22 Version 1.2 (Frozen)

MISCELLANY

Includes Demonstration Program MiscellanyPascal

Code Segmentation and Heap Space Optimisation

As stated in the CodeWarrior User's Guide, most Macintosh programs are made up of 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:

```
procedure updateSegment;
  begin
  end;
```

- 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:

```
procedure DoUnloadSegments;
  begin
  UnloadSeg(updateSegment);
  UnloadSeg(activateSegment);
  { Other UnloadSeg calls here as required.}
  end;
```

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:

```
begin
   ...
   while not(gDone) do
      begin
      if (WaitNextEvent(everyEvent, eventRec, kMaxLong, nil)) then
         DoEvents(eventRec);

      UnloadSegments;
      end;
end;
   {of main program block}
```

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 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 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 on 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 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:

```
NMRec = record
   qLink:       QElemPtr;  { Address of next element in queue. (Used internally.)}
   qType:       integer;   { Type of data. (8 = nmType).}
   nmFlags:     integer;   { (Reserved.)}
   nmPrivate:   longint;   { (Reserved.)}
   nmReserved:  integer;   { (Reserved.)}
   nmMark:      integer;   { Application to identify with   mark.}
   nmIcon:      Handle;    { Handle to small icon.}
   nmSound:     Handle;    { Handle to sound record.}
```

[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.

```
  nmStr:       StringPtr; { Pointer to string to appear in alert.}
  nmResp:      NMUPP;     { Pointer to response routine.}
  nmRefCon:    longint;   { Available for application use.}
  end;

NMRecPtr = ^NMRec;
```

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 `nil`, no icon appears in the menu bar. If `nmIcon` is not `nil`, 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 `nil`, 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 `''`, 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 `nil`, 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 `nil`. 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`:

```
procedure theResponse(nmRecordPtr: NMUPP);
```

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.

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.

---

[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.

`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.)

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.

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

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˚.

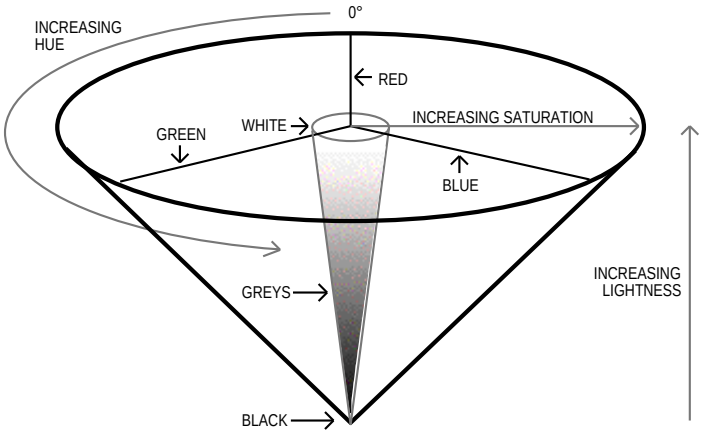

FIG 1 - HSL/HSV COLOUR CONE

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 which 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.

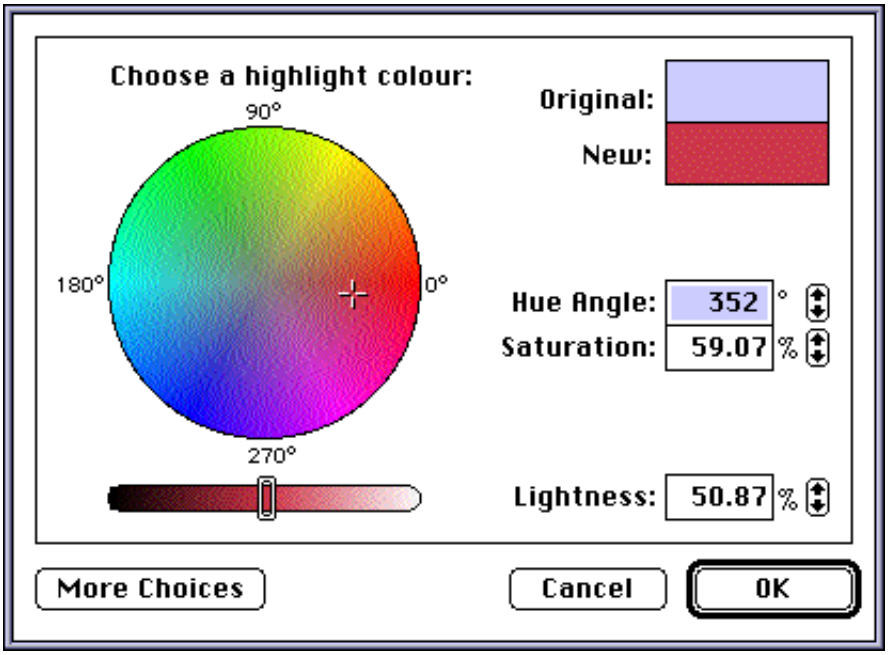

FIG 2 - COLOR PICKER DIALOG IN HSL MODE

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).

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.

