

25

MISCELLANY

Demonstration Program: Miscellany

Notification From Applications in the Background


The Need for the Notification Manager

Applications running in the background sometimes need to communicate information to the user, and need to be certain that the user has actually received that information. The Notification Manager provides for this requirement.



Elements of a Notification

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

- An  mark appears against the name of the target application in the Mac OS 8/9 Application menu.

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

- The Mac OS 8/9 Application menu begins alternating between the target application's icon and the foreground application's icon, or the Apple menu title begins alternating between the target application's icon and the Apple icon.

The location of the icon alternation in the Mac OS 8/9 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 Mac OS 8/9 Application menu, the Application menu title alternates; otherwise the Apple menu title alternates.

Note that several applications might post notifications, so there might be a series of alternating icons.

- On Mac OS 8/9, the Sound Manager plays a sound.

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

- On Mac OS 8.6, a modal alert appears, and the user dismisses it (by clicking on the Cancel button). On Mac OS 9.x a floating window appears, allowing the user to continue working in any running application without first dismissing the notification window. On Mac OS X, a system movable modal alert, which remains in front of all other windows, appears.

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

- A response function, if specified, executes.

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

Notifications in Action

Overview

The Notification Manager is automatically initialised at system startup.

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

When appropriate (that is, in the response function or when your application returns to the foreground), you can remove the notification request from the notification queue.

Creating a Notification Request

The Notification Structure

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

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

Field Descriptions:

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

<code>qType</code>	The type of operating system queue. Set to <code>nmType</code> (8).
<code>nmMark</code>	Specifies whether to place a mark next to the name of the application in the Mac OS 8/9 Application menu. 0 means no mark appears. 1 means the mark appears. Applications should ordinarily set this field to 1.
<code>nmIcon</code>	A handle to an icon family containing a small colour icon that is to alternate periodically in the menu bar. If this field is set to <code>NULL</code> , no icon appears. The handle must be non-purgeable.
<code>nmSound</code>	A handle to a sound resource. If this field is set to <code>NULL</code> , no sound is played. If this field is set to -1, the system alert sound is played. The handle must be non-purgeable.
<code>nmStr</code>	Pointer to a string that appears in the alert/floating window/system movable modal alert. If this field is set to <code>NULL</code> , no alert/ floating window/system movable modal alert appears. Your application should not dispose of this storage until it removes the notification request.

`nmResp` Universal procedure pointer to a response function. If this field is set to `NULL`, no response function executes when the notification is posted. If this field is set to `-1`, a pre-defined function removes the notification request when it has completed. However, on Mac OS 8/9, 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.

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

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

You can use response functions to remove notification requests from the notification queue, free any memory¹, or set a global variable in your application to record that the notification was posted.

`nmRefCon` 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:

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

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

Removing a Notification Request

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

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

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

On Mac OS 9.x and Mac OS X the user does not have to dismiss the notification before being able to activate the application. For this reason, when your application is running on Mac OS 9.x or Mac OS X, may wish to have it explicitly cancel the notification using `NMRemove` when the application becomes active.

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

Progress Bars

Operations within an application which tie up the machine for relatively brief periods of time should be accompanied by a cursor shape change to the watch cursor or perhaps to an animated cursor (Mac OS 8/9) or by an invocation of the wait cursor (Mac OS X). On the other hand, lengthy operations should be accompanied by the display of a progress indicator.

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

¹ 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.

As stated at Chapter 2, your application should allow the user to cancel a lengthy operation using the Command-period key combination.

Scanning for Command-Period Key-Down Events

The function `CheckEventQueueForUserCancel` may be used to scan the event queue for Command-period key-down events, and will return true if a Command-period event is found. If true is returned, the lengthy operation should be terminated and the dialog displaying the progress indicator should be closed.

Soliciting a Colour Choice From the User — The Color Picker

The Color Picker Utilities provide your application with:

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

Preamble - Colour Models

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

RGB Model

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

CYMK Model

The CYMK model is closely associated with printing, that is, putting colour on a white page. In this model, the three primary colours (cyan, yellow, and magenta²) are said to be *subtractive* because, the more of each colour you add, the closer the resulting colour is to black. (The inclusion of black in the model accounts for the fact that the colours of printer's inks may vary slightly from true cyan, yellow, and magenta, meaning that a true black may not be achievable with just a CYM model.)

HLS and HSV Models

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

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

² Cyan, magenta, and yellow are the complements of red, green, and blue.

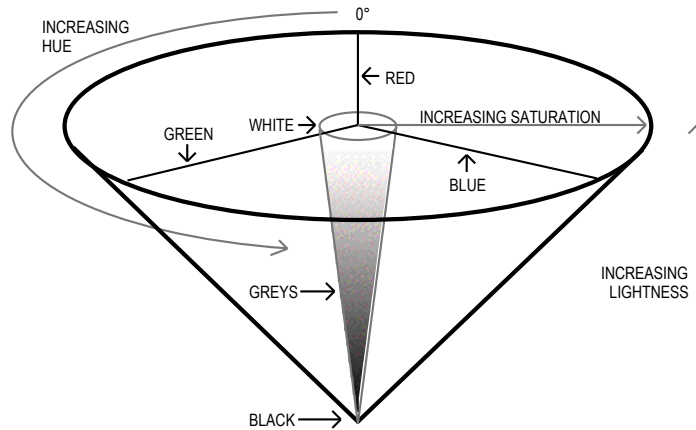


FIG 1 - HSL/HSV COLOUR CONE

The Color Picker

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

Using the Color Picker RGB Mode

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

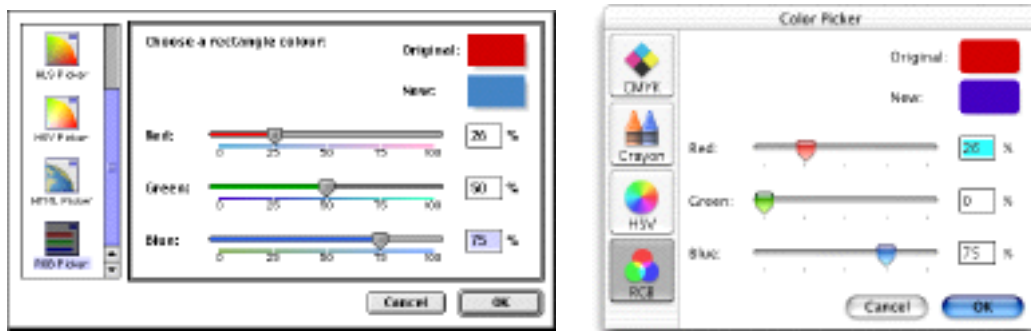


FIG 2 - COLOR PICKER DIALOG IN RGB MODE

Using the Color Picker in HSV Mode

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

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



FIG 3 - COLOR PICKER DIALOG IN HLS MODE

Invoking the Color Picker

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

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

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

where Dialog's upper-left corner. (0,0) causes the dialog to be positioned centrally on the main screen.

prompt A prompt string, which is displayed in the upper left corner of the main pane in the dialog.

