

6

THE APPEARANCE MANAGER *Includes Demonstration Program Appearance*

Introduction

The Appearance Manager, which was first introduced with Mac OS 8.0, had implications for the Menu Manager, the Window Manager, the Control Manager, and the Dialog Manager. The relatively minor implications in respect of the Menu Manager and Window Manager were incorporated into Chapter 3 — Menus and Chapter 4 — Windows. The most profound impact of the Appearance Manager, however, has been in the area of user interface objects known as **controls**, which are addressed at Chapter 7 — Introduction to Controls and at Chapter 14 — More on Controls. Accordingly, as a preparation for what is to come, this chapter now formally introduces the Appearance Manager, a component of the system software which represents the most significant improvement in the Macintosh user experience since the introduction of System 7.

The Era of Themes

The Appearance Manager ushers in the era of **themes**. Essentially, a theme is an interface "look" that spans all elements of the user interface and ties them together with a certain graphic design. Themes are data-driven, that is, all the data that describes the theme interface is contained in a **theme file**. This makes it easy to change themes on the fly.

Fig 1 shows three windows, each in a separate theme. If one of these themes has been selected by the user, all elements of the user interface (menus, windows, controls, etc.) will appear in that theme.

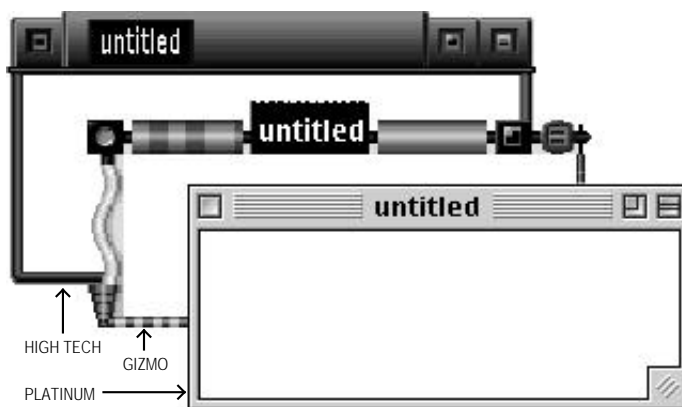


FIG 1 - WINDOWS IN THREE THEMES

At the time of writing, Platinum was the only available theme; however, switchable themes are very much part of the future of the Mac OS. Appearance was the first step towards that future.

The Appearance Manager

The Appearance Manager, whose influence is evident to a greater or lesser extent in all Macintosh C demonstration programs and in many chapters of this book:

- Coordinates the look of the Mac OS human interface into a single theme.
- Introduced new human interface elements to the Mac OS environment.
- Allows for the adaptation of pre–Appearance Manager human interface elements, both standard and custom, to the new, coordinated appearance and behaviour of elements of the user interface.
- Provides the underlying support for themes and theme switching.

New Definition Functions

To provide a system-wide coordination of appearance and behaviour, the Appearance Manager introduced new **Appearance-compliant** definition functions to replace the old pre-Appearance definition functions for menu bars, menus, windows, and controls. In addition, many new Appearance-compliant control definition functions for new types of controls (slider controls, focus rings, group boxes, etc) were introduced to obviate the necessity for developers to provide their own. (The use of Appearance-compliant definition functions is essential to achieving a unified look within any given theme.)

Mapping of Pre-Appearance Definition Functions

Another way in which the Appearance Manager achieved a unified look and behaviour was by **mapping** the following standard pre-Appearance definition functions to their Appearance-compliant equivalents:

- The menu bar definition function (MBDF) with resource ID 0.
- The menu definition function (MDEF) with resource ID 0.
- The window definition function (WDEF) with resource ID 0. (Document windows).
- The window definition function (WDEF) with resource ID 124. (Utility windows).
- The control definition function (CDEF) with resource ID 0. (Buttons, checkboxes, and radio buttons).
- The control definition function (CDEF) with resource ID 1. (Scroll bars).
- The control definition function (CDEF) with resource ID 63. (Pop-up menus).

Mapping is implemented by a set of **mapper definition functions** which are located, along with all the Appearance-compliant definition functions, in the Appearance extension. The mappers have the same resource ID as the pre-Appearance definition functions to which they relate.

Mapping occurs either on a system-wide basis (if the user has not selected system-wide Appearance off in the Appearance control panel), or on an individual-application basis if you call the function `RegisterAppearanceClient` from within your application. Of course, no mapping occurs if your application specifies the new Appearance-compliant definition functions, which means that those definition functions will be **called directly**. The left side of Figure 2 shows the ways by which it is determined how, and whether, mapping will occur for a standard definition function, in this case for

the pre-Appearance WDEF for document windows (resource ID 0). The right side of Fig 2 shows the Appearance-compliant control definition function being called directly.