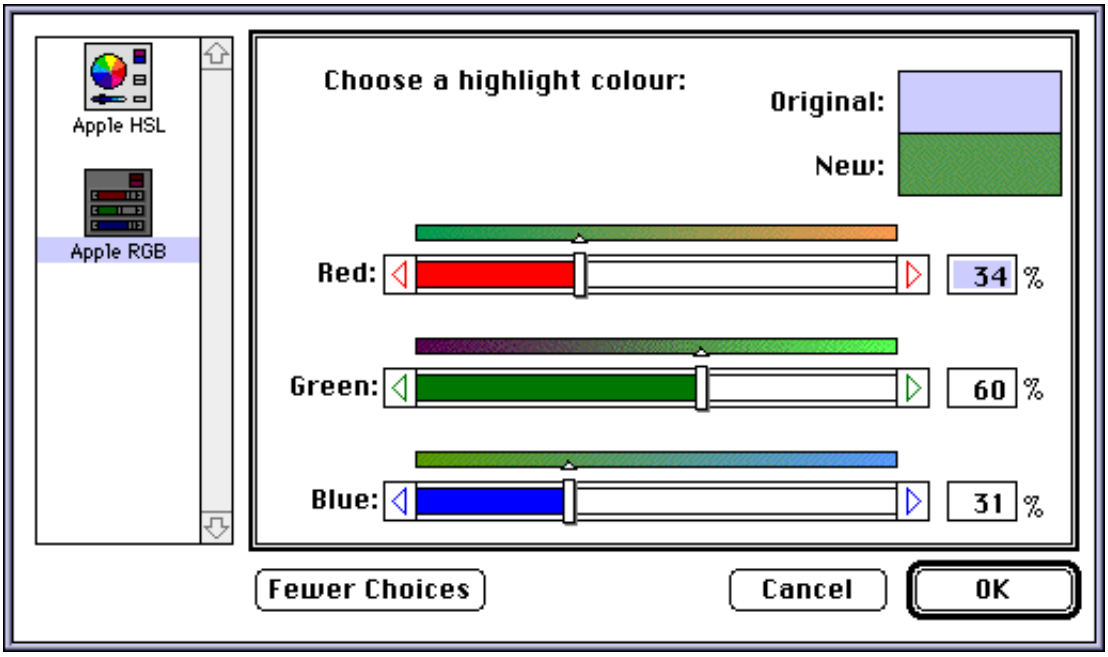

FIG 3 - COLOR PICKER DIALOG IN RGB MODE

Invoking the Color Picker

The Color Picker is invoked using the `GetColor` function:

```
function GetColor(where: Point; prompt: ConstStr255Param; var inColor: RGBColor;
                              var outColor: RGBColor): boolean;
```

`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]:

```
function Gestalt(selector: OSType; var response: longint): OSErr;
```

`selector` Selector code.

`response` 4-byte return result which provides the requested information. When all four bytes are not needed, the result is expressed in the low-order byte.

Returns: Error code. (`0` = no error.)

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

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

[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.

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 interface file GestaltEqu.p 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;
response : longint;
colorQuickDrawPresent : boolean;

colorQuickDrawPresent := true;

osErr := Gestalt(gestaltQuickdrawVersion, response);
if (osErr = noErr) then
  begin
  if (response < gestalt8BitQD) then
    colorQuickDrawPresent := false;
  end;
```

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;
response : longint;
hasSoundInputDevice : boolean;

hasSoundInputDevice := false;

osErr := Gestalt(gestaltSoundAttr, response);
if (osErr = noErr) then
  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 MC68000 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. 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[7], 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**[8], 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 1 or a 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

[7]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` organizes 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.

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

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:

```
procedure DeviceLoopDrawing(depth: integer; deviceFlags: integer;
                            targetDevice: GDHandle; userData: longint);
```

`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**[9], 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.

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.}
begin
```

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

```
...
targetDevice := LMGetMainDevice();
newGray := GetGray(targetDevice, backColour, newForeColour);
end;

if (newGray) then                    { 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:

```
function DoGetRectsDevice(theRect : Rect) : GDHandle;

var
SInt32  greatestArea, intersectArea;
GDHandledeviceHdl, deviceHdlToReturn;
Rect    intersectRect;
sectFlag : boolean;

begin
LocalToGlobal(theRect.topLeft);
LocalToGlobal(theRect.botRight);

deviceHdl := LMGetDeviceList;
greatestArea := 0;

while (deviceHdl <> nil) do
  begin
  if (TestDeviceAttribute(deviceHdl, screenDevice)) then
    if (TestDeviceAttribute(deviceHdl, screenActive)) then
      begin
      sectFlag := SectRect(theRect, deviceHdl^^.gdRect, intersectRect);

      intersectArea := longint((intersectRect.right - intersectRect.left) *
                       (intersectRect.bottom - intersectRect.top));

      if (intersectArea > greatestArea) then
        begin
        greatestArea := intersectArea;
        deviceHdlToReturn := deviceHdl;
        end;

      deviceHdl := GetNextDevice(deviceHdl);
      end;
  end;

DoGetRectsDevice := deviceHdlToReturn;
end;
```

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

```
procedure UnloadSeg(routineAddr: UNIV Ptr);
```

Terminate Caller, Release Heap, and Launch Finder

```
procedure ExitToShell;
```

Main Event Manager Data Types and Routines

Data Types

QHdr (Defines the Queue Header)

```
QHdr = record
  qFlags:   integer;
  qHead:    QElemPtr;
  qTail:    QElemPtr;
  end;

QHdrPtr = ^QHdr;
```

QElem

```
QElem = record
  qLink:    QElemPtr;
  qType:    integer;
  qData:    array [0..0] of integer;
  end;

QElemPtr = ^QElem;
```

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

```
EvQEl = record
  qLink:         QElemPtr;
  qType:         integer;
  evtQWhat:      EventKind;          { this part is identical to the EventRecord as... }
  evtQMessage:   UInt32;               { defined above }
  evtQWhen:      UInt32;
  evtQWhere:     Point;
  evtQModifiers:EventModifiers;
  end;

EvQElPtr = ^EvQEl;
```

Routines

Get Address of Event Queue Header

```
function  LMGetEventQueue: QHdrPtr;
```

Main Notification Manager Data Types and Routines

Data Types

Notification Record

```
NMRec = record
  qLink:      QElemPtr;   { next queue entry}
  qType:      integer;    { queue type -- ORD(nmType) = 8}
  nmFlags:    integer;    { reserved}
  nmPrivate:  longint;    { reserved}
  nmReserved: integer;    { reserved}
  nmMark:     integer;    { item to mark in Apple menu}
  nmIcon:     Handle;     { handle to small icon}
  nmSound:    Handle;     { handle to sound record}
  nmStr:      StringPtr;  { string to appear in alert}
  nmResp:     NMUPP;      { pointer to response routine}
  nmRefCon:   longint;    { for application use}
  end;

NMRecPtr = ^NMRec;
```

Routines

Add Notification Request to the Notification Queue

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

Remove Notification Request from the Notification Queue

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

Main Process Manager Data Types and Routines

Data Types

Process Serial Number

```
ProcessSerialNumber = record
  highLongOfPSN:  longint;
  lowLongOfPSN:   longint;
  end;

ProcessSerialNumberPtr = ^ProcessSerialNumber;
```

Routines

Get Process Serial Number of a Particular Process

```
functionGetCurrentProcess(var PSN: ProcessSerialNumber): OSErr;
```

Get Process Serial Number of Foreground Process

```
functionGetFrontProcess(var PSN: ProcessSerialNumber): OSErr;
```

Compare Two Process Serial Numbers

```
functionSameProcess(var PSN1: ProcessSerialNumber; var PSN2: ProcessSerialNumber; var result:
      boolean): OSErr;
```

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               = $000   { Original 1-bit QD.}
gestalt8BitQD                   = $100   { 8-bit color QD.}
gestalt32BitQD                  = $200   { 32-bit color QD.}
gestalt32BitQD11                = $210   { 32-bit color QDv1.1.}
gestalt32BitQD12                = $220   { 32-bit color QDv1.2.}
gestalt32BitQD13                = $230   { 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

```
function   Gestalt(selector: OSType; var response: longint): OSErr;
```

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