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.

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 against **New:.**)

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

Coping With Multiple Monitors

Overview

In a multi-monitor system, the user may 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. In a multi-monitor environment, therefore, it is possible for each screen to be set to a different pixel depth.

In more technical terms, the user's settings set the pixel depth of a particular **video device**. A brief review of the subject of video devices is therefore appropriate at this point.

Video Devices Revisited

As stated at Chapter 11:

- 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, QuickDraw stores information about video devices in `GDevice` structures, the system creates and initialises a `GDevice`

structure for each video device found during start-up all structures are linked together in a list called the **device list**, and the global variable `DeviceList` holds a handle to the first structure in the list.

- At any given time, one, and only one, graphics device is the **current device**³, that is, the one in which the drawing is taking place. A handle to the current device's `GDevice` structure is placed in the global variable `TheGDevice`.

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

Requirements of the Application

Image Optimisation

To draw a particular graphic, your application may have to call different drawing functions for that graphic depending on the characteristics of the video device intersecting your window's drawing region, the aim being to optimise the appearance of the image regardless of whether it is being displayed on, say, a grayscale device or a colour device. Recall from Chapter 11 that when `QuickDraw` displays a colour on a grayscale screen, it computes the luminance, or intensity of light, of the desired colour and uses that value to determine the appropriate gray value to draw. It is thus possible that, for example, two overlapping objects drawn in two quite different colours on a colour screen may appear in the same shade of gray on a grayscale screen. In order for the user to differentiate between these two objects on a grayscale screen, you would need to provide an alternative drawing function which draws the two objects in different shades of gray on grayscale screens.

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

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

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

`DeviceLoop` calls this function for each dissimilar video device it finds. If it encounters similar devices (that is, devices having the same pixel depth, colour table seeds, etc.) it will make only one call to `myDrawingFunction`, pointing to the first such device encountered.

Other Requirements — Classic Event Model Applications

Other requirements, for Classic event model applications only, are as follows:

- ***Window Zooming.*** 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, the window should be zoomed to the standard state on the screen which contained the largest area of the window before the zoom box was clicked. The function `ZoomWindowIdeal`, which was introduced with Mac OS 8.5, zooms windows in accordance with this requirement.
- ***Window Dragging.*** In Carbon, if `NULL` is passed in `DragWindow`'s `boundsRect` parameter, the bounding rectangle limiting the area in which the window can be dragged is set to the desktop region, which, in a multi-monitors environment, includes all screen real estate less the menu bar.

³ The current device is sometimes referred to as the **active device**.

- **Window Sizing.** In a multi-monitor environment, if you pass a constraining rectangle in `ResizeWindow's` `sizeConstraints` parameter, that rectangle should be based on the bounding rectangle of the desktop region. You should call `GetGrayRgn` to get a handle to the desktop region and then call `GetRegionBounds` to get that region's bounding rectangle.

Constraining a Window to One Screen

In a multi-monitors environment, a call to the function `ConstrainWindowToScreen` enables you to constrain a window's movement and resizing so that it is contained entirely on a single screen.

Help Tags

Help tags are the equivalent, on Mac OS X, of Help balloons on Mac OS 8/9, and appear when the cursor hovers over a user interface element. Carbon also supports Help tags on Mac OS 8/9; however, you may consider that, compared with Help balloons, their "look" is somewhat at odds with the Platinum appearance.

Human Interface Guidelines

Guidelines for Help tags are at <http://developer.apple.com/techpubs/macosx/Carbon/pdf/UsingHelpTags.pdf>.

Creating Help Tags

You create a Help tag for, say, a control by filling in the fields of an `HMHelpContentRec` structure and its associated `HMHelpContent` structure and passing the control reference and the address of the `HMHelpContentRec` structure in a call to the function `HMSetControlHelpContent`.

Data Types

The an `HMHelpContentRec` and `HMHelpContent` structures are as follows:

```

struct HMHelpContentRec
{
    SInt32          version;
    Rect            absHotRect;
    HMTagDisplaySide tagSide;
    HMHelpContent  content[2];
};
typedef struct HMHelpContentRec HMHelpContentRec;
typedef HMHelpContentRec *HMHelpContentPtr;

struct HMHelpContent
{
    HMContentType  contentType;
    union
    {
        CFStringRef  tagCFString;    // A CFStringRef reference.
        Str255       tagString;      // A Pascal string.
        HMStringResType tagStringRes; // A 'STR#' resource ID and index.
        TEHandle     tagTEHandle;    // A TextEdit handle. (Mac OS 8/9 only.)
        SInt16       tagTextRes;     // A 'TEXT'/'styl' resource ID. (Mac OS 8/9 only.)
        SInt16       tagStrRes;      // A 'STR ' resource ID
    }
    u;
};
typedef struct HMHelpContent HMHelpContent;

```

The `HMStringResType` structure is used to specify the resource ID and index when Help tag content is sourced from a 'STR#' resource:

```

struct HMStringResType
{
    short hmmResID; // Resource ID of 'STR#' resource.
    short hmmIndex; // 'STR#' resource index.
};

```



```
typedef struct HMStringResType HMStringResType;
```

Note that the content field of the `HMHelpContentRec` structure is a two-element array. The second element allows you to provide an expanded version of the Help tag message when the user presses the Command key.