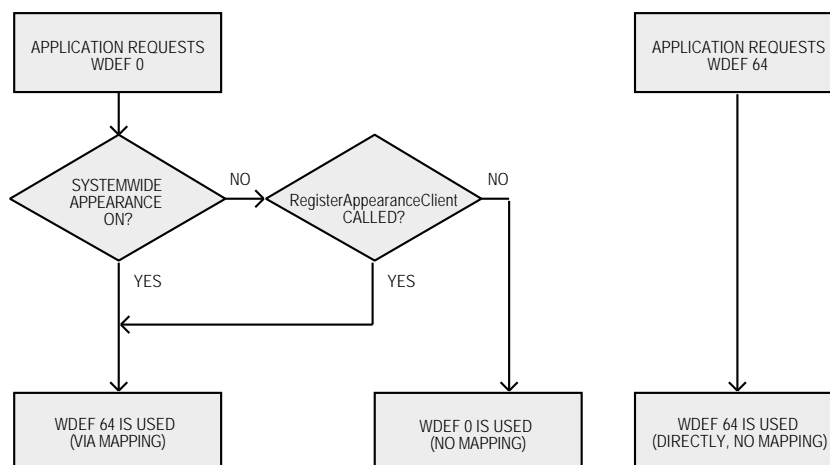


FIG 2 - MAPPING A STANDARD PRE-APPEARANCE DEFINITION FUNCTION TO ITS APPEARANCE-COMPLIANT EQUIVALENT, AND CALLING AN APPEARANCE-COMPLIANT DEFINITION FUNCTION DIRECTLY

Disadvantages of Calling a Definition Function Via a Mapper

When an Appearance-compliant definition function is called via the mappers, the associated object may have a slightly different look and behaviour than is the case when they are called directly. For example:

- Since a standard pre-Appearance WDEF cannot specify the inclusion of a horizontal zoom box, when a pre-Appearance WDEF is mapped to an Appearance-compliant WDEF, the resulting window will not have a horizontal zoom box.
- It is never necessary to call `DrawGrowIcon` to have the grow icon drawn in a window's size box when an Appearance-compliant WDEF is called directly. However, when it is called via the mapper, `DrawGrowIcon` must be called once for the grow icon to be drawn.
- When the Appearance-compliant WDEF for modal and movable modal dialog boxes is called via the mapper, the three-pixel-wide space between the content region and the structure region created by the pre-Appearance WDEF will remain. This space created certain difficulties in the past. When the Appearance-compliant WDEF is called directly, the three-pixel-wide space is banished.

For these reasons, and to eliminate the overhead involved in calling an Appearance-compliant definition function through the mappers, it is best to call the Appearance-compliant definition function directly.

Mapping of Custom Definition Functions

Custom definition functions cannot be automatically mapped to Appearance-compliant equivalents. However, the Appearance Manager does provide ways to coordinate custom user interface elements with themes. For example, using `DrawThemeListBoxFrame` creates a theme-compliant frame for a custom list box.

The RegisterAppearanceClient Function

The following describes the `RegisterAppearanceClient` function, together with its sister function `UnregisterAppearanceClient`.

Function	Description
<code>RegisterAppearanceClient</code>	This function must be called at the beginning of your application, prior to initialising or drawing any onscreen elements or invoking any definition functions, such as the menu bar. Applications that call this function will continue to have a platinum look when system-wide appearance is off. The function automatically maps standard pre-Appearance definition functions to their Appearance-compliant equivalents, whether or not the user has turned on system-wide Appearance. Although they will not make use of mapping, applications that specify Appearance-compliant definition function IDs directly should also call this function.
<code>UnregisterAppearanceClient</code>	This function makes your application non-Appearance-compliant and turns off the mapping of standard pre-Appearance definition functions to their Appearance-compliant equivalents. The function is automatically called for you when your application terminates. Accordingly this function does not normally need to be called.

Checking For the Presence of Appearance Manager

Before calling any functions dependent upon the Appearance Manager's presence, your application should check for the presence of the Appearance Manager.

The `Gestalt` function may be used to acquire a wide range of information about the operating environment, and may be used to determine:

- Whether the Appearance Manager and its functions are present.
- Whether the Macintosh is currently in compatibility mode, that is, whether the user has switched system-wide Appearance off in the Appearance control panel.
- The version of the Appearance Manager that is present.

You pass a **selector** in the `selector` parameter of `Gestalt` and the function returns a **response** in the `response` parameter. The following code fragment shows how to, in sequence, check that the Appearance Manager and its functions are present, determine whether system-wide Appearance is on, and check that Appearance Version 1.0.1 or later is present.

```

OSErr    osError;
SInt32   response;
Boolean  appearanceFunctionsAvail = false;
Boolean  version101Present        = false;
Boolean  inCompatibilityMode      = false;

// Call Gestalt with the gestaltAppearanceAttr selector
osError = Gestalt(gestaltAppearanceAttr, &response);

// If Gestalt returns no error and the bit in response represented by the constant
// gestaltAppearanceExists is set, proceed, otherwise exit with an error message.

if(osError == noErr && (BitTst(&response, 31 - gestaltAppearanceExists)))
{
    // Appearance and its functions are available. Set a flag if required.

    appearanceFunctionsAvail = true;

    // If the bit in response represented by the constant gestaltAppearanceCompatMode
    // is set, system-wide Appearance is off.

    if(BitTst(&response, 31 - gestaltAppearanceCompatMode))
        inCompatibilityMode = true;

    // Call Gestalt again with the gestaltAppearanceVersion selector.

    Gestalt(gestaltAppearanceVersion, &response);