```
function   LMGetDeviceList : GDHandle;
function   LMGetMainDevice : GDHandle;
function   GetNextDevice(curDevice: GDHandle): GDHandle;
```

Determining the Characteristics of a Video Device

```
procedure  DeviceLoop(drawingRgn: RgnHandle; drawingProc: DeviceLoopDrawingUPP;
           userData: longint; flags: DeviceLoopFlags);
function   TestDeviceAttribute(gdh: GDHandle; attribute: integer): boolean;
```

Getting the Intersection Between Two Rectangles and Determining the Overlap

```
function   SectRect(var src1: Rect; var src2: Rect; var dstRect: Rect): boolean;
```

Demonstration Program

```
1   { ############################################################################################
2   // MiscellanyPascal.p
3   // ############################################################################################
4   //
5   // Miscellany source code is contained in three files, namely, UMain.p, UDemos.p and
6   // MiscellanyPascal.p  Within the CodeWarrior project, MiscellanyPascal.p and UMain.p
7   // are in Segment 1, while UDemos.p is in Segment 2.
8   // (Note that this small program does not really require such segmentation; the
9   // code is segmented only to facilitate demonstration of the Segment Loader aspects.
10  //
11  // This program demonstrates:
12  //
13  // •  The use of stubs in code segments, together with a function which uses those stubs
```

```
14  //    to unlock code segments and make them purgeable.
15  //
16  // •  The use of a status bar to graphically indicate the current status of a time-
17  //    consuming operation.
18  //
19  // •  The use of the Command-period key combination to terminate a time-consuming
20  //    operation before it concludes.
21  //
22  // •  The use of the Notification Manager to allow an application running in the
23  //    background to communicate with the foreground application.
24  //
25  // •  The determination of whether a particular application is currently the foreground
26  //    application.
27  //
28  // •  The use of the Color Picker to solicit a choice of colour from the user.
29  //
30  // •  The determination of whether a particular trap is available.
31  //
32  // •  Image drawing optimisation and window zooming in a multi-monitors environment.
33  //
34  // The program utilises the following resources:
35  //
36  // •  An 'MBAR' resource, and 'MENU' resources for Apple, File, Edit and Demonstration
37  //    menus (preload, non-purgeable).
38  //
39  // •  A 'WIND' resource (purgeable) (initially visible) for a window in which graphics
40  //    and information relevant to the demonstrations is displayed.
41  //
42  // •  An 'ALRT' resource (purgeable), and associated 'DITL' resource (purgeable), for
43  //    displaying a message to the user from within the Notification Manager demonstration.
44  //
45  // •  A 'DLOG' resource (purgeable), and associated 'DITL' and 'dctb' resources
46  //    (purgeable), for a dialog box in which the status bar is displayed.
47  //
48  // •  'icn#', 'ics4', and 'ics8' resources (non-purgeable) which contain the Miscellany
49  //    application icon displayed in the Application menu during the Notification Manager
50  //    demonstration.
51  //
52  // •  A 'snd ' resource (non-purgeable) used in the Notification Manager demonstration.
53  //
54  // •  A 'STR ' resource (non-purgeable) containing the text displayed in the alert box
55  //    invoked by the Notification Manager.
56  //
57  // •  A 'SIZE' resource with the acceptSuspendResumeEvents doesActivateOnFGSwitch,
58  //    canBackgound, and is32BitCompatible flags set.
59  //
60  // ############################################################################################ }
61
62  program MiscellanyPascal(input, output);
63
64  { …… include the following Universal Interfaces }
65
66  uses
67
68    Windows, Fonts, Menus, TextEdit, Quickdraw, Dialogs, QuickdrawText, Processes, Types,
69    Memory, Events, TextUtils, ToolUtils, OSUtils, Devices, TextUtils,
70
71  { …… include the following user-defined units }
72
73    UMain, UDemos;
74
75  { …… global variables }
76
77  var
78
79  gColorQuickDraw : boolean; external;
80  gDone : boolean; external;
81  gWindowPtr : WindowPtr; external;
82  gProcessSerNum : ProcessSerialNumber; external;
83  gMultiMonitorsDrawDemo : boolean; external;
84
85  theErr : OSErr;
86  response : longint;
87  menubarHdl : Handle;
88  menuHdl : MenuHandle;
89  theEvent : EventRecord;
90
```

```
{ ################################################################## start of main program }

begin

  gMultiMonitorsDrawDemo := false;

  { ……… initialie managers }

  DoInitManagers;

  { ……… check for Color QuickDraw }

  gColorQuickDraw := true;

  theErr := Gestalt(gestaltQuickdrawVersion, response);
  if (response < gestalt8BitQD) then
    gColorQuickDraw := false;

  { ……… set up menu bar and menus }

  menubarHdl := GetNewMBar(rMenubar);
  if (menubarHdl = nil) then
    ExitToShell;
  SetMenuBar(menubarHdl);
  DrawMenuBar;

  menuHdl := GetMenuHandle(mApple);
  if (menuHdl = nil) then
    ExitToShell
  else
    AppendResMenu(menuHdl,'DRVR');

  { ……… open window }

  if (gColorQuickDraw) then
    gWindowPtr := GetNewCWindow(rWindow, nil, WindowPtr(-1))
  else
    gWindowPtr := GetNewWindow(rWindow, nil, WindowPtr(-1));

  if (gWindowPtr = nil) then
      ExitToShell;

  SetPort(gWindowPtr);
  TextSize(10);

  { ……… get process serial number of this process }

  theErr := GetCurrentProcess(gProcessSerNum);

  { ……… enter EventLoop }

  gDone := false;

  while not (gDone) do
    begin
    if (WaitNextEvent(everyEvent, theEvent, 30, nil)) then
      DoEvents(theEvent)
    else
      DoNullEvent;

    UnloadSegments;
    end;

end.