Constants

Typical constants relevant to the `tagSide` field of the `HMHelpContentRec` structure are as follows:

<i>Constant</i>	<i>Value</i>	<i>Description</i>
<code>kHMDefaultSide</code>	0	System default location.
<code>kHMOutsideTopScriptAligned</code>	1	Above, aligned with left or right depending on system script.
<code>kHMOutsideLeftCenterAligned</code>	2	To the left, centered vertically.
<code>kHMOutsideRightCenterAligned</code>	4	To the right, centered vertically.
<code>kHMOutsideTopLeftAligned</code>	5	Above, aligned with left.
<code>kHMOutsideTopRightAligned</code>	6	Above, aligned with right.
<code>kHMOutsideLeftTopAligned</code>	7	To the left, aligned with top.
<code>kHMOutsideLeftBottomAligned</code>	8	To the left, aligned with bottom.
<code>kHMOutsideBottomLeftAligned</code>	9	Below, aligned with left.
<code>kHMOutsideBottomRightAligned</code>	10	Below, aligned with right.

Constants relevant to the `content` field of the `HMHelpContentRec` structure are as follows:

<i>Constant</i>	<i>Value</i>	<i>Description</i>
<code>kHMMinimumContentIndex</code>	0	First element of content array.
<code>kHMMaximumContentIndex</code>	1	Second element of content array.

Constants relevant to the `contentType` field of the `HMHelpContent` structure are as follows:

<i>Constant</i>	<i>Value</i>	<i>Description</i>
<code>kHMNoContent</code>	'none'	No content.
<code>kHMCFStringContent</code>	'cfst'	Content sourced from a <code>CFStringRef</code> reference.
<code>kHMPascalStrContent</code>	'pstr'	Content sourced from a Pascal string.
<code>kHMStringResContent</code>	'str#'	Content sourced from a 'STR#' resource.
<code>kHMTEHandleContent</code>	'txth'	Content sourced from a <code>TextEdit</code> handle. (Mac OS 8/9 only.)
<code>kHMTextResContent</code>	'text'	Content sourced from 'TEXT'/'styl' resources. (Mac OS 8/9 only.)
<code>kHMStrResContent</code>	'str '	Content sourced from a 'STR ' resource.

Help Tags for Windows

You create a Help tag for a window in the same way as you do for a control, except that:

- You call the function `HMSetWindowHelpContent` instead of `HMSetControlHelpContent`.
- You assign the window's port rectangle, converted to global coordinates, to the `absHotRect` field of the `HMHelpContentRec` structure.
- You must call `HMSetControlHelpContent` again whenever the window size or position changes to ensure that the hot rectangle coordinates are updated.

Help Tags for Menus

You create Help tags for menu titles and items in the same way as you do for a control, except that you call the function `HMSetMenuItemHelpContent` instead of `HMSetControlHelpContent`.

Note

At the time of writing, menu title and menu item Help tags were not supported by Carbon or CarbonLib. However, future support was planned.

Setting the Delay Before Tag Display

You can set the delay, in milliseconds, before a tag opens by calling the function `HMSetTagDelay`.

Enabling and Disabling Help Tags

You can enable and disable help tags using the function `HMSetHelpTagsDisplayed`, and you can determine whether Help tags are currently enabled using the function `HMAreHelpTagsDisplayed`.

Ensuring Compatibility with the Operating Environment

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

Getting Operating Environment Information - The Gestalt Function

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

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

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

`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.

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

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

Gestalt Selectors

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

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

<i>Selector</i>	<i>Information Returned</i>
<code>gestaltFPUType</code>	FPU type.
<code>gestaltKeyboardType</code>	Keyboard type.
<code>gestaltPhysicalRAMSize</code>	Physical RAM size.
<code>gestaltQuickdrawVersion</code>	QuickDraw version.
<code>gestaltTextEditVersion</code>	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. `gestaltMachineType` is an example of an informational selector.

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:

<i>Suffix</i>	<i>Returned Value</i>
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 <code>0x0910</code> returned in response to the <code>gestaltSystemVersion</code> selector means that system software version 9.1.0 is present.

Using Gestalt — Examples

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

Example 1

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

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

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

Example 2

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

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

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

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

```

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

gestaltHasSoundInputDevice = 5
31 - 5 = 26

```

Carbon and Available APIs

CarbonLib and the Underlying Mac OS 8/9 System Software

CarbonLib (the implementation of the Carbon API for Mac OS 8/9) does not, in general, implement new APIs on old systems. It is basically a "pass-through" library that re-exports what is available on the underlying system software.

Thus, if your application calls a function introduced with, say, Mac OS 9.0, it will compile correctly because that function is, by definition, part of the Carbon API (all new function introduced from Mac OS 8.5 onwards are supported by Carbon); however, if the application is run on Mac OS 8.6, the call to that function will fail.

There are a few exceptions to this rule, for example, the menu, control, and window property APIs.

Interpreting Universal Headers Comments

When you run a CFM application on Mac OS X, CFM follows the same rules for runtime linkages as it does on Mac OS 8/9. Since the application links against a library named CarbonLib, there needs to be a CFM library named CarbonLib somewhere on Mac OS X.

That library is `/System/Library/CFM Support/CarbonLib`. It exports all of the APIs as does CarbonLib on Mac OS 8/9, plus some APIs that are only implemented on Mac OS X. (Those exports, incidentally, are not the real implementations but simply glue which transfers control from the CFM library to the Mach-O frameworks, where the APIs are really implemented. (Frameworks are like Mac OS 8/9 shared libraries.))

In the Universal Headers, a comment like:

```
CarbonLib: in CarbonLib 1.1 and later
```

means that the API is implemented in CarbonLib for Mac OS 8/9 and exported from the CarbonLib library on Mac OS X. This comment:

```
CarbonLib: in CarbonLib on Mac OS X
```

means that the API is not implemented in CarbonLib for Mac OS 8/9 but is still exported from the CarbonLib library on Mac OS X. The real determinant of whether an API is available in Carbon.framework is this comment:

```
Mac OS X: in version 10.0 or later
```

If the comment in CarbonLib on Mac OS X applies to a function called in your application, you must add the stub library CarbonFrameworkLib to the CodeWarrior project.

T-Vector Tests

In Carbon, the recommended method for checking the availability of a routine is to check its T-Vector (transition vector) directly, as in the following example

```

if((UInt32) CreateStandardSheet == (UInt32) kUnresolvedCFragSymbolAddress)
    // CreateStandardSheet is not available. Do this.
else
    // Do this.

