&notificationRecord);
```

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

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.
- Routines 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 HSL or HSV (hue, saturation, lightness, 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[5]) 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.)

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

HSL and HSV Models

The HSL 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 HSL model and by the value (V) value in the HSV model.

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

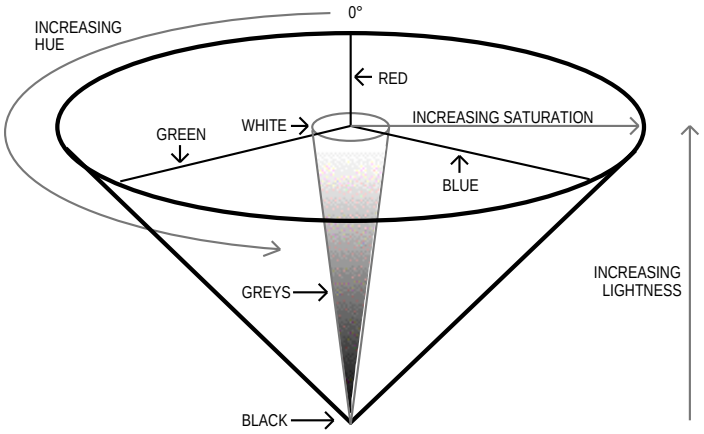

FIG 1 - HSL/HSV COLOUR CONE

The Color Picker

The Color Picker allows the user to specify a colour using either the HSL or RGB models. A somewhat refined version of Color Picker was introduced with System 7.5, and it is this version that is described below. (The previous version is broadly similar in that it allows the user to specify a colour using either the HSL or RGB models.)

Using the Color Picker HSL Mode

When first opened, the Color Picker defaults to the HSL display as shown at Fig 2. 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 2 to Fig 1, the coloured disc at Fig 2 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).

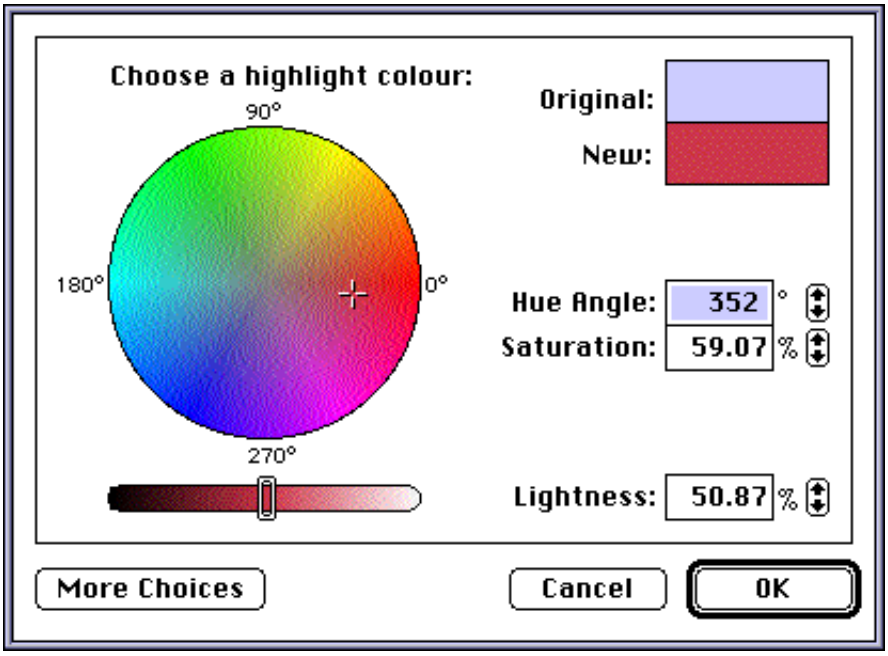

FIG 2 - COLOR PICKER DIALOG IN HSL MODE

Using the Color Picker RGB Mode

By clicking on the **More Choices** button, a list opens up showing the colour models available. Clicking on the Apple RGB item in the list results in the RGB display shown at Fig 3. The desired red, green and blue values may be set using the three slider controls or may be entered directly in the fields on the right of the sliders.

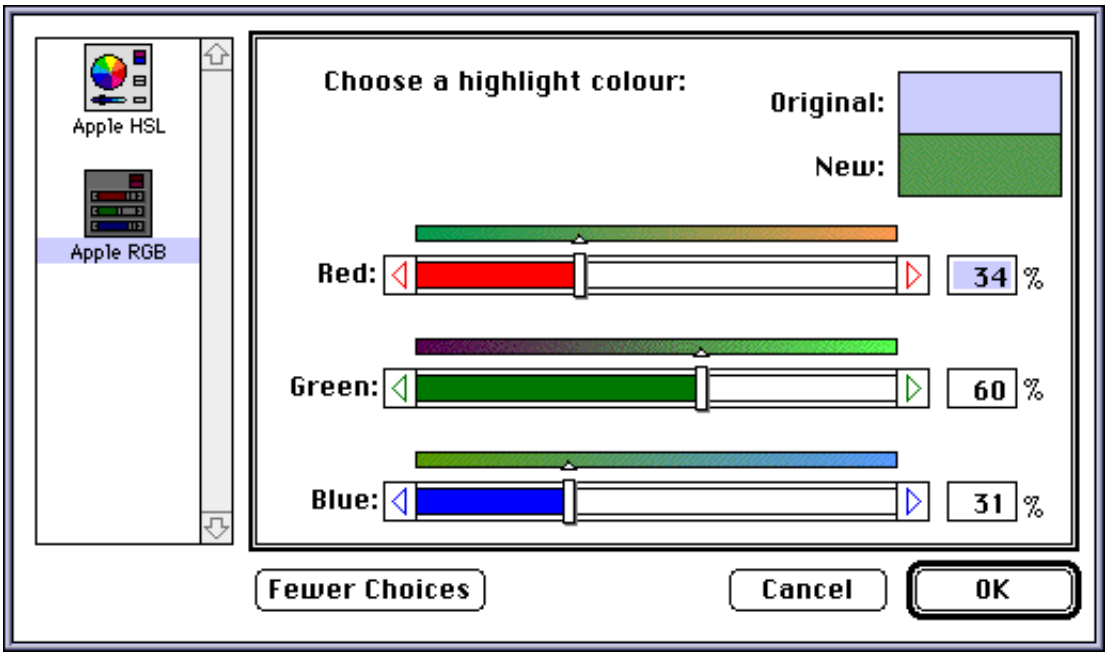

FIG 3 - COLOR PICKER DIALOG IN RGB MODE

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 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.

Ensuring Compatibility with the Operating Environment

If your application is to run successfully in all of the software and hardware environments that may be present in the full range of Macintosh models, it must be able to acquire information about a large number of machine-dependent features and, where appropriate, act on that information. For example, the demonstration program which accompanies this chapter uses different blocks of code to draw a status bar depending on whether or not Color QuickDraw is present.

Getting Operating Environment Information - The `Gestalt` Function

The `Gestalt` function may be used to acquire a wide range of information about the operating environment[6]:

```
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.

[6]Although the `Gestalt` function can provide your application with most of the basic information it needs about hardware and software features, you may still need to call other routines to determine more specific features. For example, if you need to determine the resolution of the main Macintosh screen, you will need to use the `ScreenRes` routine.