{ ############################################################################################ }

{ ############################################################################################
// UMain.p
// ############################################################################################ }

unit UMain;



interface

```

```
{ ……… include the following Universal Interfaces }

uses

  Windows, Fonts, Menus, TextEdit, Dialogs, SegLoad, ToolUtils, Devices, GestaltEqu,
  Resources, Sound, Notification, Icons, Processes, ColorPicker, Traps, LowMem,

{ ……… include the following user-defined units }

  UDemos;

{ ……… define the following constants }

const

mApple = 128;
 iAbout = 1;
mFile = 129;
 iQuit = 11;
mDemonstration = 131;
 iCommandPeriod = 1;
 iNotification = 2;
 iColourPicker = 3;
 iTrapAvailable = 4;
 iMultiMonitors = 5;

rMenubar = 128;
rWindow = 128;
rAlert = 128;
rDialog = 129;
 iUserItem = 1;
rIconFamily = 128;
rBarkSound = 8192;
rString = 128;

{ ……… global variables }

var

gColorQuickDraw : boolean;
gDone : boolean;
gWindowPtr : WindowPtr;
gProcessSerNum : ProcessSerialNumber;
gMultiMonitorsDrawDemo : boolean;

{ ……… function and procedure interfaces }

  procedure DoInitManagers;
  procedure DoEvents(theEvent : EventRecord);
  procedure DoMouseDown(theEvent : EventRecord);
  procedure DoMenuChoice(menuChoice : longint);
  procedure UnloadSegments;



implementation

{ ########################################################################## DoInitManagers }

procedure DoInitManagers;

  begin
  MaxApplZone;
  MoreMasters;

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

  InitCursor;
  FlushEvents(everyEvent, 0);
  end;
    {of procedure DoInitManagers}

```

```
{ ############################################################################## DoEvents }

procedure DoEvents(theEvent : EventRecord);

  var
  theWindowPtr : WindowPtr;
  userData : longint;

  begin
  case (theEvent.what) of

    mouseDown: begin
      DoMouseDown(theEvent);
      end;

    updateEvt: begin
      theWindowPtr := WindowPtr(theEvent.message);

      BeginUpdate(theWindowPtr);
      if (gMultiMonitorsDrawDemo = true) then
        begin
        userData := longint(theWindowPtr);
        DeviceLoop(theWindowPtr^.visRgn, DeviceLoopDrawingUPP(@DoDeviceLoopDraw),
              userData, 0);
        end;
      EndUpdate(theWindowPtr);
      end;

    osEvt: begin
      DoOSEvent(theEvent);
      HiliteMenu(0);
      end;
    end;
      {of case statement}
  end;
    {of procedure DoInitManagers}

{ ############################################################################## DoMouseDown }

procedure DoMouseDown(theEvent : EventRecord);

  var
  partCode : integer;
  theWindowPtr : WindowPtr;

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

  case (partCode) of

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

    inSysWindow: begin
      SystemClick(theEvent, theWindowPtr);
      end;

    inContent: begin
      if (theWindowPtr <> FrontWindow) then
        SelectWindow(theWindowPtr);
      end;

    inDrag: begin
      DragWindow(theWindowPtr, theEvent.where, qd.screenBits.bounds);
      end;

    inZoomIn, inZoomOut: begin
      if (TrackBox(theWindowPtr, theEvent.where, partCode)) then
        DoZoomWindowMultiMonitors(theWindowPtr, partCode);
      end;
    end;
      {of case statement}
  end;
    {of procedure DoMouseDown}

{ ############################################################################## DoMenuChoice }
```

```
322
323 procedure DoMenuChoice(menuChoice : longint);
324
325   var
326   menuID, menuItem : integer;
327   itemName : string;
328   daDriverRefNum : integer;
329
330   begin
331   menuID := HiWord(menuChoice);
332   menuItem := LoWord(menuChoice);
333
334   if (menuID = 0) then
335     Exit(DoMenuChoice);
336
337   case (menuID) of
338
339     mApple: begin
340       if (menuItem = iAbout) then
341         SysBeep(10)
342       else begin
343         GetMenuItemText(GetMenuHandle(mApple), menuItem, itemName);
344         daDriverRefNum := OpenDeskAcc(itemName);
345         end;
346       end;
347
348     mFile: begin
349       if (menuItem = iQuit) then
350         gDone := true;
351       end;
352
353     mDemonstration: begin
354       gMultiMonitorsDrawDemo := false;
355       case (menuItem) of
356
357         iCommandPeriod: begin
358           DoCommandPeriodAndStatusBar;
359           end;
360
361         iNotification: begin
362           EraseRect(gWindowPtr^.portRect);
363           DoSetUpNotification;
364           end;
365
366         iColourPicker: begin
367           DoColourPicker;
368           end;
369
370         iTrapAvailable: begin
371           EraseRect(gWindowPtr^.portRect);
372           MoveTo(150, 110);
373           if (DoCheckSlotVInstallAvailable) then
374             DrawString('Trap is available')
375           else
376             DrawString('Trap is not available');
377           end;
378
379         iMultiMonitors: begin
380           EraseRect(gWindowPtr^.portRect);
381           gMultiMonitorsDrawDemo := true;
382           InvalRect(gWindowPtr^.portRect);
383           end;
384         end;
385           {of case statement}
386       end;
387     end;
388       {of case statement}
389
390   HiliteMenu(0);
391   end;
392     {of procedure DoMenuChoice}
393
394 { ############################################################################ UnloadSegments }
395
396 procedure UnloadSegments;
397
398   begin
```

```
  UnloadSeg(@DemosSegment);
  end;
    {of procedure UnloadSegments}

end.
  {of unit UMain}

{ ############################################################################################ }

{ ############################################################################################
// UDemos.p
// ############################################################################################ }

unit UDemos;



interface

{ ……… include the following Universal Interfaces }

uses

  Windows, Fonts, Menus, TextEdit, Dialogs, SegLoad, ToolUtils, Devices, GestaltEqu,
  Resources, Sound, Notification, Icons, Processes, ColorPicker, Traps, LowMem;

{ ……… global variables }

var

gNotificationRecord : NMRec;
gStartingTickCount : longint;
gNotificationDemoInvoked : boolean;
gNotificationInQueue : boolean;
gInBackground : boolean;

gWindowPtr : WindowPtr; external;
gColorQuickDraw : boolean; external;
gProcessSerNum : ProcessSerialNumber; external;

{ ……… function and procedure interfaces }

  procedure DemosSegment;
  procedure DoCommandPeriodAndStatusBar;
  procedure DoSetUpNotification;
  procedure DoDeviceLoopDraw(depth, deviceFlags : integer; targetDeviceHdl : GDHandle;
        userData : longint);
  procedure DoNullEvent;
  procedure DoOSEvent(theEvent : EventRecord);
  procedure DoColourPicker;
  function  DoCheckSlotVInstallAvailable : boolean;
  procedure DoZoomWindowMultiMonitors(theWindowPtr : WindowPtr;
        zoomInOrOut : longint);



implementation

uses

{ ……… include the following user-defined units }

  UMain;

{ ……… function and procedure interfaces }

  procedure DoDrawStatusBar(modalDlgPtr : DialogPtr; barRect : Rect;
        statusCurrent, statusMax : integer); forward;
  function  DoCheckForCommandPeriod : boolean; forward;
  procedure DoPrepareNotificationRecord; forward;
  procedure DoDisplayMessageToUser; forward;
  function  DoDecimalToHexadecimal(decimalNumber : UInt16) : string; forward;
  function  TrapAvailable(theTrap : integer) : boolean; forward;
  procedure DoRedoWindowContent(theWindowPtr : WindowPtr); forward;