```

Main Notification Manager Data Types and Functions

Data Types

Notification Structure

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

Functions

Add Notification Request to the Notification Queue

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

Remove Notification Request from the Notification Queue

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

Relevant Process Manager Data Types and Functions

Data Types

Process Serial Number

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

Functions

Getting Process Serial Numbers

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

Comparing Two Process Serial Numbers

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

Relevant Event Manager Function

Check For Command-Period

```
Boolean  CheckEventQueueForUserCancel(void);
```

Relevant Color Picker Utilities Function

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

Relevant QuickDraw Function

Drawing Across Multiple Video Devices

```
void DeviceLoop(RgnHandle drawingRgn,DeviceLoopDrawingUP drawingProc,long userData,
                DeviceLoopFlags flags);
```

Relevant Window Manager Function

```
OSStatus ConstrainWindowToScreen(WindowRef inWindowRef,WindowRegionCode inRegionCode,
                                  WindowConstrainOptions inOptions,const Rect *inScreenRect,
                                  Rect *outStructure);
```

Main Help Package Constants, Data Types, and Functions

Constants

Content Type

```
kHMNoContent           = FOUR_CHAR_CODE('none')
kHMCFStringContent     = FOUR_CHAR_CODE('cfst')
kHMPascalStrContent    = FOUR_CHAR_CODE('pstr')
kHMStringResContent    = FOUR_CHAR_CODE('str#')
kHMTEHandleContent     = FOUR_CHAR_CODE('txth')
kHMTextResContent      = FOUR_CHAR_CODE('text')
kHMStrResContent       = FOUR_CHAR_CODE('str ')
```

Tag Display Side

```
kHMDefaultSide        = 0
kHMOutsideTopScriptAligned = 1
kHMOutsideLeftCenterAligned = 2
kHMOutsideBottomScriptAligned = 3
kHMOutsideRightCenterAligned = 4
kHMOutsideTopLeftAligned = 5
kHMOutsideTopRightAligned = 6
kHMOutsideLeftTopAligned = 7
kHMOutsideLeftBottomAligned = 8
kHMOutsideBottomLeftAligned = 9
kHMOutsideBottomRightAligned = 10
kHMOutsideRightTopAligned = 11
kHMOutsideRightBottomAligned = 12
kHMOutsideTopCenterAligned = 13
kHMOutsideBottomCenterAligned = 14
kHMInsideRightCenterAligned = 15
kHMInsideLeftCenterAligned = 16
kHMInsideBottomCenterAligned = 17
kHMInsideTopCenterAligned = 18
kHMInsideTopLeftCorner = 19
kHMInsideTopRightCorner = 20
kHMInsideBottomLeftCorner = 21
kHMInsideBottomRightCorner = 22
kHMAbsoluteCenterAligned = 23
```

For HMHelpContentRec.content

```
kHMMinimumContentIndex = 0
kHMMaximumContentIndex = 1
```

Data Types

```
typedef UInt32 HMContentType;
typedef SInt16 HMTagDisplaySide;
```

HelpContent

```
struct HMHelpContent
{
    HMContentType    contentType;
    union
    {
        {
            CFStringRef    tagCFString;
            Str255         tagString;
            HMStringResType tagStringRes;
            TEHandle       tagTEHandle;
            SInt16         tagTextRes;
            SInt16         tagStrRes;
        }
        u;
    };
};
typedef struct HMHelpContent HMHelpContent;
```

HMHelpContentRec

```
struct HMHelpContentRec
{
    SInt32            version;
    Rect              absHotRect;
    HMTagDisplaySide tagSide;
    HMHelpContent    content[2];
};
typedef struct HMHelpContentRec HMHelpContentRec;
typedef HMHelpContentRec *HMHelpContentPtr;
```

HMStringResType

```
struct HMStringResType
{
    short    hmmResID;
    short    hmmIndex;
};
typedef struct HMStringResType HMStringResType;
```

Functions

Installing and Retrieving Content

```
OSStatus  HMSetControlHelpContent(ControlRef inControl, const HMHelpContentRec *inContent);
OSStatus  HMGetControlHelpContent(ControlRef inControl, HMHelpContentRec *outContent);
OSStatus  HMSetWindowHelpContent(WindowRef inWindow, const HMHelpContentRec *inContent);
OSStatus  HMGetWindowHelpContent(WindowRef inWindow, HMHelpContentRec *outContent);
OSStatus  HMSetMenuItemHelpContent(MenuRef inMenu, MenuItemIndex inItem,
    const HMHelpContentRec *inContent);
OSStatus  HMGetMenuItemHelpContent(MenuRef inMenu, MenuItemIndex inItem,
    HMHelpContentRec *outContent);
```

Enabling and Disabling Help Tags

```
Boolean  HMAreHelpTagsDisplayed(void);
OSStatus HMSetHelpTagsDisplayed(Boolean inDisplayTags);
```

Setting and Getting Tag Delay

```
OSStatus HMSetTagDelay(Duration inDelay);
OSStatus HMGetTagDelay(Duration *outDelay);
```

Displaying Tags

```
OSStatus HMDisplayTag(const HMHelpContentRec *inContent);
```