- The type of memory management unit (MMU).
- The size of the available RAM.
- The amount of available virtual memory.
- The version of QuickDraw currently present.
- 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 for the existence of Color QuickDraw, the value returned in the `response` parameter may be compared with `gestalt8BitQD` as follows:

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

osErr = Gestalt(gestaltQuickdrawVersion,&response);
if(osErr == noErr)
{
  if(response < gestalt8BitQD)
     colorQuickDrawPresent = 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
```

Determining Whether a Trap is Available

If you call a system routine (that is, a trap) on a machine which does not implement it, your application will crash. Before your application calls a trap that may not be available on all machines, therefore, it needs to determine that trap's availability in the current operating environment.

One way to do this, of course, is to use the `Gestalt` function. If you happen to know that the trap has been included in the system software from a particular version number onwards, you could have your application call the `Gestalt` function to ascertain what version of the relevant driver or manager is present.

There are several cases, however, where you cannot use `Gestalt` for this purpose. For example, the trap for whose existence you wish to test might not be included in any manager, or there may not be a `Gestalt` selector code for the particular manager concerned. In this situation you must test directly for the existence of the trap. Unfortunately, this is not as simple a procedure as you might suppose; however, the demonstration program shows how it can be done.

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 control panel. The user can set the maximum number of colours (or grays) to be displayed to black-and-white (pixel depth = 1), four colours/grays (pixel depth = 2), sixteen colours/grays (pixel depth = 4), and so on up to that pixel depth which equates to the computer's maximum video capability.[7] These settings are made separately for each individual screen. 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 9 — 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` records, the system creates and initialises a `GDevice` record for each video device found during start-up[8], all records are linked together in a list called the **device list**, and the global variable `DeviceList` holds a handle to the first record in the list.

- At any given time, one, and only one, graphics device is the **current device**[9], that is, the one in which the drawing is taking place. A handle to the current device's `GDevice` record is placed in the global variable `TheGDevice`.

By default, the `GDevice` record 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` record for the video device on which it is actively drawing.

Also recall from Chapter 9 — QuickDraw Preliminaries that two of the fields in a `GDevice` record 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 two issues:

- **Image Optimisation.** To draw a particular graphic, your application may have to call different drawing routines for that graphic depending on the pixel depth of the video device intersecting your window's drawing region, the aim being to optimise the appearance of the image regardless of whether it is being displayed on a device set to a pixel depth of, say, 1 or a

---

[7]Some Macintosh computers do not permit the setting of pixel depths of less than 8.

[8]The Monitors 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.

[9]The current device is sometimes referred to as the **active device**.

device set to a pixel depth of, say, 8. For example, in the case of a device set to a pixel depth of 1 (black-and-white), you might elect to draw a specific part of the image using the pattern `dkGray` whereas in the case of a device set to a pixel depth of 4, 8, or 16 you might elect to draw the same part in dark blue.

- **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.

Image Optimisation

The `DeviceLoop` routine 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 pixel depth and other attributes. Armed with the pixel depth information, your application can then invoke whichever of its drawing routines is optimised for that particular colour resolution.

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

```
pascal void MyDrawingProc(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 `MyDrawingProc`, 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 procedure.
`dontMatchSeeds`	Do not consider `ctSeed` fields of `ColorTable` records for graphics devices when comparing them.
`allDevices`	Ignore value of `drawingRgn` parameter and instead call drawing procedure 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**[10], 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` record.
- 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.

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

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.

An Image Optimisation Short Cut — Default Button Bold Outline

Recall that the demonstration program at Chapter 6 — Dialogs and Alerts contains an action procedure which draws the bold outline around the default button in a dialog box, that a pointer to that routine is installed in a user item in the dialog box, and that, as a result, the action procedure is called whenever the user item is part of the dialog box's update region during a dialog box update.

When the default button is inactive, and if the draw is to a basic graphics port, the action procedure draws the bold outline in the `gray` pattern; however, if the draw is to a colour graphics port, `GetGray` is called to get an intermediate RGB colour between the current foreground and background colours. Assuming the `GetGray` call is successful, the colour returned is the best intermediate colour available on the device specified in the first parameter of the `GetGray` call, and the bold outline is drawn in that intermediate colour. The relevant lines of code (Lines 1137-1149 at Chapter 6) are as follows:

```
if(isColour)                                    // If drawing to a colour graphics port.
{
  ...
  targetDevice = LMGetMainDevice();
  newGray = GetGray(targetDevice,&backColour,&newForeColour);
}

if(newGray)                      // If the GetGray call gets an intermediate colour ...
  RGBForeColor(&newForeColour);             // ... the draw will be in this colour ...
else                                                          // ... otherwise ...
  PenPat(gray);                                 // ... the draw will be in this pattern.
```

Note that the device specified in the `GetGray` call is that associated with the main screen (the screen with the menu bar). This is a satisfactory approach in a single monitor environment; however, it is not satisfactory in a multi-monitor environment. If for, example, the main screen's pixel depth is 1, the second screen's pixel depth is 8, and the movable modal or modeless dialog box has been dragged to the second screen, the bold outline will be drawn using the `gray` pattern rather than an intermediate colour. (`GetGray` will return `false` when the specified device's pixel depth is 1.)

The solution for a multi-monitors environment is to specify to `GetGray` the device on which the **OK** button, or the greater part of that button, is currently being displayed. Accordingly, the line before the `GetGray` call should be replaced by:

```
targetDevice = doGetRectsDevice((*(ControlHandle) itemHandle)->contrlRect);
```

and the following function should be included:

```
GDHandle  doGetRectsDevice(Rect theRect)
{
  SInt32    greatestArea, intersectArea;
  GDHandle  deviceHdl, deviceHdlToReturn;
  Rect      intersectRect;

  LocalToGlobal(&topLeft(theRect));
  LocalToGlobal(&botRight(theRect));

  deviceHdl = LMGetDeviceList();
  greatestArea = 0;

  while(deviceHdl != NULL)
  {
    if(TestDeviceAttribute(deviceHdl,screenDevice) &&
      TestDeviceAttribute(deviceHdl,screenActive))
    {
      SectRect(&theRect,&(*deviceHdl)->gdRect,&intersectRect);
```

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

This function checks the default button's rectangle against the boundary rectangle of all active video devices in the device list and determines which device contains the greater part of the button's rectangle. The code is essentially identical to that used in the `doZoomWindowMultiMonitor` function in this chapter's demonstration program to check a window's rectangle against the boundary rectangle of all active video devices in order to determine which device contains the greater part of the window's rectangle.

This image-optimisation example has been termed a "short cut" because it does not involve the use of `DeviceLoop`, which means that it will produce the required result only if the default button does not span two screens, one of those screens being set to a pixel depth of 1 and the other to some higher pixel depth. This simplified approach may, nonetheless, be considered acceptable in the case of a small and relatively insignificant image like the default button outline, given the low probability of the user positioning a movable modal or modeless dialog box such that the default button spans two screens and/or setting the pixel depth of one of those screens to 1.