{ ############################################################################ DemosSegment }
```

```
476  procedure DemosSegment;
477
478    begin
479    end;
480      {of procedure DemosSegment}
481
482  { ############################################################ DoCommandPeriodAndStatusBar }
483
484  procedure DoCommandPeriodAndStatusBar;
485
486    var
487    modalDlgPtr : DialogPtr;
488    barBackColour, barColour : RGBColor;
489    itemType : integer;
490    itemHdl : Handle;
491    itemRect : Rect;
492    a, b, c, temp1, temp2 : integer;
493    soundHdl : Handle;
494    theRect : Rect;
495    statusMax, statusCurrent : integer;
496    finalTicks : UInt32;
497    ignored : OSErr;
498
499    begin
500    EraseRect(gWindowPtr^.portRect);
501
502    modalDlgPtr := GetNewDialog(rDialog, nil, WindowPtr(-1));
503    if (modalDlgPtr = nil) then
504      ExitToShell;
505
506    DrawDialog(modalDlgPtr);
507    SetPort(modalDlgPtr);
508
509    if (gColorQuickDraw) then
510      begin
511      barBackColour.red := $BFFF;
512      barBackColour.green := $BFFF;
513      barBackColour.blue := $FFFF;
514
515      barColour.red := $6FFF;
516      barColour.green := $6FFF;
517      barColour.blue := $6FFF;
518      end;
519
520    GetDialogItem(modalDlgPtr, iUserItem, itemType, itemHdl, itemRect);
521    InsetRect(itemRect, -1, -1);
522    FrameRect(itemRect);
523    InsetRect(itemRect, 1, 1);
524
525    if (gColorQuickDraw) then
526      begin
527      RGBBackColor(barBackColour);
528      FillRect(itemRect, qd.white);
529      RGBForeColor(barColour);
530      end;
531
532    SetPort(gWindowPtr);
533
534    statusMax := 2184;
535    statusCurrent := 0;
536
537    for a := 0 to 7 do
538      begin
539      if (DoCheckForCommandPeriod) then
540        begin
541        soundHdl := GetResource('snd ', rBarkSound);
542        ignored := SndPlay(nil, SndListHandle(soundHdl), false);
543        ReleaseResource(soundHdl);
544        DisposeDialog(modalDlgPtr);
545
546        SetPort(gWindowPtr);
547        EraseRect(gWindowPtr^.portRect);
548        MoveTo(115, 110);
549        ForeColor(blackColor);
550        DrawString('Operation cancelled at user request');
551
552        Exit(DoCommandPeriodAndStatusBar);
```

```
        end;
      for temp1 := 0 to 20 do
        begin
        b := temp1 * 18 + 12;
        for temp2 := 0 to 12 do
          begin
          c := temp2 * 18 + 8;
          SetRect(theRect, b + a, c + a, b + 16 - a, c + 16 - a);
          if (a < 3) then
            ForeColor(redColor)
          else if ((a > 2) and (a < 6)) then
            ForeColor(greenColor)
          else if (a > 5) then
            ForeColor(blueColor);
          FrameRect(theRect);

          DoDrawStatusBar(modalDlgPtr, itemRect, statusCurrent, statusMax);
          statusCurrent := statusCurrent + 1;
          end;
        Delay(2, finalTicks);
        end;
      end;

    DisposeDialog(modalDlgPtr);
    EraseRect(gWindowPtr^.portRect);
    MoveTo(150, 110);
    ForeColor(blackColor);
    DrawString('Operation completed');
    end;
      {of procedure DoCommandPeriodAndStatusBar}

  { ######################################################################## DoDrawStatusBar }

  procedure DoDrawStatusBar(modalDlgPtr : DialogPtr; barRect : Rect;
                      statusCurrent, statusMax : integer);

    var
    barMaxWidth : integer;
    barRequiredWidth : real;

    begin
    SetPort(modalDlgPtr);

    barMaxWidth := barRect.right - barRect.left;
    barRequiredWidth := (statusCurrent / statusMax) * barMaxWidth;
    barRect.right := barRect.left + trunc(barRequiredWidth);

    if (gColorQuickDraw) then
      FillRect(barRect, qd.black)
    else
      FillRect(barRect, qd.gray);

    SetPort(gWindowPtr);
    end;
      {of procedure DoDrawStatusBar}

  { ############################################################## DoCheckForCommandPeriod }

  function  DoCheckForCommandPeriod : boolean;

    var
    foundCommandPeriod : boolean;
    eventQHdrPtr : QHdrPtr;
    eventQElPtr : EvQElPtr;
    keyCode : longint;
    commandKeyDown : longint;

    begin
    foundCommandPeriod := false;

    eventQHdrPtr := GetEvQHdr;
    eventQElPtr := EvQElPtr(eventQHdrPtr^.qHead);

    while ((eventQElPtr <> nil) and not (foundCommandPeriod)) do
      begin
      if (eventQElPtr^.evtQWhat = keyDown) then
        begin
```

```
630         keyCode := BAnd(eventQElPtr^.evtQMessage, charCodeMask);
631
632         commandKeyDown := BAnd(eventQElPtr^.evtQModifiers, cmdKey);
633
634         if (commandKeyDown <> 0) then
635           if (keyCode = ord('.')) then
636             foundCommandPeriod := true;
637         end;
638
639       if not (foundCommandPeriod) then
640         eventQElPtr := EvQElPtr(eventQElPtr^.qLink);
641       end;
642
643     DoCheckForCommandPeriod := foundCommandPeriod;
644     end;
645       {of function DoCheckForCommandPeriod}
646
647   { ##################################################################### DoSetUpNotification }
648
649   procedure DoSetUpNotification;
650
651     begin
652     DoPrepareNotificationRecord;
653     gNotificationDemoInvoked := true;
654
655     gStartingTickCount := TickCount;
656
657     MoveTo(12, 100);
658     DrawString('Please click on the desktop now to make the Finder ');
659     DrawString('the frontmost application.');
660     MoveTo(42, 120);
661     DrawString('(This application will post a notification 10 seconds from now.)');
662     end;
663       {of procedure DoSetUpNotification}
664
665   { ############################################################ DoPrepareNotificationRecord }
666
667   procedure DoPrepareNotificationRecord;
668
669     var
670     iconSuiteHdl : Handle;
671     soundHdl : Handle;
672     stringHdl : StringHandle;
673     ignored : OSErr;
674
675     begin
676     ignored := GetIconSuite(iconSuiteHdl, rIconFamily, svAllSmallData);
677     soundHdl := GetResource('snd ', rBarkSound);
678     stringHdl := GetString(rString);
679
680     gNotificationRecord.qType := nmType;
681     gNotificationRecord.nmMark := 1;
682     gNotificationRecord.nmIcon := iconSuiteHdl;
683     gNotificationRecord.nmSound := soundHdl;
684     gNotificationRecord.nmStr := stringHdl^;
685     gNotificationRecord.nmResp := nil;
686     gNotificationRecord.nmRefCon := 0;
687     end;
688       {of procedure DoPrepareNotificationRecord}
689
690   { ############################################################################## DoNullEvent }
691
692   procedure DoNullEvent;
693
694     var
695     frontProcessSerNum : ProcessSerialNumber;
696     isSameProcess : boolean;
697     ignored : OSErr;
698
699     begin
700     if (gNotificationDemoInvoked) then
701       begin
702       if (TickCount > (gStartingTickCount + 600)) then
703         begin
704         ignored := GetFrontProcess(frontProcessSerNum);
705         ignored := SameProcess(frontProcessSerNum, gProcessSerNum, isSameProcess);
706         if not (isSameProcess) then
```