```

```

// If the low order word in response is 0x0101, Version 1.0.1 or later is present.
if(response & 0x00000101 == 0x00000101)
    version101Present = true;
}
else
{
    // Nil-Appearance error alert presented here, then exit.
}

```

Note in the above that the function `BitTst` is used to determine whether a 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 in the above example 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

gestaltAppearanceExists = 0
31 - 0 = 0
gestaltAppearanceCompatMode = 1
31 - 1 = 1

```

Compatibility Mode

The above code fragment makes reference to **compatibility mode**. When system-wide appearance is selected off in the Appearance control panel, the machine is said to be in compatibility mode, meaning that there will be no mapping of the pre-Appearance definition functions.

Colours, Patterns, and the Current Theme

The Appearance Manager provides drawing primitives, and the means to set the colours and patterns, needed to draw consistently with the current theme. Using these drawing primitives, colours, and patterns makes it easier to create visual entities and custom definition functions that are consistent with the current theme.

Drawing Appearance Primitives

Appearance provides functions for drawing **Appearance primitives**. As will become apparent at Chapter 7 — Introduction to Controls and at Chapter 14 — More on Controls, most of these primitives relate to certain controls. The control definition functions for these controls call these primitives when drawing the relevant control. For example, the control definition function for a primary group box calls the primitive `DrawThemePrimaryGroup` to draw the two-pixel wide Appearance-compliant visual representation of that control.

Your application might use these primitives to, for example:

- Draw an Appearance-compliant image of a placard, window header, edit text field frame, etc., when you don't want to use a control.
- Assist you in making a custom list box Appearance-compliant by using `DrawThemeListBoxFrame` to draw the frame and `DrawThemeFocusRect` to draw the focus ring.

The following functions draw Appearance primitives. Those which do not relate to controls are `DrawThemeModelDialogFrame`, `DrawThemeFocusRect`, and `DrawThemeFocusRegion`.

Function	Description
<code>DrawThemePrimaryGroup</code>	Draws a primary group box frame consistent with the current theme.
<code>DrawThemeSecondaryGroup</code>	Draws a secondary group box frame consistent with the current theme. Allows you to nest a secondary group box frame within the primary group box frame.
<code>DrawThemeSeparator</code>	Draws a separator line consistent with the current theme. The orientation of the rectangle determines where the separator line is drawn. If the rectangle is wider than it is tall, the separator line is horizontal; otherwise it is vertical.
<code>DrawThemeWindowHeader</code>	Draws a window header consistent with the current theme. This function draws a window header such as that used by the Finder. The window header is drawn inside the rectangle that is passed.
<code>DrawThemeWindowListViewHeader</code>	Draws a window list view header, such as that used by the Finder, consistent with the current theme. The header is drawn inside the rectangle that is passed in. A window list view header is drawn without a line on its bottom edge, so that bevel buttons can be placed against it without overlapping.
<code>DrawThemePlacard</code>	Draws a placard consistent with the current theme.
<code>DrawThemeModellessDialogFrame</code>	Draws a modeless dialog box frame, like the one drawn by the Dialog Manager, consistent with the current theme. This function may be used to make a custom modeless dialog box Appearance-compliant. The purpose of the modeless dialog frame is to assist in making modeless dialog windows visually distinguishable from normal document windows.
<code>DrawThemeEditTextFrame</code>	Draws an edit text field frame consistent with the current theme. The rectangle passed in should be the same as the one passed in the function <code>DrawThemeFocusRect</code> (see below) so you get the correct focus look for your edit text field control. You should not use these frames for items other than edit text fields.
<code>DrawThemeListBoxFrame</code>	Draws a list box frame consistent with the current theme. The rectangle passed in should be the same as the one passed into the function <code>DrawThemeFocusRect</code> (see below) so that you get the correct focus look for your list box.
<code>DrawThemeGenericWell</code>	Draws an image well frame consistent with the current theme. Image well frames are for use with custom image well controls. You can specify that the center of the well be filled with white. Ensure that Appearance 1.0.1 or later is present before calling this function.
<code>DrawThemeFocusRect</code>	Draws or erases a focus ring around a specified rectangle consistent with the current theme. To achieve the right look, you should first call <code>DrawThemeEditTextFrame</code> OR <code>DrawThemeListBoxFrame</code> and then call <code>DrawThemeFocusRect</code> , passing the same rectangle in the <code>inRect</code> parameter. If you use <code>DrawThemeFocusRect</code> to erase the focus ring around an edit text field frame or list box frame, you will have to redraw the edit text field frame or list box frame because there is typically an overlap.
<code>DrawThemeFocusRegion</code>	Draws or erases an Appearance-compliant focus ring around a specified region. Ensure that Appearance 1.0.1 or later is present before calling this function.

Draw State Constants

The following constants are passed in the `inState` parameter of all of these functions (except `DrawThemeFocusRect` and `DrawThemeFocusRegion`) to specify whether the primitive should be drawn in the active or deactivated mode.¹