Main Segment Loader Routines

Unlock Code Segments and Make Purgeable

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

Terminate Caller, Release Heap, and Launch Finder

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

Main Event Manager Data Types and Routines

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;
```

Routines

Get Address of Event Queue Header

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

Main Notification Manager Data Types and Routines

Data Types

Notification Record

```
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 record.
  StringPtr   nmStr;        // String to appear in alert.
  NMUPP       nmResp;       // Pointer to response routine.
  long        nmRefCon;     // For application use.
};

typedef struct NMRec NMRec,*NMRecPtr;
```

Routines

Add Notification Request to the Notification Queue

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

Remove Notification Request from the Notification Queue

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

Main Process Manager Data Types and Routines

Data Types

Process Serial Number

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

Routines

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);
```

Main Gestalt Manager Constants and Routines

Constants

Gestalt Error Codes

```
gestaltUnknownErr               = -5550, // Value returned if Gestalt doesn't know the answer.
gestaltUndefSelectorErr         = -5551, // Undefined selector was passed to Gestalt.
```

Environment Selectors

```
gestaltAddressingModeAttr       'addr'   // Addressing mode attributes.
gestalt32BitAddressing          = 0      // Using 32-bit addressing mode.
gestalt32BitSysZone             = 1      // 32-bit compatible system zone.
gestalt32BitCapable             = 2      // Machine is 32-bit capable.

gestaltFPUType                  'fpu '   // FPU type.
gestaltNoFPU                    = 0      // No FPU.
gestalt68881                    = 1      // 68881 FPU.
gestalt68882                    = 2      // 68882 FPU.
gestalt68040FPU                 = 3      // 68040 built-in FPU.

gestaltKeyboardType             'kbd '   // Keyboard type.
gestaltMacKbd                   = 1
gestaltMacAndPad                = 2
gestaltMacPlusKbd               = 3
gestaltExtADBKbd                = 4
gestaltStdADBKbd                = 5
gestaltPrtblADBKbd              = 6
gestaltPrtblISOKbd              = 7
gestaltStdISOADBKbd             = 8
gestaltExtISOADBKbd             = 9
gestaltADBKbdII                 = 10
gestaltADBISOKbdII              = 11
gestaltPwrBookADBKbd            = 12
gestaltPwrBookISOADBKbd         = 13

gestaltProcessorType            'proc'   // Processor type.
gestalt68000                    = 1
gestalt68010                    = 2
gestalt68020                    = 3
gestalt68030                    = 4
gestalt68040                    = 5

gestaltQuickdrawVersion         'qd  '   // QuickDraw version.
gestaltOriginalQD               = 0x000  // Original 1-bit QD.
gestalt8BitQD                   = 0x100  // 8-bit color QD.
gestalt32BitQD                  = 0x200  // 32-bit color QD.
gestalt32BitQD11                = 0x210  // 32-bit color QDv1.1.
gestalt32BitQD12                = 0x220  // 32-bit color QDv1.2.
gestalt32BitQD13                = 0x230  // 32-bit color QDv1.3.

gestaltQuickdrawFeatures        'qdrw'   // QuickDraw features.
gestaltHasColor                 = 0      // Color QuickDraw present.
gestaltHasDeepGWorlds           = 1      // GWorlds can be deeper than 1-bit.
gestaltHasDirectPixMaps         = 2      // PixMaps can be direct (16 or 32 bit).
```

```
gestaltHasGrayishTextOr         = 3      // supports text mode grayishTextOr.

gestaltPhysicalRAMSize          'ram '   // Physical RAM size.

gestaltSoundAttr                'snd '   // Sound attributes.
gestaltStereoCapability         = 0      // Sound hardware has stereo capability.
gestaltStereoMixing             = 1      // Stereo mixing on external speaker.
gestaltSoundIOMgrPresent        = 3      // The Sound I/O Manager is present.
gestaltBuiltInSoundInput        = 4      // Built-in Sound Input hardware is present.
gestaltHasSoundInputDevice      = 5      // Sound Input device available.
```

Information-only Selectors

```
gestaltMachineType              'mach'   // Machine type.
kMachineNameStrID               = -16395
gestaltClassic                  = 1
gestaltMacXL                    = 2
gestaltMac512KE                 = 3
gestaltMacPlus                  = 4
gestaltMacSE                    = 5
gestaltMacII                    = 6
gestaltMacIIx                   = 7
gestaltMacIIcx                  = 8
gestaltMacSE030                 = 9
gestaltPortable                 = 10
gestaltMacIIci                  = 11
gestaltMacIIfx                  = 13
gestaltMacClassic               = 17
gestaltMacIIsi                  = 18
gestaltMacLC                    = 19
gestaltQuadra900                = 20
gestaltPowerBook170             = 21
gestaltQuadra700                = 22
gestaltClassicII                = 23
gestaltPowerBook100             = 24
gestaltPowerBook140             = 25

gestaltSystemVersion            'sysv'   // System version.
```

Routines

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

Relevant QuickDraw Constants and Routines

Constants

Flag Bits for `gdFlags` Field of `GDevice` Record

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

Routines

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);
```

Demonstration Program

```
// ############################################################################################
// Miscellany.h
// ############################################################################################
//
// Miscellany source code is contained in two files, namely, Main.c and Demos.c.  Within
// the CodeWarrior project, Main.c is in Segment 2 and Demos.c is in Segment 3.  (Note
// that this small program does not really require such segmentation; the code is
// segmented only to facilitate demonstration of the Segment Loader aspects.
//
// This program demonstrates:
//
// •  The use of stubs in code segments, together with a function which uses those stubs
//    to unlock code segments and make them purgeable.
//
// •  The use of a status bar to graphically indicate the current status of a time-
//    consuming operation.
//
// •  The use of the Command-period key combination to terminate a time-consuming
//    operation before it concludes.
//
// •  The use of the Notification Manager to allow an application running in the
//    background to communicate with the foreground application.
//
// •  The determination of whether a particular application is currently the foreground
//    application.
//
// •  The use of the Color Picker to solicit a choice of colour from the user.
//
// •  The determination of whether a particular trap is available.
//
// •  Image drawing optimisation and window zooming in a multi-monitors environment.
//
// 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.
//
// •  An 'ALRT' resource (purgeable), and associated 'DITL' resource (purgeable), for
//    displaying a message to the user from within the Notification Manager demonstration.
//
// •  A 'DLOG' resource (purgeable), and associated 'DITL' and 'dctb' resources
//    (purgeable), for a dialog box in which the status bar is displayed.
//
// •  '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 'SIZE' resource with the acceptSuspendResumeEvents, doesActivateOnFGSwitch,
//    canBackgound, and is32BitCompatible flags set.
//
// ############################################################################################

// ……… includes

#include <Fonts.h>
#include <Menus.h>
#include <TextEdit.h>
#include <Dialogs.h>
#include <SegLoad.h>
#include <ToolUtils.h>
#include <Devices.h>
#include <Processes.h>
#include <Gestalt.h>
#include <Resources.h>
#include <Sound.h>
#include <Notification.h>
#include <Icons.h>
```

```
#include <Processes.h>
#include <ColorPicker.h>
#include <Traps.h>
#include <LowMem.h>

// ……… defines

#define mApple          128
#define  iAbout         1
#define mFile           129
#define  iQuit          11
#define mDemonstration  131
#define  iCommandPeriod 1
#define  iNotification  2
#define  iColourPicker  3
#define  iTrapAvailable 4
#define  iMultiMonitors 5
#define rMenubar        128
#define rWindow         128
#define rAlert          128
#define rDialog         129
#define  iUserItem      1
#define rIconFamily     128
#define rBarkSound      8192
#define rString         128
#define topLeft(r)      (((Point *) &(r))[0])
#define botRight(r)     (((Point *) &(r))[1])