Relevant Gestalt Manager Function

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

Demonstration Program Miscellany Listing

```
// *****
// Miscellany.h CARBON EVENT MODEL
// *****
//
// This program demonstrates:
//
// • The use of the Notification Manager to allow an application running in the background to
//   to communicate with the foreground application.
//
// • The use of the determinate progress bar control to show progress during a time-
//   consuming operation, together with scanning the event queue for Command-period key-down
//   events for the purpose of terminating the lengthy operation before it concludes of its
//   own accord.
//
// • The use of the Color Picker to solicit a choice of colour from the user.
//
// • Image drawing optimisation in a multi-monitors environment.
//
// • Help tags for controls and windows with minimum and maximum content.
//
// The program utilises the following resources:
//
// • A 'plst' resource.
//
// • An 'MBAR' resource, and 'MENU' resources for Apple, File, Edit and Demonstration menus
//   (preload, non-purgeable).
//
// • A 'DLOG' resource (purgeable), and associated 'DITL', 'dlgx', and 'dftb' resources
//   (purgeable), for a dialog box in which the progress indicator is displayed.
//
// • 'CNTL' resources (purgeable) for the progress indicator dialog.
//
// • 'icn#', 'ics4', and 'ics8' resources (non-purgeable) which contain the application icon
//   shown in the Application menu during the Notification Manager demonstration.
//
// • A 'snd ' resource (non-purgeable) used in the Notification Manager demonstration.
//
// • A 'STR ' resource (non-purgeable) containing the text displayed in the alert box invoked
//   by the Notification Manager.
//
// • 'STR#' resources (purgeable) containing the label and narrative strings for the
//   notification-related alert displayed by Miscellany and the minimum and maximum Help tag
//   content.
//
// • A 'SIZE' resource with the acceptSuspendResumeEvents, canBackground,
//   doesActivateOnFGSwitch, and isHighLevelEventAware flags set.
//
// *****

// ..... includes

#include <Carbon.h>

// ..... defines

#define rMenuBar          128
#define mAppleApplication 128
#define iAbout            1
#define mFile             129
#define iQuit             12
#define mDemonstration    131
#define iNotification     1
#define iProgress         2
#define iColourPicker     3
#define iMultiMonitors    4
#define iHelpTag          5
```



```

#define rWindow          128
#define rDialog          128
#define iProgressIndicator 1
#define rIconFamily      128
#define rBarkSound       8192
#define rString          128
#define rAlertStrings    128
#define indexLabel       1
#define indexNarrative   2
#define rPicture         128
#define topLeft(r)       (((Point *) &(r))[0])
#define botRight(r)      (((Point *) &(r))[1])

// ..... function prototypes

void    main                (void);
void    doPreliminaries    (void);
OSStatus appEventHandler    (EventHandlerCallRef,EventRef,void *);
OSStatus windowEventHandler (EventHandlerCallRef,EventRef,void *);
void    doMenuChoice       (MenuID,MenuItemIndex);

void    doSetUpNotification (void);
void    doPrepareNotificationStructure (void);
void    doIdle              (void);
void    doDisplayMessageToUser (void);

void    doProgressBar       (void);

void    deviceLoopDraw      (SInt16,SInt16,GDHandle,SInt32);

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

void    doHelpTagControl    (void);
void    doHelpTagWindow     (void);

// *****
// Miscellany.c
// *****

#include "Miscellany.h"

// ..... global variables

DeviceLoopDrawingUPP gDeviceLoopDrawUPP;
WindowRef            gWindowRef;
ControlRef           gBevelButtonControlRef;
ProcessSerialNumber  gProcessSerNum;
Boolean              gMultiMonitorsDrawDemo = false;
Boolean              gColourPickerDemo = false;
Boolean              gHelpTagsDemo = false;
RGBColor             gWhiteColour = { 0xFFFF, 0xFFFF, 0xFFFF };
RGBColor             gBlueColour = { 0x6666, 0x6666, 0x9999 };

extern Boolean        gNotificationInQueue;

// ***** main

void main(void)
{
    MenuBarHandle menubarHdl;
    SInt32         response;
    MenuRef        menuRef;
    Rect           contentRect = { 100,100,402,545 };
    Rect           portRect;
    Rect           controlRect = { 65,10,155,100 };
    EventTypeSpec  applicationEvents[] = { { kEventClassApplication, kEventAppActivated },
                                           { kEventClassCommand, kEventProcessCommand } };

```

```

EventTypeSpec windowEvents[] = { { kEventClassWindow, kEventWindowDrawContent },
                                  { kEventClassWindow, kEventWindowGetIdealSize },
                                  { kEventClassWindow, kEventWindowGetMinimumSize },
                                  { kEventClassWindow, kEventWindowBoundsChanged } };

// ..... do preliminaries
doPreliminaries();

// ..... create universal procedure pointer
gDeviceLoopDrawUPP = NewDeviceLoopDrawingUPP((DeviceLoopDrawingProcPtr) deviceLoopDraw);

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

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

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

// ..... install application event handler and timer
InstallApplicationEventHandler(NewEventHandlerUPP((EventHandlerProcPtr) appEventHandler),
                              GetEventTypeCount(applicationEvents),applicationEvents,
                              0,NULL);

InstallEventLoopTimer(GetCurrentEventLoop(),0,1,
                      NewEventLoopTimerUPP((EventLoopTimerProcPtr) doIdle),NULL,NULL);

// ..... open window

CreateNewWindow(kDocumentWindowClass,kWindowStandardHandlerAttribute |
               kWindowStandardDocumentAttributes,&contentRect,&gWindowRef);

ChangeWindowAttributes(gWindowRef,0,kWindowCloseBoxAttribute);
SetWTitle(gWindowRef,"pMiscellany");
RepositionWindow(gWindowRef,NULL,kWindowAlertPositionOnMainScreen);

SetPortWindowPort(gWindowRef);
TextSize(10);

ShowWindow(gWindowRef);
GetWindowPortBounds(gWindowRef,&portRect);
InvalWindowRect(gWindowRef,&portRect);

// ..... install window event handler
InstallWindowEventHandler(gWindowRef,
                         NewEventHandlerUPP((EventHandlerProcPtr) windowEventHandler),
                         GetEventTypeCount(windowEvents),windowEvents,0,NULL);

```

```

// ..... create control and help tags
CreateBevelButtonControl(gWindowRef,&controlRect,CFSTR("Control"),
                        kControlBevelButtonNormalBevel,kControlBehaviorPushbutton,
                        NULL,0,0,0,&gBevelButtonControlRef);
doHelpTagControl();
HideControl(gBevelButtonControlRef);
doHelpTagWindow();
HMSetHelpTagsDisplayed(false);

// ..... get process serial number of this process
GetCurrentProcess(&gProcessSerNum);

// ..... run application event loop
RunApplicationEventLoop();
}

// ***** doPreliminaries
void doPreliminaries(void)
{
    MoreMasterPointers(640);
    InitCursor();
}

// ***** appEventHandler
OSStatus appEventHandler(EventHandlerCallRef eventHandlerCallRef,EventRef eventRef,
                        void * userData)
{
    OSStatus    result = eventNotHandledErr;
    UInt32      eventClass;
    UInt32      eventKind;
    HICommand   hiCommand;
    MenuID      menuID;
    MenuItemIndex menuItem;

    eventClass = GetEventClass(eventRef);
    eventKind  = GetEventKind(eventRef);

    switch(eventClass)
    {
    case kEventClassApplication:
        if(eventKind == kEventAppActivated)
        {
            if(gNotificationInQueue)
                doDisplayMessageToUser();
            result = noErr;
        }
        break;

    case kEventClassCommand:
        if(eventKind == kEventProcessCommand)
        {
            GetEventParameter(eventRef,kEventParamDirectObject,typeHICommand,NULL,
                            sizeof(HICommand),NULL,&hiCommand);
            menuID = GetMenuID(hiCommand.menu.menuRef);
            menuItem = hiCommand.menu.menuItemIndex;
            if((hiCommand.commandID != kHICommandQuit) &&
                (menuID >= mAppleApplication && menuID <= mDemonstration))
            {
                doMenuChoice(menuID,menuItem);
                result = noErr;
            }
        }
        break;
    }
}