Constant	Value	Description
<code>kThemeStateDisabled</code>	0	Draw the primitive in the inactive mode.
<code>kThemeStateActive</code>	1	Draw the primitive in the active mode.

¹ `DrawThemeFocusRect` and `DrawThemeFocusRegion` either draw or erase the focus rectangle depending on whether true or false is passed in the `inHasFocus` parameter.

Another draw state constant (`kThemeStatePressed`) is available to draw certain primitives in the pressed mode; however, the primitives listed above can only be drawn in the active and inactive modes.

Fig 3 and Fig 4 show examples, in platinum Appearance, of images drawn in both the active and inactive modes using the Appearance primitives.

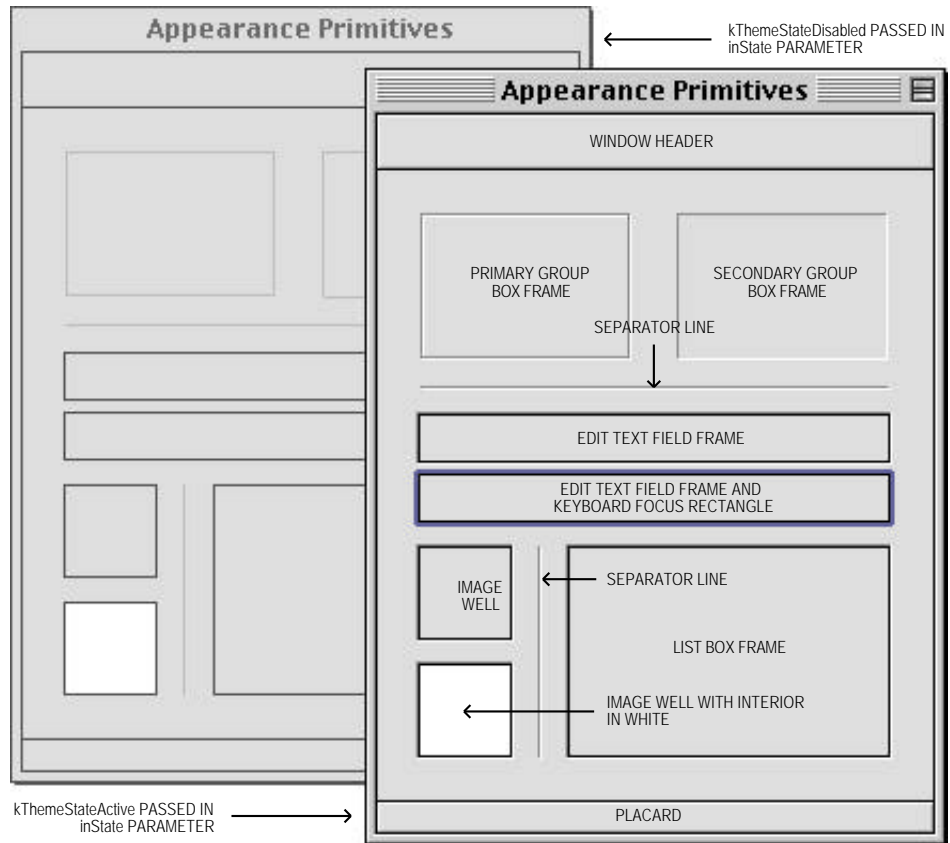


FIG 3 - IMAGES DRAWN WITH OTHER APPEARANCE DRAWING PRIMITIVES



FIG 4 - MODELESS DIALOG FRAME DRAWN WITH APPEARANCE DRAWING PRIMITIVE

Drawing in Colours and Patterns Consistent With the Current Theme

The following functions are those used to draw using colours and patterns consistent with the current theme.

Function	Description
SetThemeWindowBackground	Sets the Appearance-compliant colour or pattern that the window background will be repainted to when <code>PaintOne</code> is called. This function sets the colour or pattern to which the Window Manager will erase the window background. The constant in the <code>inBrush</code> parameter can represent a colour or pattern, depending on the current theme. See also Appearance-Compliant Brush Type Constants, below.
SetThemeBackground	Sets an element's background colour or pattern to comply with the current theme. This function should be called each time you wish to draw an element in a specified brush constant using Appearance Manager draw functions. The constant in the <code>inBrush</code> parameter can represent a colour or pattern, depending on the current theme. See also Appearance-Compliant Brush Type Constants, below.
SetThemePen	Sets an element's pen pattern or colour to comply with the current theme. This function should be called each time you wish to draw an element in a specified brush constant using Appearance Manager draw functions. See also Appearance-Compliant Brush Type Constants, below.
SetThemeTextColor	Sets an element's foreground colour for drawing text to comply with the current theme. This function is typically used inside a <code>DevilceLoop</code> drawing procedure to set the foreground colour for drawing text in order to coordinate with the current theme. See also Appearance-Compliant Text Colour Constants, below.
IsThemeInColor	Checks to see whether the current theme would draw in colour in the given environment. This function is useful when you are drawing elements to match the current theme and need to determine whether or not the theme would be drawn in colour or black and white. If the function returns <code>true</code> , you can draw in colour; if it returns <code>false</code> , you should draw in black and white.
GetThemeAccentColors	Gets a copy of the accent colours for the platinum theme. This function returns a copy of an element's accent colours, but only for the platinum theme. If <code>GetThemeAccentColors</code> is called when another theme is current, it returns an error.

Appearance-Compliant Brush Type Constants

The following constants, which are of type `ThemeBrush`, may be passed in the `inBrush` parameter of calls to `SetThemeWindowBackground`, `SetThemeBackground`, and `SetThemePen` to specify Appearance-compliant colours or patterns for user interface elements.

Constant	Description
<code>kThemeActiveDialogBackgroundBrush</code>	An active dialog box's background colour or pattern.
<code>kThemeInactiveDialogBackgroundBrush</code>	An inactive dialog box's background colour or pattern.
<code>kThemeActiveAlertBackgroundBrush</code>	An active alert box's background colour or pattern.
<code>kThemeInactiveAlertBackgroundBrush</code>	An inactive alert box's background colour or pattern.
<code>kThemeActiveModellessDialogBackgroundBrush</code>	An active modeless dialog box's background colour or pattern.
<code>kThemeInactiveModellessDialogBackgroundBrush</code>	An inactive modeless dialog box's background colour or pattern.
<code>kThemeActiveUtilityWindowBackgroundBrush</code>	An active utility window's background colour or pattern.
<code>kThemeInactiveUtilityWindowBackgroundBrush</code>	An inactive utility window's background colour or pattern.
<code>kThemeListViewSortColumnBackgroundBrush</code>	The background colour or pattern of the column upon which a list view is sorted.
<code>kThemeListViewBackgroundBrush</code>	The background colour or pattern of a list view column that is not being sorted upon.
<code>kThemeIconLabelBackgroundBrush</code>	An icon label's colour or pattern.
<code>kThemeListViewSeparatorBrush</code>	A list view separator's colour or pattern.
<code>kThemeChasingArrowsBrush</code>	Asynchronous arrows' colour or pattern.
<code>kThemeDragHighlightBrush</code>	The background colour or pattern of an element responding to a drag and drop, indicating that the element is a valid recipient.
<code>kThemeDocumentWindowBackgroundBrush</code>	A document window's background colour or pattern.
<code>kThemeFinderWindowBackgroundBrush</code>	A Finder window's background colour or pattern. Generally, you should not use this constant unless you are trying to create a window that matches the Finder window.

These constants can represent either a straight colour or a **pattern**, depending on the current theme. Patterns are explained at Chapter 11 — QuickDraw Preliminaries. Chapter 12 — Drawing With QuickDraw addresses certain measures which need to be taken consequential to the fact that both colours *and* patterns can be set by the Appearance functions `SetThemeWindowBackground`, `SetThemeBackground`, and `SetThemePen`.