// ……… function prototypes

// Main.c

void        main                        (void);
void        doInitManagers              (void);
void        doEvents                    (EventRecord *);
void        doMouseDown                 (EventRecord *);
void        doKeyDown                   (UInt8);
void        doMenuChoice                (SInt32 menuChoice);
void        unloadSegments              (void);

// Demos.c

void        demosSegment                (void);
void        doCommandPeriodAndStatusBar (void);
Boolean     doCheckForCommandPeriod     (void);
void        doDrawStatusBar             (DialogPtr,Rect,SInt16,SInt16);
void        doSetUpNotification         (void);
void        doPrepareNotificationRecord (void);
void        doNullEvent                 (void);
void        doOSEvent                   (EventRecord *);
void        doDisplayMessageToUser      (void);
void        doColourPicker              (void);
char        *doDecimalToHexadecimal       (UInt16 n);
Boolean     doCheckSlotVInstallAvailable (void);
Boolean     trapAvailable               (SInt16 theTrap);
pascal void doDeviceLoopDraw            (SInt16,SInt16,GDHandle,SInt32);
void        doZoomWindowMultiMonitors   (WindowPtr,SInt16);
void        doRedoWindowContent         (WindowPtr);

// ############################################################################################

// ############################################################################################
// Main.c
// ############################################################################################

#include "Miscellany.h"

// ……… global variables

Boolean             gColorQuickDraw;
Boolean             gDone;
WindowPtr           gWindowPtr;
ProcessSerialNumber gProcessSerNum;
Boolean             gMultiMonitorsDrawDemo = false;

// ############################################################################## main
```

```
151
152 void  main(void)
153 {
154   OSErr         osErr;
155   SInt32        response;
156   Handle        menubarHdl;
157   MenuHandle    menuHdl;
158   EventRecord   eventRec;
159
160   // ……… initialise managers
161
162   doInitManagers();
163
164   // ……… check for Color QuickDraw
165
166   gColorQuickDraw = true;
167
168   osErr = Gestalt(gestaltQuickdrawVersion,&response);
169   if(response < gestalt8BitQD)
170     gColorQuickDraw = false;
171
172   // ……… set up menu bar and menus
173
174   menubarHdl = GetNewMBar(rMenubar);
175   if(menubarHdl == NULL)
176     ExitToShell();
177   SetMenuBar(menubarHdl);
178   DrawMenuBar();
179
180   menuHdl = GetMenuHandle(mApple);
181   if(menuHdl == NULL)
182     ExitToShell();
183   else
184     AppendResMenu(menuHdl,'DRVR');
185
186   // ……… open window
187
188   if(gColorQuickDraw)
189     gWindowPtr = GetNewCWindow(rWindow,NULL,(WindowPtr)-1);
190   else
191     gWindowPtr = GetNewWindow(rWindow,NULL,(WindowPtr)-1);
192
193   if(gWindowPtr == NULL)
194       ExitToShell();
195
196   SetPort(gWindowPtr);
197   TextSize(10);
198
199   // ……… get process serial number of this process
200
201   GetCurrentProcess(&gProcessSerNum);
202
203   // ……… enter eventLoop
204
205   gDone = false;
206
207   while(!gDone)
208   {
209     if(WaitNextEvent(everyEvent,&eventRec,30,NULL))
210       doEvents(&eventRec);
211     else
212       doNullEvent();
213
214     unloadSegments();
215   }
216 }
217
218 // ########################################################################## doInitManagers
219
220 void  doInitManagers(void)
221 {
222   MaxApplZone();
223   MoreMasters();
224
225   InitGraf(&qd.thePort);
226   InitFonts();
```

```
  InitWindows();
  InitMenus();
  TEInit();
  InitDialogs(NULL);

  InitCursor();
  FlushEvents(everyEvent,0);
}

// ########################################################################## doEvents

void  doEvents(EventRecord *eventRecPtr)
{
  WindowPtr windowPtr;
  SInt32    userData;

  switch(eventRecPtr->what)
  {
    case mouseDown:
      doMouseDown(eventRecPtr);
      break;

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

      BeginUpdate(windowPtr);
      if(gMultiMonitorsDrawDemo == true)
      {
        userData = (SInt32) windowPtr;
        DeviceLoop(windowPtr->visRgn,(DeviceLoopDrawingUPP) doDeviceLoopDraw,userData,0);
      }
      EndUpdate(windowPtr);
      break;

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

// ########################################################################## doMouseDown

void  doMouseDown(EventRecord *eventRecPtr)
{
  SInt16    partCode;
  WindowPtr windowPtr;

  partCode = FindWindow(eventRecPtr->where,&windowPtr);

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

    case inSysWindow:
      SystemClick(eventRecPtr,windowPtr);
      break;

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

    case inDrag:
      DragWindow(windowPtr,eventRecPtr->where,&qd.screenBits.bounds);
      break;