```

```

}

return result;
}

// ***** windowEventHandler

OSStatus windowEventHandler(EventHandlerCallRef eventHandlerCallRef,EventRef eventRef,
                           void* userData)
{
    OSStatus result = eventNotHandledErr;
    UInt32 eventClass;
    UInt32 eventKind;
    WindowRef windowRef;
    SInt32 deviceLoopUserData;
    RgnHandle regionHdl;
    Rect portRect, positioningBounds;
    Point idealHeightAndWidth, minimumHeightAndWidth;

    eventClass = GetEventClass(eventRef);
    eventKind = GetEventKind(eventRef);

    switch(eventClass)
    {
    case kEventClassWindow:
        GetEventParameter(eventRef,kEventParamDirectObject,typeWindowRef,NULL,sizeof(windowRef),
                          NULL,&windowRef);
        switch(eventKind)
        {
        case kEventWindowDrawContent:
            if(gMultiMonitorsDrawDemo)
            {
                RGBBackColor(&gWhiteColour);
                deviceLoopUserData = (SInt32) windowRef;
                regionHdl = NewRgn();
                if(regionHdl)
                {
                    GetPortVisibleRegion(GetWindowPort(windowRef),regionHdl);
                    DeviceLoop(regionHdl,gDeviceLoopDrawUPP,deviceLoopUserData,0);
                    DisposeRgn(regionHdl);
                }
            }
            else if(gColourPickerDemo )
            {
                RGBBackColor(&gBlueColour);
                GetWindowPortBounds(windowRef,&portRect);
                EraseRect(&portRect);
                doDrawColourPickerChoice();
            }
            else
            {
                RGBBackColor(&gBlueColour);
                GetWindowPortBounds(windowRef,&portRect);
                EraseRect(&portRect);
                if(gHelpTagsDemo)
                {
                    Draw1Control(gBevelButtonControlRef);
                    RGBForeColor(&gWhiteColour);
                    MoveTo(10,20);
                    DrawString("\pHover the cursor in the window, and over the bevel button, ");
                    DrawString("\puntil the Help tag appears.");
                    MoveTo(10,35);
                    DrawString("\pPress the Command key to invoke the maximum content.");
                    MoveTo(10,50);
                    DrawString("\pDrag the window to a new location.");
                }
            }
        }
        result = noErr;
        break;
}

```

```

    case kEventWindowGetIdealSize:
        GetAvailableWindowPositioningBounds(GetMainDevice(),&positioningBounds);
        idealHeightAndWidth.v = positioningBounds.bottom;
        idealHeightAndWidth.h = positioningBounds.right;
        SetEventParameter(eventRef,kEventParamDimensions,typeQDPoint,
            sizeof(idealHeightAndWidth),&idealHeightAndWidth);
        result = noErr;
        break;

    case kEventWindowGetMinimumSize:
        minimumHeightAndWidth.v = 302;
        minimumHeightAndWidth.h = 445;
        SetEventParameter(eventRef,kEventParamDimensions,typeQDPoint,
            sizeof(minimumHeightAndWidth),&minimumHeightAndWidth);
        result = noErr;
        break;

    case kEventWindowBoundsChanged:
        doHelpTagWindow();
        GetWindowPortBounds(windowRef,&portRect);
        InvalWindowRect(windowRef,&portRect);
        result = noErr;
        break;
    }
    break;
}
return result;
}

// ***** doMenuChoice

void doMenuChoice(MenuID menuID,MenuItemIndex menuItem)
{
    Rect portRect;

    if(menuID == 0)
        return;

    switch(menuID)
    {
    case mAppleApplication:
        if(menuItem == iAbout)
            SysBeep(10);
        break;

    case mDemonstration:
        gMultiMonitorsDrawDemo = gColourPickerDemo = gHelpTagsDemo = false;
        if(HMAreHelpTagsDisplayed)
            HMSetHelpTagsDisplayed(false);
        HideControl(gBevelButtonControlRef);
        GetWindowPortBounds(gWindowRef,&portRect);

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

        case iProgress:
            RGBBackColor(&gBlueColour);
            EraseRect(&portRect);
            doProgressBar();
            break;

        case iColourPicker:

```

```

        gColourPickerDemo = true;
        doColourPicker();
        break;

    case iMultiMonitors:
        gMultiMonitorsDrawDemo = true;
        InvalWindowRect(gWindowRef,&portRect);
        break;

    case iHelpTag:
        gHelpTagsDemo = true;
        InvalWindowRect(gWindowRef,&portRect);
        ShowControl(gBevelButtonControlRef);
        HMSetHelpTagsDisplayed(true);
        break;
    }

    break;
}
}

// *****
// Notification.c
// *****

#include "Miscellany.h"

// ..... global variables

NMRec          gNotificationStructure;
long           gStartingTickCount;
Boolean        gNotificationDemoInvoked;
Boolean        gNotificationInQueue;
extern WindowRef gWindowRef;
extern ProcessSerialNumber gProcessSerNum;
extern RGBColor gWhiteColour;
extern RGBColor gBlueColour;

// ***** doSetUpNotification

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

    gStartingTickCount = TickCount();

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

// ***** doPrepareNotificationStructure

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

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

    gNotificationStructure.qType = nmType;
    gNotificationStructure.nmMark = 1;
}

```

```

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

// ***** doIdle

void doIdle(void)
{
    ProcessSerialNumber frontProcessSerNum;
    Boolean              isSameProcess;
    Rect                 portRect;

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

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

            GetWindowPortBounds(gWindowRef,&portRect);
            EraseRect(&portRect);
        }
    }
}

// ***** doDisplayMessageToUser

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

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

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

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