Appearance-Compliant Text Colour Constants

Constants of type `ThemeTextColor` may be passed in the `inColor` parameter of the function `SetThemeTextColor` calls to specify Appearance-compliant text colours for user interface elements in their active, inactive, and highlighted states. Some of these constants are as follows:

Constant	Description
<code>kThemeActiveWindowHeaderTextColor</code>	Text colour for active window header.
<code>kThemeInactiveWindowHeaderTextColor</code>	Text colour for inactive window header.
<code>kThemeActivePlacardTextColor</code>	Text colour for active placard.
<code>kThemeInactivePlacardTextColor</code>	Text colour for inactive placard.
<code>kThemePressedPlacardTextColor</code>	Text colour for highlighted placard.
<code>kThemeListViewTextColor</code>	Text colour for list view.

Window Metrics and Appearance

Prior to the introduction of the Appearance Manager, the matter of positioning windows (which is based on the top-left corner of the content region) was quite straightforward given that window borders were always one pixel wide, the title bar was always 19 pixels high, and the menu bar was always a fixed height. However, as shown at Fig 1, such assumptions cannot be made under Appearance. When switchable themes are implemented, the widths of window borders, the height of window title bars, and the height of the menu bar, will vary by theme. Where appropriate, your application must deal with this situation by getting the structure and content rectangles for the window, calculating the width of the window border and title bar, and adjusting the position and size of the window accordingly.

Things That Do Not Change

In the era of themes, where the look of the entire interface can change at any time, it may seem that your application has to be ready for anything. In some respects, that is true. (For example, you need to lay out your user interface elements with enough room to look good in all themes.) However, there are some things you will always be able to rely on. These are as follows:

- Edit text field frames, group box frames and list box frames will always be a maximum of two pixels thick.
- The menu bar height can vary, but will never be more than 24 pixels high.
- Metrics of controls will be the same across themes, though borders, which are drawn outside a control's rectangle, can change for default rings on push button controls² and keyboard focus frames. That said, default rings and keyboard focus frames will never be outset more than three pixels from their associated rectangles.
- Progress indicator³ borders will never be outset more than two pixels from their associated rectangles.

As previously stated, window structure metrics do change and your application needs to be able to cope with a theme switch resulting in a possible change in window frame width, title bar height, and menu bar height.

² See Chapter 7 — Introduction to Controls.

³ See Chapter 14 — More On Controls.

Advising Your Application That a Theme Switch Has Occurred

Your application is advised that a theme switch has occurred via an Apple event⁴. Your application can then take the appropriate action, such as adjusting its window positions, to accommodate the switch.

Appearance-Compliant Applications

Making a New Application Appearance-Compliant

The following lists the actions required to make a new application Appearance-compliant:

- Call `RegisterAppearanceClient` early in your application code, before you draw the menu bar.
- Use the system-supplied Appearance-compliant menu and window definition functions.
- Use Appearance Manager functions and constants to get any colours and patterns you need to draw consistently with the current theme, and to draw Appearance-compliant visual entities such as window headers when you don't want to use a control of that type.
- As will be explained at Chapter 7 — Introduction to Controls and at Chapter 14 — More on Controls, use the system-supplied Appearance-compliant control definition functions.
- As will be explained at Chapter 8 — Dialogs and Alerts:
 - Use the new 'dlgx' and 'alrx' resources to supplement your 'DLOG' and 'ALRT' resources.
 - Enable embedding and Appearance-compliant backgrounds.

In addition, and because the Appearance Manager introduces a movable modal alert and simplifies the handling of movable modal alert and dialog boxes, make all your alerts and dialogs movable. Also use the `StandardAlert` routine, introduced with the Appearance Manager, to create your alerts whenever possible.

Making Old Applications Appearance-Compliant

Ultimately, the task of making an old non-Appearance compliant application fully Appearance-compliant will involve all of the steps listed at Making a New Application Appearance-Compliant, above.

The task may be phased, however, by taking one simple initial step. That step is to simply insert a call to `RegisterAppearanceClient` early in your code. This will cause the mappers to invoke the new definition functions.

When converting an application, be sure to select system-wide Appearance off in the Appearance control panel. This puts your system back into the old System 7 look for applications that have not adopted Appearance, which makes it easy for you to tell where you have implemented the new look and where you still have work to do. (If you are running with system-wide Appearance selected on, you will not be able to distinguish the changes you've made from those performed automatically by the system.)

⁴ See Chapter 10 — Required Apple Events.

Main Constants, Data Types, and Functions

In the following, those items appearing on a gray background are available only with Appearance Version 1.0.1 and later.

Constants

Checking For Appearance, Appearance Functions, and Version