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

```
303
304 // ############################################################################## doMenuChoice
305
306 void  doMenuChoice(SInt32 menuChoice)
307 {
308   SInt16  menuID, menuItem;
309   Str255  itemName;
310   SInt16  daDriverRefNum;
311
312   menuID = HiWord(menuChoice);
313   menuItem = LoWord(menuChoice);
314
315   if(menuID == 0)
316     return;
317
318   switch(menuID)
319   {
320     case mApple:
321       if(menuItem == iAbout)
322         SysBeep(10);
323       else
324       {
325         GetMenuItemText(GetMenuHandle(mApple),menuItem,itemName);
326         daDriverRefNum = OpenDeskAcc(itemName);
327       }
328       break;
329
330     case mFile:
331       if(menuItem == iQuit)
332         gDone = true;
333       break;
334
335     case mDemonstration:
336       gMultiMonitorsDrawDemo = false;
337       switch(menuItem)
338       {
339         case iCommandPeriod:
340           doCommandPeriodAndStatusBar();
341           break;
342
343         case iNotification:
344           EraseRect(&gWindowPtr->portRect);
345           doSetUpNotification();
346           break;
347
348         case iColourPicker:
349           doColourPicker();
350           break;
351
352         case iTrapAvailable:
353           EraseRect(&gWindowPtr->portRect);
354           MoveTo(150,110);
355           if(doCheckSlotVInstallAvailable())
356             DrawString("\pTrap is available");
357           else
358             DrawString("\pTrap is not available");
359           break;
360
361         case iMultiMonitors:
362           EraseRect(&gWindowPtr->portRect);
363           gMultiMonitorsDrawDemo = true;
364           InvalRect(&gWindowPtr->portRect);
365           break;
366       }
367   }
368
369   HiliteMenu(0);
370 }
371
372 // ############################################################################ unloadSegments
373
374 void  unloadSegments(void)
375 {
376   UnloadSeg(demosSegment);
377 }
378
```

```
// ############################################################################################

// ############################################################################################
// Demos.c
// ############################################################################################

#include "Miscellany.h"

// ……… global variables

NMRec                       gNotificationRecord;
SInt32                      gStartingTickCount;
Boolean                     gNotificationDemoInvoked;
Boolean                     gNotificationInQueue;
Boolean                     gInBackground;
extern WindowPtr            gWindowPtr;
extern Boolean              gColorQuickDraw;
extern ProcessSerialNumber  gProcessSerNum;

// ########################################################################## demosSegment

void  demosSegment(void) {}

// ############################################################ doCommandPeriodAndStatusBar

void  doCommandPeriodAndStatusBar(void)
{
  DialogPtr modalDlgPtr;
  RGBColor  barBackColour, barColour;
  SInt16    itemType;
  Handle    itemHdl;
  Rect      itemRect;
  SInt16    a, b, c;
  Handle    soundHdl;
  Rect      theRect;
  SInt16    statusMax, statusCurrent;
  SInt32    finalTicks;

  EraseRect(&gWindowPtr->portRect);

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

  DrawDialog(modalDlgPtr);
  SetPort(modalDlgPtr);

  if(gColorQuickDraw)
  {
    barBackColour.red   = 0xBFFF;
    barBackColour.green = 0XBFFF;
    barBackColour.blue  = 0XFFFF;

    barColour.red   = 0x6FFF;
    barColour.green = 0x6FFF;
    barColour.blue  = 0x6FFF;
  }

  GetDialogItem(modalDlgPtr,iUserItem,&itemType,&itemHdl,&itemRect);
  InsetRect(&itemRect,-1,-1);
  FrameRect(&itemRect);
  InsetRect(&itemRect,1,1);

  if(gColorQuickDraw)
  {
    RGBBackColor(&barBackColour);
    FillRect(&itemRect,&qd.white);
    RGBForeColor(&barColour);
  }

  SetPort(gWindowPtr);

  statusMax = 2184;
  statusCurrent = 0;

  for(a=0;a<8;a++)
  {
```

```
455       if(doCheckForCommandPeriod() == true)
456       {
457         soundHdl = GetResource('snd ',rBarkSound);
458         SndPlay(NULL,(SndListHandle) soundHdl,false);
459         ReleaseResource(soundHdl);
460         DisposeDialog(modalDlgPtr);
461
462         SetPort(gWindowPtr);
463         EraseRect(&gWindowPtr->portRect);
464         MoveTo(115,110);
465         ForeColor(blackColor);
466         DrawString("\pOperation cancelled at user request");
467
468         return;
469       }
470       for(b=12;b<373;b+=18)
471       {
472         for(c=8;c<232;c+=18)
473         {
474           SetRect(&theRect,b+a,c+a,b+16-a,c+16-a);
475           if(a < 3)               ForeColor(redColor);
476           else if(a > 2 && a < 6)  ForeColor(greenColor);
477           else if(a > 5)          ForeColor(blueColor);
478           FrameRect(&theRect);
479
480           doDrawStatusBar(modalDlgPtr,itemRect,statusCurrent ++,statusMax);
481         }
482         Delay(2,&finalTicks);
483       }
484     }
485
486     DisposeDialog(modalDlgPtr);
487     EraseRect(&gWindowPtr->portRect);
488     MoveTo(150,110);
489     ForeColor(blackColor);
490     DrawString("\pOperation completed");
491   }
492
493   // ########################################################################## doDrawStatusBar
494
495   void  doDrawStatusBar(DialogPtr modalDlgPtr,Rect barRect,SInt16 statusCurrent,
496                         SInt16 statusMax)
497   {
498     SInt16  barMaxWidth;
499     float   barRequiredWidth;
500
501     SetPort(modalDlgPtr);
502
503     barMaxWidth = barRect.right - barRect.left;
504     barRequiredWidth = ((float) statusCurrent * (float) barMaxWidth) / statusMax;
505     barRect.right = barRect.left + (SInt16) barRequiredWidth;
506
507     if(gColorQuickDraw)
508       FillRect(&barRect,&qd.black);
509     else
510       FillRect(&barRect,&qd.gray);
511
512     SetPort(gWindowPtr);
513   }
514
515   // ############################################################## doCheckForCommandPeriod
516
517   Boolean  doCheckForCommandPeriod(void)
518   {
519     Boolean   foundCommandPeriod;
520     QHdrPtr   eventQHdrPtr;
521     EvQElPtr  eventQElPtr;
522     SInt32    keyCode;
523     SInt32    commandKeyDown;
524
525     foundCommandPeriod = false;
526
527     eventQHdrPtr = LMGetEventQueue();
528     eventQElPtr = (EvQElPtr) eventQHdrPtr->qHead;
529
530     while(eventQElPtr && !foundCommandPeriod)
```

```
  {
    if(eventQElPtr->evtQWhat == keyDown)
    {
      keyCode = eventQElPtr->evtQMessage & keyCodeMask;
      keyCode = keyCode >> 8;

      commandKeyDown = eventQElPtr->evtQModifiers & cmdKey;

      if(commandKeyDown)
        if(keyCode == 0x2f)
          foundCommandPeriod = true;
    }

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

  return(foundCommandPeriod);
}

// ########################################################################## doSetUpNotification

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

  gStartingTickCount = TickCount();

  MoveTo(12,100);
  DrawString("\pPlease click on the desktop now to make the Finder ");
  DrawString("\pthe frontmost application.");
  MoveTo(42,120);
  DrawString("\p(This application will post a notification 10 seconds from now.)");
}

// ########################################################### doPrepareNotificationRecord

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

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

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

// ############################################################################## doNullEvent

void  doNullEvent(void)
{
  ProcessSerialNumber frontProcessSerNum;
  Boolean             isSameProcess;

  if(gNotificationDemoInvoked)
  {
    if(TickCount() > gStartingTickCount + 600)
    {
      GetFrontProcess(&frontProcessSerNum);
      SameProcess(&frontProcessSerNum,&gProcessSerNum,&isSameProcess);
      if(!isSameProcess)
      {
        NMInstall(&gNotificationRecord);
        gNotificationDemoInvoked = false;
        gNotificationInQueue = true;
      }
```

```
607         else
608         {
609           doDisplayMessageToUser();
610           gNotificationDemoInvoked = false;
611         }
612
613         EraseRect(&gWindowPtr->portRect);
614       }
615     }
616   }
617
618   // ############################################################################## doOSEvent
619
620   void  doOSEvent(EventRecord *eventRecPtr)
621   {
622     switch((eventRecPtr->message >> 24) & 0x000000FF)
623     {
624       case suspendResumeMessage:
625         gInBackground = (eventRecPtr->message & resumeFlag) == 0;
626         if((!gInBackground) && gNotificationInQueue)
627           doDisplayMessageToUser();
628         break;
629
630       case mouseMovedMessage:
631         break;
632     }
633   }
634
635   // ########################################################### doDisplayMessageToUser
636
637   void  doDisplayMessageToUser(void)
638   {
639     if(gNotificationInQueue)
640     {
641       NMRemove(&gNotificationRecord);
642       gNotificationInQueue = false;
643     }
644
645     NoteAlert(rAlert,NULL);
646
647     DisposeIconSuite(gNotificationRecord.nmIcon,false);
648     ReleaseResource(gNotificationRecord.nmSound);
649     ReleaseResource((Handle) gNotificationRecord.nmStr);
650   }
651
652   // ############################################################################ doColourPicker
653
654   void  doColourPicker(void)
655   {
656     RGBColor  inColour, outColour, blackColour;
657     Rect      theRect;
658     Point     where;
659     Str255    prompt = "\pChoose a rectangle colour:";
660     Boolean   okButton;
661     char      *cString;
662
663     EraseRect(&gWindowPtr->portRect);
664
665     inColour.red    = 0xFFFF;
666     inColour.green  = 0x0000;
667     inColour.blue   = 0x0000;
668
669     blackColour.red   = 0x0000;
670     blackColour.green = 0x0000;
671     blackColour.blue  = 0x0000;
672
673     theRect = gWindowPtr->portRect;
674     InsetRect(&theRect,50,50);
675     RGBForeColor(&inColour);
676     FillRect(&theRect,&qd.black);
677
678     where.v = 0;
679     where.h = 0;
680
681     okButton = GetColor(where,prompt,&inColour,&outColour);
682
```

```
  if(okButton)
  {
    RGBForeColor(&outColour);
    FillRect(&theRect,&qd.black);
    RGBForeColor(&blackColour);

    MoveTo(50,20);
    DrawString("\pRed Value: ");
    cString = doDecimalToHexadecimal(outColour.red);
    MoveTo(115,20);
    DrawText(cString,0,6);

    MoveTo(50,33);
    DrawString("\pGreen Value: ");
    cString = doDecimalToHexadecimal(outColour.green);
    MoveTo(115,33);
    DrawText(cString,0,6);

    MoveTo(50,46);
    DrawString("\pBlue Value: ");
    cString = doDecimalToHexadecimal(outColour.blue);
    MoveTo(115,46);
    DrawText(cString,0,6);
  }
  else
  {
    RGBForeColor(&inColour);
    FillRect(&theRect,&qd.black);
    RGBForeColor(&blackColour);
    MoveTo(75,125);
    DrawString("\pCancel button was clicked.  Rectangle remains red.");
  }
}

// ############################################################## 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;
}

// ########################################################## doCheckSlotVInstallAvailable

Boolean  doCheckSlotVInstallAvailable(void)
{
  return trapAvailable(_SlotVInstall);
}

// ################################################################################ trapAvailable

Boolean  trapAvailable(SInt16 theTrap)
{
  TrapType  trapType;
  SInt16    trapMask = 0x0800;
  SInt16    numToolboxTraps;

  if((theTrap & trapMask) > 0)
    trapType = ToolTrap;
  else
    trapType = OSTrap;

  if(trapType == ToolTrap)
    theTrap = theTrap & 0x07FF;

  if(NGetTrapAddress(_InitGraf,ToolTrap) == NGetTrapAddress(0xAA6E,ToolTrap))
    numToolboxTraps = 0x0200;
  else
    numToolboxTraps = 0x0400;