```
          begin
          ignored := NMInstall(NMRecPtr(@gNotificationRecord));
          gNotificationDemoInvoked := false;
          gNotificationInQueue := true;
          end
        else begin
          DoDisplayMessageToUser;
          gNotificationDemoInvoked := false;
          end;

        EraseRect(gWindowPtr^.portRect);
        end;
      end;
    end;
      {of procedure DoNullEvent}

{ ################################################################################ DoOSEvent }

procedure DoOSEvent(theEvent : EventRecord);

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

    suspendResumeMessage: begin
      gInBackground := BAnd(theEvent.message, resumeFlag) = 0;
      if (not (gInBackground) and gNotificationInQueue) then
        DoDisplayMessageToUser;
      end;

    mouseMovedMessage: begin
      end;
    end;
      {of case statement}
  end;
    {of procedure DoOSEvent}

{ ################################################################## DoDisplayMessageToUser }

procedure DoDisplayMessageToUser;

  var
  ignored : OSErr;

  begin
  if (gNotificationInQueue) then
    begin
    ignored := NMRemove(NMRecPtr(@gNotificationRecord));
    gNotificationInQueue := false;
    end;

  ignored := NoteAlert(rAlert, nil);

  ignored := DisposeIconSuite(gNotificationRecord.nmIcon, false);
  ReleaseResource(gNotificationRecord.nmSound);
  ReleaseResource(Handle(gNotificationRecord.nmStr));
  end;
    {of procedure DoDisplayMessageToUser}

{ ################################################################################ DoColourPicker }

procedure DoColourPicker;

  var
  inColour, outColour, blackColour : RGBColor;
  theRect : Rect;
  where : Point;
  prompt : Str255;
  okButton : boolean;
  theString : string;

  begin
  prompt := 'Choose a rectangle colour:';
  EraseRect(gWindowPtr^.portRect);

  inColour.red := $FFFF;
  inColour.green := $0000;
  inColour.blue := $0000;
```

```
784
785     blackColour.red := $0000;
786     blackColour.green := $0000;
787     blackColour.blue := $0000;
788
789     theRect := gWindowPtr^.portRect;
790     InsetRect(theRect, 50, 50);
791     RGBForeColor(inColour);
792     FillRect(theRect, qd.black);
793
794     where.v := 0;
795     where.h := 0;
796
797     okButton := GetColor(where, prompt, inColour, outColour);
798
799     if (okButton) then
800       begin
801       RGBForeColor(outColour);
802       FillRect(theRect, qd.black);
803       RGBForeColor(blackColour);
804
805       MoveTo(50, 20);
806       DrawString('Red Value: ');
807       theString := DoDecimalToHexadecimal(outColour.red);
808       MoveTo(115, 20);
809       DrawString(theString);
810
811       MoveTo(50, 33);
812       DrawString('Green Value: ');
813       theString := DoDecimalToHexadecimal(outColour.green);
814       MoveTo(115, 33);
815       DrawString(theString);
816
817       MoveTo(50, 46);
818       DrawString('Blue Value: ');
819       theString := DoDecimalToHexadecimal(outColour.blue);
820       MoveTo(115, 46);
821       DrawString(theString);
822       end
823     else begin
824       RGBForeColor(inColour);
825       FillRect(theRect, qd.black);
826       RGBForeColor(blackColour);
827       MoveTo(75, 125);
828       DrawString('Cancel button was clicked.  Rectangle remains red.');
829       end;
830     end;
831       {of procedure DoColourPicker}
832
833   { ################################################################ DoDecimalToHexadecimal }
834
835   function  DoDecimalToHexadecimal(decimalNumber : UInt16) : string;
836
837     var
838     theString : string;
839     hexCharas : string;
840     a : integer;
841
842     begin
843     theString := 'OxXXXX';
844     hexCharas := '0123456789ABCDEF';
845
846     for a := 0 to 3 do
847       begin
848       theString[6 - a] := hexCharas[BAnd(decimalNumber, $F) + 1];
849       decimalNumber := BSR(decimalNumber, 4);
850       end;
851
852     DoDecimalToHexadecimal := theString;
853     end;
854       {of function DoDecimalToHexadecimal}
855
856   { ########################################################## DoCheckSlotVInstallAvailable }
857
858   function  DoCheckSlotVInstallAvailable : boolean;
859
860     begin
```

```
  DoCheckSlotVInstallAvailable := TrapAvailable(_SlotVInstall);
  end;
    {of function DoCheckSlotVInstallAvailable}

{ ############################################################################ TrapAvailable }

function  TrapAvailable(theTrap : integer) : boolean;

  var
  theTrapType : TrapType;
  trapMask : integer;
  numToolboxTraps : integer;

  begin
  trapMask := $0800;

  if (BAnd(theTrap,  trapMask) > 0) then
    theTrapType := ToolTrap
  else
    theTrapType := OSTrap;

  if (theTrapType = ToolTrap) then
    theTrap := BAnd(theTrap, $07FF);

  if (NGetTrapAddress(_InitGraf, ToolTrap) = NGetTrapAddress($AA6E, ToolTrap)) then
    numToolboxTraps := $0200
  else
    numToolboxTraps := $0400;

  if (theTrap >= numToolboxTraps) then
    theTrap := _Unimplemented;

  TrapAvailable :=
     NGetTrapAddress(theTrap, theTrapType) <> NGetTrapAddress(_Unimplemented, ToolTrap);
  end;
    {of function TrapAvailable}