```
gestaltAppearanceAttr          = FOUR_CHAR_CODE('appr')
gestaltAppearanceExists       = 0
gestaltAppearanceCompatMode   = 1
gestaltAppearanceVersion      = FOUR_CHAR_CODE('apvr')
```

Appearance-Compliant Brush Type Constants

```
kThemeActiveDialogBackgroundBrush = 1
kThemeInactiveDialogBackgroundBrush = 2
kThemeActiveAlertBackgroundBrush = 3
kThemeInactiveAlertBackgroundBrush = 4
kThemeActiveModelEssDialogBackgroundBrush = 5
kThemeInactiveModelEssDialogBackgroundBrush = 6
kThemeActiveUtilityWindowBackgroundBrush = 7
kThemeInactiveUtilityWindowBackgroundBrush = 8
kThemeListViewSortColumnBackgroundBrush = 9
kThemeListViewBackgroundBrush = 10
kThemeIconLabelBackgroundBrush = 11
kThemeListViewSeparatorBrush = 12
kThemeChasingArrowsBrush = 13
kThemeDragHighlightBrush = 14
kThemeDocumentWindowBackgroundBrush = 15
kThemeFinderWindowBackgroundBrush = 16
```

Appearance-Compliant Text Colour Constants

```
kThemeActiveWindowHeaderTextColour = 7
kThemeInactiveWindowHeaderTextColour = 8
kThemeActivePlacardTextColour = 9
kThemeInactivePlacardTextColour = 10
kThemePressedPlacardTextColour = 11
```

Appearance-Compliant Draw State Constants (For Primitives)

```
kThemeStateDisabled = 0
kThemeStateActive = 1
kThemeStatePressed = 2
```

Appearance Manager Apple Events

```
kAppearanceEventClass = FOUR_CHAR_CODE('appr')
kAETHEME_SWITCH = FOUR_CHAR_CODE('thme')
```

Data Types

```
typedef UInt32 ThemeDrawState;
typedef UInt16 ThemeBrush;
typedef UInt16 ThemeTextColour;
```

Functions

Initialising the Appearance Manager

```
OSStatus RegisterAppearanceClient (void);
OSStatus UnregisterAppearanceClient (void);
```

Drawing Appearance Primitives

```
OSStatus DrawThemeWindowHeader(const Rect *inRect, ThemeDrawState inState)
OSStatus DrawThemeWindowListViewHeader(const Rect *inRect, ThemeDrawState inState)
OSStatus DrawThemePlacard(const Rect *inRect, ThemeDrawState inState)
```

```

OSStatus DrawThemeEditTextFrame(const Rect *inRect, ThemeDrawState inState)
OSStatus DrawThemeListBoxFrame(const Rect *inRect, ThemeDrawState inState)
OSStatus DrawThemeFocusRect(const Rect *inRect, Boolean inHasFocus)
OSStatus DrawThemePrimaryGroup(const Rect *inRect, ThemeDrawState inState)
OSStatus DrawThemeSecondaryGroup(const Rect *inRect, ThemeDrawState inState)
OSStatus DrawThemeSeparator(const Rect *inRect, ThemeDrawState inState)
OSStatus DrawThemeModellessDialogFrame(const Rect *inRect, ThemeDrawState inState)
OSStatus DrawThemeGenericWell(const Rect *inRect, ThemeDrawState inState, Boolean inFillCenter)
OSStatus DrawThemeFocusRegion(RgnHandle inRegion, Boolean inHasFocus)

```

Drawing in Colours and Patterns Consistent With the Current Theme