```

```
  if(theTrap >= numToolboxTraps)
    theTrap = _Unimplemented;

  return(NGetTrapAddress(theTrap,trapType) != NGetTrapAddress(_Unimplemented,ToolTrap));
}

// ########################################################################## doDeviceLoopDraw

pascal void  doDeviceLoopDraw(SInt16 depth,SInt16 deviceFlags,GDHandle targetDeviceHdl,
                              SInt32 userData)
{
  WindowPtr windowPtr;
  Rect      theRect;
  RGBColor  oldForeColour;
  RGBColor  green = { 0x6666, 0xFFFF, 0x6666 };
  RGBColor  red   = { 0xFFFF, 0x6666, 0x6666 };
  RGBColor  blue  = { 0x9999, 0x9999, 0xFFFF };

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

  switch(depth)
  {
    case 1:
    case 2:
      SetRect(&theRect,70,40,320,200);
      FillRect(&theRect,&qd.ltGray);
      InsetRect(&theRect,30,30);
      FillRect(&theRect,&qd.gray);
      InsetRect(&theRect,30,30);
      FillRect(&theRect,&qd.dkGray);
      break;

    case 4:
    case 8:
    case 16:
    case 32:
      GetForeColor(&oldForeColour);
      SetRect(&theRect,70,40,320,200);
      RGBForeColor(&green);
      PaintRect(&theRect);
      InsetRect(&theRect,30,30);
      RGBForeColor(&red);
      PaintRect(&theRect);
      InsetRect(&theRect,30,30);
      RGBForeColor(&blue);
      PaintRect(&theRect);
      RGBForeColor(&oldForeColour);
      break;
  }
}

// ########################################################################### doZoomWindow

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

  GetPort(&oldPort);
  SetPort(windowPtr);

  EraseRect(&windowPtr->portRect);

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

  if(zoomInOrOut == inZoomOut)
  {
    if(!gColorQuickDraw)
```

```
    {
      zoomRect = qd.screenBits.bounds;
      zoomRect.top = zoomRect.top + LMGetMBarHeight() + titleBarHeight;
      InsetRect(&zoomRect,3,3);

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

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

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

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

// ########################################################################## doResizeWindow

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

  InvalRect(&windowPtr->portRect);
}

// ############################################################################################
```

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 Command-Period and Status Bar item, noting that the status bar dialog box is disposed of when the (simulated) time-consuming task concludes.

- Choose the Command-Period and Status Bar item again, and this time press the Command-period key combination before the (simulated) time-consuming task concludes. Note that

the status bar dialog box is disposed of when the Command-period key combination is pressed.

- 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 and alternates the Finder and Miscellany icons in the menu bar above the Application menu. Observing the instructions in the alert box, dismiss the alert 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 Color Picker item and make colour choices using both the HSL and RGB modes. Note that, when the Color Picker is dismissed by clicking the OK button, the RGB colour values for the chosen colour are displayed in hexadecimal, together with a rectangle in that colour, in the Miscellany window.

- Choose the Trap Available Check item, noting the result returned by the functions which perform this check. For the purposes of demonstration, the trap checked for is _SlotVInstall, which is not available on black-and-white Macintoshes.

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

    - On a monitor set to bit depths of 1 (black-and-white) and 2 (four colours), the image is drawn in black-and-white using the patterns ltGray, Gray, and dkGray.

    - On a monitor set to bit depths of 4 (16 colours) and higher, the image is drawn in three colours.

    (If the user's system does not have more than one monitor, this aspect of multiple monitors handling can nonetheless be demonstrated by opening the Monitors control panel after the Multiple Monitors Draw item has been chosen, selecting various colours and grays settings (and the black-and-white setting), and observing the effects on the demonstration image.)

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.

Note that the notification demonstration follows the same notification sequence as does PrintMonitor. This particular demonstration does not, therefore, involve a response procedure.

Miscellany.h

Because the source code is divided into two files (main.c and demos.c), all #includes, #defines and function prototypes have been placed in Miscellany.h, which is included by both main.c and demos.c.

#define

Lines 82-93 establish constants relating to menu and window resource IDs, and to menu item numbers. Lines 94-99 establish constants relating to resources. Lines 100-101 define two common macros. The first converts the top and left fields of a Rect to a Point. The second converts the bottom and right field of a Rect to a Point.

main.c

Global Variables

gColorQuickDraw will be set to true if Color QuickDraw is present. gDone controls program termination. gWindowPtr will be assigned the pointer to the window opened by the program. gProcessSerialNum will be assigned the process serial number of the Miscellany application.

gMultiMonitorsDrawDemo will be set to true when the Multiple Monitors Draw item in the Demonstration menu is chosen.

main

The main function initialises the system software managers (Line 162), establishes whether Color QuickDraw is present (Lines 166-170), sets up the menus (Lines 174-184), opens a window and sets the text size (Lines 188-197), gets the process serial number of this process (Line 201), and enters the main event loop (Lines 205-212).

Note that, at Line 214, 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 and doMouseDown

doEvents and doMouseDown perform minimal initial event handling consistent with the satisfactory execution of the demonstration.

Note that, in the case of an update event which occurs after the Multiple Monitors Draw item in the Demonstration menu has been chosen (Line 253), a call is made to DeviceLoop and the address of the application-defined drawing procedure doDeviceLoopDraw is passed as the second parameter in this call (Lines 255-256).