```

```

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

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

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

// *****
// ProgressIndicator.c
// *****

#include "Miscellany.h"

// ..... global variables

extern WindowRef gWindowRef;
extern RGBColor gWhiteColour;

// ***** doProgressBar

void doProgressBar(void)
{
    DialogRef dialogRef;
    RgnHandle visRegionHdl = NewRgn();
    ControlRef progressBarRef;
    SInt16 statusMax, statusCurrent;
    SInt16 a, b, c;
    Handle soundHdl;
    Rect portRect, theRect;
    RGBColor redColour = { 0xFFFF, 0x0000, 0x0000 };

    if(!(dialogRef = GetNewDialog(rDialog,NULL,(WindowRef) -1)))
        ExitToShell();

    SetPortDialogPort(dialogRef);
    GetPortVisibleRegion(GetWindowPort(GetDialogWindow(dialogRef)),visRegionHdl);
    UpdateControls(GetDialogWindow(dialogRef),visRegionHdl);
    QDFlushPortBuffer(GetDialogPort(dialogRef),NULL);

    SetPortWindowPort(gWindowRef);
    GetWindowPortBounds(gWindowRef,&portRect);

    GetDialogItemAsControl(dialogRef,iProgressIndicator,&progressBarRef);

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

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

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

```



```

        return;
    }

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

    QDFlushPortBuffer(GetWindowPort(gWindowRef),NULL);
    QDFlushPortBuffer(GetDialogPort(dialogRef),NULL);

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

DisposeRgn(visRegionHdl);
DisposeDialog(dialogRef);
EraseRect(&portRect);
MoveTo(10,292);
RGBForeColor(&gWhiteColour);
DrawString("\pOperation completed");
}

// *****
// ColourPicker.c
// *****

#include "Miscellany.h"

// ..... global variables

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

// ***** doColourPicker

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

    GetWindowPortBounds(gWindowRef,&portRect);
    theRect = portRect;

    RGBBackColor(&gBlueColour);
    EraseRect(&theRect);
    InsetRect(&theRect,55,55);
    RGBForeColor(&gInColour);
    PaintRect(&theRect);

    where.v = where.h = 0;

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

    InvalWindowRect(gWindowRef,&portRect);
}

// ***** doDrawColorPickerChoice

void doDrawColourPickerChoice(void)
{
    Rect portRect;

```

```

char *cString;

GetWindowPortBounds(gWindowRef,&portRect);
InsetRect(&portRect,55,55);

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

    RGBForeColor(&gWhiteColour);

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

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

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

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

// ***** doDecimalToHexadecimal

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

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

    return cString;
}

// *****
// MultiMonitor.c
// *****

#include "Miscellany.h"

// ***** deviceLoopDraw

void deviceLoopDraw(SInt16 depth,SInt16 deviceFlags,GDHandle targetDeviceHdl,SInt32 userData)
{
    RGBColor oldForeColor;
    WindowRef windowRef;
    Rect portRect;
    RGBColor greenColour = { 0x0000, 0xAAAA, 0x1111 };

```

```

RGBColor  redColour    = { 0xAAAA, 0x4444, 0x3333 };
RGBColor  blueColour   = { 0x5555, 0x4444, 0xFFFF };
RGBColor  ltGrayColour = { 0xDDDD, 0xDDDD, 0xDDDD };
RGBColor  grayColour   = { 0x9999, 0x9999, 0x9999 };
RGBColor  dkGrayColour = { 0x4444, 0x4444, 0x4444 };

GetForeColor(&oldForeColor);

windowRef = (WindowRef) userData;
GetWindowPortBounds(windowRef,&portRect);
EraseRect(&portRect);

if(((1 << gdDevType) & deviceFlags) != 0)
{
    InsetRect(&portRect,50,50);
    RGBForeColor(&greenColour);
    PaintRect(&portRect);
    InsetRect(&portRect,40,40);
    RGBForeColor(&redColour);
    PaintRect(&portRect);
    InsetRect(&portRect,40,40);
    RGBForeColor(&blueColour);
    PaintRect(&portRect);
}
else
{
    InsetRect(&portRect,50,50);
    RGBForeColor(&ltGrayColour);
    PaintRect(&portRect);
    InsetRect(&portRect,40,40);
    RGBForeColor(&grayColour);
    PaintRect(&portRect);
    InsetRect(&portRect,40,40);
    RGBForeColor(&dkGrayColour);
    PaintRect(&portRect);
}

RGBForeColor(&oldForeColor);
}

// *****
// HelpTag.c
// *****

#include "Miscellany.h"
#include <string.h>

// ..... global variables

extern ControlRef gBevelButtonControlRef;
extern WindowRef  gWindowRef;

// ..... doHelpTagControl

void doHelpTagControl(void)
{
    HMHelpContentRec helpContent;

    memset(&helpContent,0,sizeof(helpContent));
    HMSetTagDelay(50);

    helpContent.version = kMacHelpVersion;
    helpContent.tagSide = kHMOutsideBottomLeftAligned;

    helpContent.content[kHMMinimumContentIndex].contentType = kHMStringResContent;
    helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmResID = 129;
    helpContent.content[kHMMinimumContentIndex].u.tagStringRes.hmmIndex = 1;
    helpContent.content[kHMMaximumContentIndex].contentType = kHMStringResContent;
    helpContent.content[kHMMaximumContentIndex].u.tagStringRes.hmmResID = 129;
}

```

```

helpContent.content[kHMMMaximumContentIndex].u.tagStringRes.hmmIndex = 2;

HMSetControlHelpContent(gBevelButtonControlRef,&helpContent);
}

// ..... doHelpTagWindow

void doHelpTagWindow(void)
{
    Rect          hotRect;
    HMHelpContentRec helpContent;

    memset(&helpContent,0,sizeof(helpContent));
    HMSetTagDelay(500);

    helpContent.version = kMacHelpVersion;
    helpContent.tagSide = kHMOoutsideRightCenterAligned;

    GetWindowPortBounds(gWindowRef,&hotRect);
    LocalToGlobal(&topLeft(hotRect));
    LocalToGlobal(&botRight(hotRect));
    helpContent.absHotRect = hotRect;

    helpContent.content[kHMMMinimumContentIndex].contentType = kHMStringResContent;
    helpContent.content[kHMMMinimumContentIndex].u.tagStringRes.hmmResID = 129;
    helpContent.content[kHMMMinimumContentIndex].u.tagStringRes.hmmIndex = 3;
    helpContent.content[kHMMMaximumContentIndex].contentType = kHMStringResContent;
    helpContent.content[kHMMMaximumContentIndex].u.tagStringRes.hmmResID = 129;
    helpContent.content[kHMMMaximumContentIndex].u.tagStringRes.hmmIndex = 4;

    HMSetWindowHelpContent(gWindowRef,&helpContent);
}

// *****