```

OSStatus SetThemeWindowBackground(WindowPtr inWindow, ThemeBrush inBrush, Boolean inUpdate)
OSStatus SetThemeBackground(ThemeBrush inBrush, SInt16 inDepth, Boolean inIsColorDevice)
OSStatus SetThemePen(ThemeBrush inBrush, SInt16 inDepth, Boolean inIsColorDevice)
OSStatus SetThemeTextColor(ThemeTextColor inColor, SInt16 inDepth, Boolean inIsColorDevice)
OSStatus GetThemeAccentColors(CTabHandle *outColors); // Works with Platinum theme only
OSStatus IsThemeInColor(SInt16 inDepth, Boolean inIsColorDevice)

```

Demonstration Program

```

// *****
// Appearance.c
// *****
//
// This program opens two kWindowDocumentProc windows containing:
//
// • In the first window, an Appearance-compliant list view.
//
// • In the second window, various images drawn with Appearance primitives and window
//   header text drawn in the correct theme colour.
//
// When the windows are opened, they are immediately moved to a position which accounts
// for the differing window widths and menu bar heights under different themes.
//
// Two of the images in the second window are edit text field frames and one is a list
// box frame. At any one time, one of these will have a keyboard focus frame drawn
// around it. Clicking in one of the other frames will move the keyboard focus frame
// to that frame.
//
// The program is terminated by the choosing the Quit item in the File menu.
//
// The program utilises the following resources:
//
// • An 'MBAR' resource, and 'MENU' resources for Apple, File, Edit, and Demonstration
//   menus, and the pop-up menus (preload, non-purgeable).
//
// • Two 'WIND' resources (purgeable) (initially not visible).
//
// • 'hrct' and 'hwin' resources (both purgeable), which provide help balloons
//   describing the contents of the windows.
//
// • A 'SIZE' resource with the acceptSuspendResumeEvents, doesActivateOnFGSwitch,
//   and is32BitCompatible flags set.
//
// *****
// ..... includes
//
#include <Appearance.h>
#include <Devices.h>
#include <Fonts.h>
#include <Gestalt.h>
#include <Menus.h>
#include <Processes.h>
#include <Sound.h>
#include <ToolUtils.h>
#include <LowMem.h>
// ..... defines
#define rMenubar 128
#define rNewWindow1 128
#define rNewWindow2 129

```

```

#define mApple      128
#define iAbout      1
#define mFile       129
#define iQuit       11

#define MAXLONG     0x7FFFFFFF
#define topLeft(r)  (((Point *) &(r))[0])

// ..... global variables

Boolean  gInCompatibilityMode      = false;
Boolean  gVersion101Present        = false;
Boolean  gDone;
Boolean  gInBackground;
WindowPtr gWindowPtr1, gWindowPtr2;
SInt16   gPixelDepth;
Boolean  gIsColourDevice          = false;
Rect     gCurrentRect;

// ..... function prototypes

void  main                (void);
void  doInitManagers      (void);
void  doEvents            (EventRecord *);
void  doUpdate            (EventRecord *);
void  doActivate          (EventRecord *);
void  doActivateWindow   (WindowPtr, Boolean);
void  doOSEvent           (EventRecord *);
void  doMoveWindowForThemes (WindowPtr, Boolean);
void  doDrawThemePrimitives (ThemeDrawState);
void  doDrawThemeCompliantText (WindowPtr, ThemeDrawState);
void  doDrawListView      (WindowPtr);
void  doChangeKeyboardFocus (Point);
void  doGetDepthAndDevice (void);

// ***** main

void  main(void)
{
    OSErr      osError;
    SInt32     response;
    Handle     menubarHdl;
    MenuHandle menuHdl;
    EventRecord EventStructure;

    // ..... initialise managers

    doInitManagers();

    // ..... check for Appearance and functions, compatibility mode, Appearance version

    osError = Gestalt(gestaltAppearanceAttr, &response);

    if(osError == noErr && (BitTst(&response, 31 - gestaltAppearanceExists)))
    {
        if(BitTst(&response, 31 - gestaltAppearanceCompatMode))
            gInCompatibilityMode = true;

        Gestalt(gestaltAppearanceVersion, &response);
        if(response & 0x00000101 == 0x00000101)
            gVersion101Present = true;
    }
    else
        ExitToShell();

    // ..... cause the Appearance-compliant menu bar definition function to be called directly

    RegisterAppearanceClient();

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

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

```

```

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

// ..... open windows, set font size, show windows, move windows

if(!(gWindowPtr1 = GetNewCWindow(rNewWindow1, NULL, (WindowPtr)-1)))
    ExitToShell();

SetPort(gWindowPtr1);
TextSize(10);
ShowWindow(gWindowPtr1);
doMoveWindowForThemes(gWindowPtr1, true);

if(!(gWindowPtr2 = GetNewCWindow(rNewWindow2, NULL, (WindowPtr)-1)))
    ExitToShell();