Also note that, in the case of a mouseDown event in the window's zoom box, the application-defined function doZoomWindowMultiMonitors is called if the cursor is still within the zoom box when the mouse button is released (Lines 296-300).

doMenuChoice

doMenuChoice further processes menu choices.

Lines 335-366 respond to choices from the Demonstration menu. Note that, at Lines 352-359, one string or other will be drawn in the window depending on whether the trap-available check returns true or false. Also note that, when the Multiple Monitors Draw item is chosen, the global variable gMultiMonitorsDrawDemo is set to true and the window's port rectangle is invalidated so as to force an update event and consequential call to DeviceLoop (Lines 361-365).

unloadSegments

unloadSegments unlocks, and marks as purgeable, the specified code segment, that is, the segment in which the stub ("do nothing" routine) demosSegment resides.

demos.c

Global Variables

gNotificationRecord's fields will be assigned values prior to the installation of the notification request into the notification queue. gStartingTickCount will be assigned the number of ticks since system startup, and gNotificationDemoInvoked will be set to true, at the time that the user chooses Notification from the Demonstration menu. gNotificationInQueue will be set to true after NMInstall is called to install the notification request in the queue. gInBackground relates to foreground/background switching.

demosSegment

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

doCommandPeriodAndStatusBar

doCommandPeriodAndStatusBar is called when the user chooses Command-Period and Status Bar from the Demonstration menu.

Line 417 erases the window's content region. Line 419 opens a dialog box using the specified resource. Line 422 draws the contents of the dialog box (two static text items) and Line 423 sets the dialog box's graphics port as the current port preparatory to the drawing of the status bar's box. If Color QuickDraw is present (Line 425), Lines 426-434 establish the colours to be used for the status bar's background colour (light blue) and the moving status bar itself (grey).

One dialog box item is a user item. This item's rectangle is used to define the size and location of the status bar's box. The call to GetDItem at Line 436 gets this rectangle. Line 437 then expands this rectangle by one pixel all around before Line 438 draws a rectangle frame. Line 439 returns the rectangle to its original size. If Color QuickDraw is present (Line 441), Lines 442-446 fill the rectangle with the status bar background colour and then set the foreground colour to the moving status bar colour. That done, Line 448 sets the window's graphics port as the current port.

Lines 453-488 will perform a simulated time-consuming task, represented to the user by the drawing of a large number of coloured rectangles in the window. The task involves 2184 calls to FrameRect. Accordingly, Line 450 assigns a value representing the number of steps in the task to a variable. Line 451 sets a variable to indicate that none of these steps has yet been completed.

Within the outer loop initiated at Line 453, Line 455 calls an application-defined function which checks whether the user has pressed the Command-period key combination. If this key press has occurred, Lines 457-469 execute. Specifically, Lines 457-458 load a 'snd ' resource and play the sound, Line 459 disposes of the resource, Line 460 disposes of the dialog box, Lines 463-466 draw an advisory message in the window, and Line 468 causes doCommandPeriodAndStatusBar to return.

Within the inner of the three loops, the rectangles are drawn (Lines 472-481). Each time round this inner loop, an application-defined function is called (Line 480) to redraw the moving status bar according to the value in the variable statusCurrent, which is incremented on each cycle of the inner loop.

When the outer loop exits (that is, when the Command-period key combination is not pressed before the simulated time-consuming task completes) Line 486 disposes of the dialog, and Lines 487-490 draw an advisory message in the window.

doDrawStatusBar

doDrawStatusBar draws the moving status bar.

Line 501 sets the dialog box's graphics port as the current graphics port. Lines 503-505 define a rectangle so that the left, top, and bottom fields equate to those of the user item rectangle, with the right field being assigned a value which bears the same relationship to the total width of the status bar's box as does the variable statusCurrent to the variable statusMax. Lines 507-510 draw the moving status bar in the previously set grey colour (Color QuickDraw present) or in the gray pattern (Color QuickDraw not present). That done, Line 512 sets the window's graphics port as the current graphics port.

doCheckForCommandPeriod

doCheckForCommandPeriod scans the event queue for a Command-period keyboard event.

Line 525 sets a variable so as to begin by assuming that such an event is not in the queue.

Line 527 gets a pointer to the event queue header. Line 528 gets a pointer to first queue element. Line 530 initiates a loop which will scan the whole of the event queue, exiting only when a Command-period key event is found in the queue or, if no such event is found, the entire queue has been scanned.

Inside the loop, if a key-down event is found (Line 532), Lines 534-535 get the key code and Line 537 checks whether the Command key was down. If the Command key was down (Line 539), and if the period key was the key pressed (Line 540), the variable foundCommandPeriod is set to true (Line 541), causing the loop to exit. Otherwise, the loop calls up the next queue entry for examination (Lines 544-545).

Line 548 returns the result of the search.

doSetUpNotification

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

Line 555 calls an application-defined function which fills in the relevant fields of a notification record. That done, Line 556 assigns true to a global variable which records that the Notification item has been chosen by the user.

Line 558 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 Line 558. Lines 560-564 simply draw some advisory text in the window.

doPrepareNotificationRecord

doPrepareNotificationRecord fills in the relevant fields of the notification record.

First, however, Line 575 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. Line 576 loads the specified 'snd ' resource and gets its handle. Line 577 loads the specified 'STR ' resource and gets its handle.

Line 579 specifies the type of operating system queue. Line 580 specifies that the ◆ mark is to appear next to the application's name in the Application menu. Lines 581-583 assign the icon suite, sound and string handles previously obtained. Line 584 specifies that no response procedure is required to be executed when the notification is posted.

doNullEvent

doNullEvent is called from the main event loop when a null event is received. (Note from Line 209 that the sleep parameter in the WaitNextEvent call is set to 30 (half a second) so that doNullEvent is called fairly frequently. Also, recall that the canBackground flag is set, 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 (Line 595), doNullEvent simply returns immediately.

If, however, that item has just been chosen (Line 595), and if 10 seconds have elapsed since that choice was made (Line 597), the following occurs:

- Lines 599-600 determine whether the current foreground process is Miscellany. If it is not, the notification request is installed in the notification queue (Line 603) and a global variable is set to indicate that a request has been placed in the queue by Miscellany (Line 605). Also, Line 604 resets the gNotificationDemoInvoked variable to false so as to ensure that Lines 596-615 only execute once after the Notification item is chosen.

- If, however, the current foreground process is Miscellany (Line 607), an application-defined function is called to present the required message to the user, via an alert box, in the normal way (Line 609). Once again gNotificationDemoInvoked is reset to false so as to ensure that Lines 596-615 only execute once after the Notification item is chosen.

doOSEvent

doOSEvent handles operating system events.

If the event is a resume event (that is, Miscellany is now in the foreground) and if the notification request is still in the notification queue (Line 626), an application-defined function is called to remove the notification request from the queue and have Miscellany convey the required message to the user via an alert box (Line 627).

doDisplayMessageToUser

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

If a Miscellany notification request is in the queue (Line 639), Lines 641-642 remove it from the queue and set the gNotificationInQueue variable to reflect this condition. (Recall that, if the nmResp field of the notification record 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 presents its alert at Line 645 and the notification's icon suite, sound and string resources are released/disposed of (Lines 647-649).

doColourPicker

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

Line 663 erases the window's content region. Lines 665-667 assign red to the RGBColor variable to be specified as the inColor parameter of the GetColor call at Line 681. Lines 669-671 assign black to another RGB colour variable.