{ ############################################################################ DoDeviceLoopDraw }

procedure DoDeviceLoopDraw(depth, deviceFlags : integer; targetDeviceHdl : GDHandle;
                           userData : longint);

  var
  theWindowPtr : WindowPtr;
  theRect : Rect;
  oldForeColour : RGBColor;
  green: RGBColor;
  red : RGBColor;
  blue : RGBColor;

  begin
  green.red := $6666;
  green.green := $FFFF;
  green.blue := $6666;
  red.red := $FFFF;
  red.green := $6666;
  red.blue := $6666;
  blue.red := $9999;
  blue.green := $9999;
  blue.blue := $FFFF;

  theWindowPtr := WindowPtr(userData);
  EraseRect(theWindowPtr^.portRect);

  case (depth) of

    1, 2: begin
      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);
      end;

    4, 8, 16, 32: begin
      GetForeColor(oldForeColour);
```

```
938         SetRect(theRect, 70, 40, 320, 200);
939         RGBForeColor(green);
940         PaintRect(theRect);
941         InsetRect(theRect, 30, 30);
942         RGBForeColor(red);
943         PaintRect(theRect);
944         InsetRect(theRect, 30, 30);
945         RGBForeColor(blue);
946         PaintRect(theRect);
947         RGBForeColor(oldForeColour);
948         end;
949       end;
950         {of case statement}
951     end;
952       {of procedure DoDeviceLoopDraw}
953 
954   { ############################################################ DoZoomWindowMultiMonitors }
955 
956   procedure DoZoomWindowMultiMonitors(theWindowPtr : WindowPtr; zoomInOrOut : longint);
957 
958     var
959     oldPort : GrafPtr;
960     windRect, intersectRect, zoomRect : Rect;
961     titleBarHeight : integer;
962     winStateDataPtr : WStateDataPtr;
963     deviceHdl, zoomDeviceHdl : GDHandle;
964     intersectArea, greatestArea : longint;
965     sectFlag : boolean;
966 
967     begin
968     GetPort(oldPort);
969     SetPort(theWindowPtr);
970 
971     EraseRect(theWindowPtr^.portRect);
972 
973     windRect := theWindowPtr^.portRect;
974     LocalToGlobal(windRect.topLeft);
975     LocalToGlobal(windRect.botRight);
976     titleBarHeight := windRect.top - WindowPeek(theWindowPtr)^.strucRgn^^.rgnBBox.top - 1;
977 
978     if (zoomInOrOut = inZoomOut) then
979       begin
980       if not (gColorQuickDraw) then
981         begin
982         zoomRect := qd.screenBits.bounds;
983         zoomRect.top := zoomRect.top + LMGetMBarHeight + titleBarHeight;
984         InsetRect(zoomRect, 3, 3);
985 
986         winStateDataPtr := WStateDataPtr(WindowPeek(theWindowPtr)^.dataHandle);
987         winStateDataPtr^.stdState := zoomRect;
988         end
989       else begin
990         windRect.top := windRect.top - titleBarHeight;
991 
992         deviceHdl := LMGetDeviceList;
993         greatestArea := 0;
994 
995         while (deviceHdl <> nil) do
996           begin
997           if (TestDeviceAttribute(deviceHdl, screenDevice)) then
998             if (TestDeviceAttribute(deviceHdl, screenActive)) then
999               begin
1000              sectFlag := SectRect(windRect, deviceHdl^^.gdRect, intersectRect);
1001
1002              intersectArea := longint((intersectRect.right - intersectRect.left) *
1003                                       (intersectRect.bottom - intersectRect.top));
1004
1005              if (intersectArea > greatestArea) then
1006                begin
1007                greatestArea := intersectArea;
1008                zoomDeviceHdl := deviceHdl;
1009                end;
1010
1011              deviceHdl := GetNextDevice(deviceHdl);
1012              end;
1013          end;
1014
```

```
      if (zoomDeviceHdl = LMGetMainDevice) then
        titleBarHeight := titleBarHeight + LMGetMBarHeight;

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

      winStateDataPtr := WStateDataPtr(WindowPeek(theWindowPtr)^.dataHandle);
      winStateDataPtr^.stdState := zoomRect;
      end;
    end;

  ZoomWindow(theWindowPtr, zoomInOrOut, theWindowPtr = FrontWindow);
  DoRedoWindowContent(theWindowPtr);
  SetPort(oldPort);
  end;
    {of procedure DoZoomWindowMultiMonitors}

{ ################################################################## DoRedoWindowContent }

procedure DoRedoWindowContent(theWindowPtr : WindowPtr);