```

Demonstration Program Miscellany Comments

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

- Choose the Notification item and, observing the instructions in the window, click the desktop immediately to make the Finder the foreground application. A notification will be posted by Miscellany about 10 seconds after the Notification item choice is made. Note that, when about 10 seconds have elapsed, the Notification Manager invokes an alert (Mac OS 8.6), floating window (Mac OS 9.x), or system movable modal alert (Mac OS X) and alternates the Finder and Miscellany icons in the OS 8/9 Application menu title. Observing the instructions in the alert/floating window/system movable modal alert:
 - Dismiss the alert (Mac OS 8.6 only).
 - On Mac OS 8/9, then choose the Miscellany item in the OS 8/9 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.
 - On Mac OS X, click on the application's icon in the Dock.
- Choose the Notification item again and, this time, leave Miscellany in the foreground. Note that only the alert 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 invoked by Miscellany appears.
- Choose the Determinate Progress Indicator item, noting that the progress indicator dialog is automatically disposed of when the (simulated) time-consuming task concludes.
- Choose the Determinate Progress Indicator item again, and this time press the Command-period key combination before the (simulated) time-consuming task concludes. Note that the progress indicator dialog is disposed of when the Command-period key combination is pressed.
- Choose the Colour Picker item and make colour choices using the various available modes. Note that, when the Colour Picker is dismissed by clicking the OK button, the requested RGB colour values for the chosen colour are displayed in hexadecimal, together with a rectangle in that colour, in the Miscellany window.
- Choose the Multiple Monitors Draw item, noting that the drawing of the simple demonstration image is optimised as follows:
 - On a monitor set to display 256 or more colours, the image is drawn in three distinct colours. The luminance of the three colours is identical, meaning that, if these colours are drawn on a grayscale screen, they will all appear in the same shade of gray.
 - On a monitor set to display 256 shades of gray, the image is drawn in three distinct shades of gray.
- Choose the Help Tags item, hover the cursor over the window and, when the Help tag appears, press the Command key to observe the maximum content version of the tag. Repeat this while hovering the cursor over the bevel button control.

Miscellany.c

Global Variables

`gDeviceLoopDrawUPP` will be assigned a universal procedure pointer to the image-optimising drawing function `deviceLoopDraw` called by `DeviceLoop`. `gProcessSerNum` will be assigned the process serial number of the Miscellany application.

main

The call to `NewDeviceLoopDrawingProc` creates a universal procedure pointer to the image-optimising drawing function `deviceLoopDraw`.

A timer is installed and set to fire every one second. When it fires, the function `doIdle` is called.

A bevel button control is created, following which the calls to `doHelpTagControl` and `doHelpTagWindow` create Help tags for the bevel button control and the window. `HMSetHelpTagsDisplayed` is called to disable the tags until the Help Tags item is chosen from the Demonstration menu.

`GetCurrentProcess` gets the process serial number of this process. The timer and the process serial number are used in the notification demonstration.

appEventHandler

When the `kEventAppActivated` event type is received, if the global variable `gNotificationInQueue` is set to true, `doDisplayMessageToUser` is called. This is part of the notification demonstration.

windowEventHandler

When the `kEventWindowDrawContent` event type is received, if the Multiple Monitors Draw item in the Demonstration menu has been chosen (`gMultiMonitorsDrawDemo` is true), a call is made to `DeviceLoop` and the universal procedure pointer to the application-defined (callback) drawing function `deviceLoopDraw` is passed as the second parameter.

doMenuChoice

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

Notification.c

doSetUpNotification

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

The first line calls `doPrepareNotificationStructure`, which fills in the relevant fields of a notification structure. The next line assigns true to a global variable which records that the Notification item has been chosen by the user.

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

doPrepareNotificationStructure

`doPrepareNotificationStructure` fills in the relevant fields of the notification structure.

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

The first line of the main block specifies the type of operating system queue. The next line specifies that the mark is to appear next to the application's name in the Mac OS 8/9 Application menu. The next three lines assign the icon suite (for Mac OS 8/9), sound (for Mac OS 8/9) and string handles previously obtained. The next line specifies that no response function is required to be executed when the notification is posted.

doIdle

`doIdle` is called when the installed timer fires.

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

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

- The calls to `GetFrontProcess` and `SameProcess` determine whether the current foreground process is Miscellany. If it is not, the notification request is installed in the notification queue by `NMInstall` and the global variable `gNotificationInQueue` is set to indicate that a request has been placed in the queue by Miscellany. (This latter causes `doDisplayMessageToUser` to be called when the `kEventAppActivated` event is received. `doDisplayMessageToUser` removes the notification request from the queue and has Miscellany convey the required message to the user.) Also, `gNotificationDemoInvoked` is set to false so as to ensure that the main if block only executes once after the Notification item is chosen.
- If, however, the current foreground process is Miscellany, the function `doDisplayMessageToUser` is called to present the required message to the user in the normal way. Once again

`gNotificationDemoInvoked` is reset to false so as to ensure that the main if block only executes once after the Notification item is chosen.

doDisplayMessageToUser

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

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

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

ProgressBar.c

doProgressBar

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

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

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

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

Each time round around the inner for loop, a progress indicator control's value is incremented.

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

ColourPicker.c

doColourPicker

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

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

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

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

doDrawColourPickerChoice

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

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

doDecimalToHexadecimal

`doDecimalToHexadecimal` converts a UInt16 value to a hexadecimal string.

MultiMonitor.c

deviceLoopDraw

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

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

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

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

HelpTag.c

doHelpTagControl and doHelpTagWindow

`doHelpTagControl` and `doHelpTagWindow` create Help tags for the bevel button control and the window.

The call to `memset` clears the specified block of memory. The call to `HMSetTagDelay` sets the delay, in milliseconds, before the tag opens.

For the bevel button, the `tagSide` field of the `HMHelpContentRec` structure is assigned a value which will cause the control's tag to be displayed below the control with its left side aligned with the left side of the button. For the window, the `tagSide` field is assigned a value which will cause the control's tag to be displayed on the window's right, centered vertically.

The main block sets the content type and retrieves and assigns the minimum and maximum content strings from a 'STR#' resource. The calls to `HMSetControlHelpContent` and `HMSetWindowHelpContent` install the Help tags on the control and window.