Lines 673-676 draw a filled rectangle in the window in the inColor colour (red). Lines 678-679 assign 0 to the fields of the Point variable used as the first parameter in the GetColor call at Line 699. ((0,0) will cause the Color Picker to be centered on the main screen.)

Line 681 displays the Color Picker's dialog box. GetColor retains control until the user clicks either the OK button or the Cancel button.

If the user clicks the OK button (Line 683), Lines 685-705 draw a filled rectangle 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 Lines 691, 697, and 703 call an application-defined function to convert the decimal (unsigned short) values in the fields of the RGBColor variable outColor to hexadecimal.

If the user clicks the Cancel button (Line 707), a filled rectangle is drawn in the window in the colour returned in GetColor's outColor parameter. (In this instance, since the Cancel button was clicked, GetColor simply assigns the value in inColour to outColour. The rectangle is thus drawn in the original red.)

doDecimalToHexadecimal

doDecimalToHexadecimal converts an unsigned short to a hexadecimal string.

doCheckSlotVInstallAvailable

doCheckSlotVInstallAvailable is called when the user chooses Trap Available Check from the Demonstration Menu. It specifies the trap SlotVInstall, calls the application-defined function which checks whether that trap is available, and returns the result of the check.

trapAvailable

trapAvailable checks for the existence of the trap passed to it in the theTrap parameter.

Before explaining the code, some backgound is necessary. All system routines are numbered, and their addresses are contained in a table in RAM called the trap dispatch table. Routines which are not implemented are also included in this table . Unimplemented routines contain the address of a special "unimplemented trap" handler. This means that you can determine whether a trap is implemented by finding its address and comparing it with the address of the unimplemented trap handler. If the two are the same, the trap in question is not implemented.

There is, however, a complication: there are two different sizes of trap tables. The original trap table had room for 512 Toolbox traps; the newer trap table has room for 1024.

With the introduction of the larger trap table, bit 9 of the trap word was used to distinguish between the original traps and the newly-defined traps. Now, it so happens that, if you call NGetTrapAddress to get the address of one of the new traps on a machine with the old-size trap table, NGetTrapAddress will turn off bit 9 of the value passed in the trapNum parameter before looking up and returning the address. You can take advantage of this behaviour to determine which sized trap table is present.

The procedure is to call NGetTrapAddress twice, using two traps which differ only in their setting of bit 9, and compare the result. (You must ensure, of course, that at least one of these traps is sure to exist regardless of the trap table size present. _InitGraf (0xA86E) is a good choice in this regard. If you use _InitGraf in the first call, the second call would use 0xAA6E (that is, 0xA86E with bit nine set).) If the addresses returned by NGetTrapAddress are the same, then NGetTrapAddress must have turned off bit 9 of the trapNum parameter in the second call, meaning that the new size trap table is not present.

One further detail remains: there are two types of traps (Toolbox traps and Operating System traps) and you must pass the appropriate type in NGetTrapAddress' trapType parameter. Resolving this issue is relatively straightforward, however. Operating System traps are numbered in the range 0xA000 to 0xA7FF and Toolbox traps are numbered in the range 0xA800 to 0xAFFF. Thus bit 11 of the trap word will be on if the trap is a Toolbox trap but not if it is an Operating System trap. Accordingly, all that is required is to test bit 11 of the trap number.

Also of relevance is the fact that all system routines on the 680X0 Macintosh are implemented as so-called A-traps, that is, Motorola 68000 instructions which begin with the digit 0xA. 68000 instructions are 16 bits long and the 0xA takes the first four bits, leaving the least significant 12 bits to define the rest of the trap.

Now to the code.

Lines 746-749 determine whether the trap is a Toolbox trap or an Operating System trap by testing bit 11.

If the trap is a Toolbox trap, Lines 751-752 change the value in theTrap to the value which would obtain if Toolbox traps were numbered from 0xA000 rather than from 0xA800.

Lines 754-757 get the size of the trap table. If the value in the variable theTrap is such that the trap cannot be present in a table of this size (Line 759), then the trap is clearly not present. Accordingly, Line 760 changes the value in theTrap to _Unimplemented, in which case Line 762 will return false.

On the other hand, even if the trap number is within the size of the trap table present, the check at Line 762 is still required. In this case, Line 762 will only return true if the addresses returned by the two calls to NGetTrapAddress are equal.

> POSSIBLE OBSOLETE CODE
>
> As stated above, the original trap table had room for 512 Toolbox traps, while the newer trap table has room for 1024. This latter has been the case since Color QuickDraw was introduced. Accordingly, if your application is not intended for machines without Color QuickDraw, Line 744 and Lines 754-760 (which check for the expanded trap table) may be regarded as obsolete code and may thus be deleted.

doDeviceLoopDraw

doDeviceLoopDraw is the drawing procedure whose address is passed as the second parameter in the DeviceLoop call at Line 256. (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.

Line 777 typecasts the long value received in the userData parameter to a WindowPtr. Line 778 erases the port rectangle of the specified window.

Line 780 switches according to the value received in the depth parameter. If the depth parameter indicates a pixel depth of 1 or 2, three overlapping rectangles are drawn using the ltGray, Gray, and dkGray patterns. If the depth parameter indicates a pixel depth of 4 to 32, the same rectangles are drawn, but in the colours green, red, and blue.

doZoomWindowMultiMonitors

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

Lines 822-823 save and set the current graphics port. Line 825 erases the window's port rectangle prior to the zoom so as to avoid flicker. Lines 827-830 get the height of the window's title bar, which will be used later if the window is being zoomed "out" to the standard state.

Lines 833-880 execute only if (Line 832) 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, which monitor the zoomed window is to be displayed on.

Multiple monitors cannot be supported unless Color QuickDraw is present. Accordingly, Line 834 determines if multiple monitors have to be catered for. If not, Lines 835-842 simply establish a rectangle three pixels inside the screen's gray area and assign this rectangle to the stdState field of the window's state data record.

If, on the other hand, the possibility of multiple monitors has to be catered for (that is, Color QuickDraw is present) (Line 843):

- Line 845 establishes a rectangle equal to the window's port rectangle, plus the window's title bar, in global coordinates. Line 847 gets a handle to the first gDevice record in the device list and Line 848 sets the variable greatestArea to 0. The while loop entered at Line 850 then walks the device list. For each active video device (Lines 852-853), the associated gDevice record'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 (Line 857).

- 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 (Line 862).

- Line 865 gets the handle to the next device in the device list. The while loop exits when this call returns NULL. When the while 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 this device is the main device (Line 869), the height of the menu bar is added to the value in the variable which holds the window's title bar height.

- Lines 872-875 then establish the standard state rectangle. This is three pixels inside the rectangle contained in the gdRect field of the device's gDevice record, 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). Lines 877-878 then assign this rectangle to the stdState field of the window's state data record.

Line 882 calls ZoomWindow to zoom the window in the appropriate direction, following which an application-defined function is called (Line 883) to redraw the window contents as appropriate. Finally, the saved graphics port is restored (Line 884).

doRedoWindowContent

doRedoWindowContent is called by doZoomWindowMultiMonitors to redraw the content region of a newly-zoomed window. Line 893 invalidates the window's port rectangle, forcing an update event.