  begin
  { Do scroll bar and TextEdit, etc, adjustments here as appropriate. }

  InvalRect(theWindowPtr^.portRect);
  end;
    {of procedure DoRedoWindowContent}

end.
  {of unit UDemos}

{ ############################################################################################ }
```

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 optimied 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.

Organisation of the source code files

Because the source code is divided into three files (UMain.p and UDemos.p), constants and global variables used by all three files have been placed in UMain.p.

MiscellanyPascal.p

The main program block

The main function initialises the system software managers (Line 99), establishes whether Color QuickDraw is present (Lines 103-107), sets up the menus (Lines 111-121), opens a window and sets the text size (Lines 125-134), gets the process serial number of this process (Line 138), and enters the main event loop (Lines 144-152).

Note that, at Line 151, the application-defined procedure UnloadSegments is called at the bottom of the event loop after the event received by WaitNextEvent has been handled to completion.

UMain.p

The constant declaration block

Lines 183-192 establish constants relating to menu and window resource IDs, and to menu item numbers. Lines 194-201 establish constants relating to resources.

The variable declaration block

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

The procedures 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 264), 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 266-268).

Also note that, in the case of a mouseDown event in the window's zoom box, the application-defined procedure DoZoomWindowMultiMonitors is called if the cursor is still within the zoom box when the mouse button is released (Lines 313-314).

The procedure DoMenuChoice

DoMenuChoice further processes menu choices.

Lines 354-386 respond to choices from the Demonstration menu. Note that, at Lines 373-376, 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 380-382).

The procedure UnloadSegments

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

UDemos.p

The variable declaration block

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.

The procedure DemosSegment

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

The procedure DoCommandPeriodAndStatusBar

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

Line 500 erases the window's content region. Line 502 opens a dialog box using the specified resource. Line 506 draws the contents of the dialog box (two static text items) and Line 507 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 509), Lines 511-517 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 GetDialogItem at Line 520 gets this rectangle. Line 521 then expands this rectangle by one pixel all around before Line 522 draws a rectangle frame. Line 523 returns the rectangle to its original size. If Color QuickDraw is present (Line 525), Lines 527-529 fill the rectangle with the status bar background colour and then set the foreground colour to the moving status bar colour. That done, Line 532 sets the window's graphics port as the current port.

Lines 537-574 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 534 assigns a value representing the number of steps in the task to a variable. Line 535 sets a variable to indicate that none of these steps has yet been completed.

Within the outer loop initiated at Line 537, Line 539 calls an application-defined function which checks whether the user has pressed the Command-period key combination. If this key press has occurred, Lines 541-552 execute. Specifically, Lines 541-542 load a 'snd ' resource and play the sound, Line 543 frees the memory occupied by the 'snd ' resource, Line 544 disposes of the dialog box, Lines 546-550 draw an advisory message in the window, and Line 552 causes DoCommandPeriodAndStatusBar to exit.

Within the inner of the three loops, the rectangles are drawn (Lines 557-571). Each time round this inner loop, an application-defined procedure is called (Line 569) 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 576 disposes of the dialog, and Lines 577-580 draw an advisory message in the window.

The procedure DoDrawStatusBar

DoDrawStatusBar draws the moving status bar.

Line 594 sets the dialog box's graphics port as the current graphics port. Lines 596-598 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 600-603 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 605 sets the window's graphics port as the current graphics port.

The function DoCheckForCommandPeriod

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

Line 621 sets a variable so as to begin by assuming that such an event is not in the queue.

Line 623 gets a pointer to the event queue header. Line 624 gets a pointer to first queue element. Line 626 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 628), Line 630 gets the key code and Line 632 checks whether the Command key was down. If the Command key was down (Line 634), and if the period key was the key pressed (Line 635), the variable foundCommandPeriod set to true (Line 636), causing the loop to exit. Otherwise, the loop calls up the next queue entry for examination (Lines 639-640).

Line 643 returns the result of the search.

The procedure DoSetUpNotification

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

Line 652 calls an application defined function which fills in the relevant fields of a notification record. That done, Line 653 assigns true to a global variable which records that the Notification item has been chosen by the user.

Line 655 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 655. Lines 657-661 simply draw some advisory text in the window.

The procedure DoPrepareNotificationRecord

DoPrepareNotificationRecord fills in the relevant fields of the notification record.

First, however, Line 676 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 677 loads the specified 'snd ' resource and gets its handle. Line 678 loads the specified 'STR ' resource and gets its handle.

Line 680 specifies the type of operating system queue. Line 681 specifies that the mark is to appear next to the application's name in the Application menu. Lines 682-684 assign the icon suite, sound and string handles previously obtained. Line 685 specifies that no response procedure is required to be executed when the notification is posted.

The procedure 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 700), DoNullEvent simply returns immediately.

If, however, that item has just been chosen (Line 700), and if 10 seconds have elapsed since that choice was made (Line 702), the following occurs:

- Lines 704-705 determine whether the current foreground process is Miscellany. If it is not, the notification request is installed in the notification queue (Line 708) and a global variable is set to indicate that a request has been placed in the queue by Miscellany (Line 710). Also, Line 709 resets the gNotificationDemoInvoked variable to false so as to ensure that Lines 702-717 only execute once after the Notification item is chosen.

- If, however, the current foreground process is Miscellany (Line 712), an application-defined procedure is called to present the required message to the user, via an alert box, in the normal way (Line 713). Once again gNotificationDemoInvoked is reset to false so as to ensure that Lines 702-717 only execute once after the Notification item is chosen.

The procedure 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 732), 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 733).

The procedure DoDisplayMessageToUser

DoDisplayMessageToUser is called by DoOSEvent and DoNullEvent in the circumstances previously described.

If a Miscellany notification request is in the queue (Line 751), Lines 753-754 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 757 and the notification's icon suite, sound and string resources are released/disposed of (Lines 759-761).

The procedure DoColourPicker

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

Line 779 erases the window's content region. Lines 781-783 assign red to the RGBColor variable to be specified as the inColor parameter of the GetColor call at Line 797. Lines 785-787 assign black to another RGB colour variable.

Lines 789-792 draw a filled rectangle in the window in the inColor colour (red). Lines 794-795 assign 0 to the fields of the Point variable used as the first parameter in the GetColor call at Line 797. ((0,0) will cause the Color Picker to be centred on the main screen.)

Line 797 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 799), Lines 801-821 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 807, 813, and 819 call an application-defined function to convert the decimal (unsigned 16-bit integer) values in the fields of the RGBColor variable outColor to hexadecimal.

If the user clicks the Cancel button (Line 823), 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.)

The function DoDecimalToHexadecimal

DoDecimalToHexadecimal converts an unsigned 16-bit integer to a hexadecimal string.

The function 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.

The function 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 ($A86E) is a good choice in this regard. If you use _InitGraf in the first call, the second call would use $AA6E (that is, $A86E 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 $A000 to $A7FF and Toolbox traps are numbered in the range $A800 to $AFFF. 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 $A. 68000 instructions are 16 bits long and the $A takes the first four bits, leaving the least significant 12 bits to define the rest of the trap.

Now to the code.

Lines 877-880 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 882-883 change the value in theTrap to the value which would obtain if Toolbox traps were numbered from $A000 rather than from $A800.

Lines 885-888 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 890), then the trap is clearly not present. Accordingly, Line 891 changes the value in theTrap to _Unimplemented, in which case Line 894 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 894 is still required. In this case, Line 894 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 872 and Lines 875-891 (which check for the expanded trap table) may be regarded as obsolete code and may thus be deleted.

The procedure 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 922 typecasts the long value received in the userData parameter to a WindowPtr. Line 923 erases the port rectangle of the specified window.

Line 925 branches 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.

The procedure DoZoomWindowMultiMonitors

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

Lines 968-969 save and set the current graphics port. Line 971 erases the window's port rectangle prior to the zoom so as to avoid flicker. Lines 973-976 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 979-1026 execute only if (Line 978) 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 980 determines if multiple monitors have to be catered for. If not, Lines 982-987 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 989):

- Line 990 establishes a rectangle equal to the window's port rectangle, plus the window's title bar, in global coordinates. Line 992 gets a handle to the first gDevice record in the device list and Line 993 sets the variable greatestArea to 0. The while loop entered at Line 995 then walks the device list. For each active video device 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.
    - The area of the intersection rectangle is calculated and stored in the variable intersectArea (Line 1002).
    - 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 1008).
- Line 1011 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 1015), the height of the menu bar is added to the value in the variable which holds the window's title bar height.
- Lines 1018-1021 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 1023-1024 then assign this rectangle to the stdState field of the window's state data record.

Line 1028 calls ZoomWindow to zoom the window in the appropriate direction, following which an application-defined procedure is called (Line 1029) to redraw the window contents as appropriate. Finally, the saved graphics port is restored (Line 1030).

The procedure DoRedoWindowContent

DoRedoWindowContent is called by DoZoomWindowMultiMonitors to redraw the content region of a newly-zoomed window. Line 1041 invalidates the window's port rectangle, forcing an update event.