SetPort(gWindowPtr2);
TextSize(10);
ShowWindow(gWindowPtr2);
doMoveWindowForThemes(gWindowPtr2, true);

// ..... set Appearance-compliant colour/pattern for second window
SetThemeWindowBackground(gWindowPtr2, kThemeActiveDialogBackgroundBrush, true);

// ..... get pixel depth and whether colour device for certain Appearance functions
doGetDepthAndDevice();

// ..... set top edit text field rectangle as target for initial keyboard focus frame
SetRect(&gCurrentRect, 20, 141, 239, 162);

// ..... enter eventLoop

gDone = false;

while(!gDone)
{
    if(WaitNextEvent(everyEvent, &EventStructure, MAXLONG, NULL))
        doEvents(&EventStructure);
}

// ***** doInitManagers

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

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

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

// ***** doEvents

void doEvents(EventRecord *eventStrucPtr)
{
    SInt8    charCode;
    SInt32   menuChoice;
    SInt16   menuID, menuItem;
    SInt16   partCode;
    WindowPtr windowPtr;
    Str255   itemName;
    SInt16   dadriverRefNum;

```

```

switch(eventStrucPtr->what)
{
case keyDown:
case autoKey:
    charCode = eventStrucPtr->message & charCodeMask;
    if((eventStrucPtr->modifiers & cmdKey) != 0)
    {
        menuChoice = MenuEvent(eventStrucPtr);
        menuID = HiWord(menuChoice);
        menuItem = LoWord(menuChoice);
        if(menuID == mFile && menuItem == iQuit)
            gDone = true;
    }
    break;

case mouseDown:
    if(partCode = FindWindow(eventStrucPtr->where, &windowPtr))
    {
        switch(partCode)
        {
        case inMenuBar:
            menuChoice = MenuSelect(eventStrucPtr->where);
            menuID = HiWord(menuChoice);
            menuItem = LoWord(menuChoice);

            if(menuID == 0)
                return;

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

            case mFile:
                if(menuItem == iQuit)
                    gDone = true;
                break;
            }
            HiLiteMenu(0);
            break;

        case inContent:
            if(windowPtr != FrontWindow())
                SelectWindow(windowPtr);
            else
            {
                if(FrontWindow() == gWindowPtr2)
                {
                    SetPort(gWindowPtr2);
                    doChangeKeyboardFocus(eventStrucPtr->where);
                }
            }
            break;

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

case updateEvt:
    doUpdate(eventStrucPtr);
    break;

case activateEvt:
    doActivate(eventStrucPtr);
    break;

case osEvt:

```

```

        doOSEvent(eventStrucPtr);
        Hi li teMenu(0);
        break;
    }
}

// ***** doUpdate

void doUpdate(EventRecord *eventStrucPtr)
{
    WindowPtr windowPtr;

    windowPtr = (WindowPtr) eventStrucPtr->message;

    BeginUpdate(windowPtr);

    SetPort(windowPtr);

    if(windowPtr == gWindowPtr2)
    {
        if(gWindowPtr2 == FrontWindow() && !gInBackground)
        {
            doDrawThemePrimitives(kThemeStateActive);
            doDrawThemeCompliantText(windowPtr, kThemeStateActive);
            DrawThemeFocusRect(&gCurrentRect, true);
        }
        else
        {
            doDrawThemePrimitives(kThemeStateDisabled);
            doDrawThemeCompliantText(windowPtr, kThemeStateDisabled);
        }
    }

    if(windowPtr == gWindowPtr1)
        doDrawListView(windowPtr);

    EndUpdate(windowPtr);
}

// ***** doActivate

void doActivate(EventRecord *eventStrucPtr)
{
    WindowPtr windowPtr;
    Boolean becomingActive;

    windowPtr = (WindowPtr) eventStrucPtr->message;
    becomingActive = ((eventStrucPtr->modifiers & activeFlag) == activeFlag);
    doActivateWindow(windowPtr, becomingActive);
}

// ***** doActivateWindow

void doActivateWindow(WindowPtr windowPtr, Boolean becomingActive)
{
    if(windowPtr == gWindowPtr2)
    {
        SetPort(gWindowPtr2);

        doDrawThemePrimitives(becomingActive);
        doDrawThemeCompliantText(windowPtr, becomingActive);
        DrawThemeFocusRect(&gCurrentRect, becomingActive);
    }
}

// ***** doOSEvent

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

```



```

// ***** doMoveWindowForThemes
void doMoveWindowForThemes(WindowPtr windowPtr, Boolean bringToFront)
{
    Rect    portRect, structureRect, contentRect;
    Point   structureTopLeft, contentTopLeft;
    SInt32  difference;
    SInt16  contentRectTop, contentRectLeft, menuBarHeight;

    portRect = windowPtr->portRect;
    LocalToGlobal(&topLeft(portRect));
    contentRectTop = portRect.top;
    contentRectLeft = portRect.left;

    structureRect = (*((WindowPeek) windowPtr)->strucRgn)->rgnBBox;
    contentRect = (*((WindowPeek) windowPtr)->contRgn)->rgnBBox;

    structureTopLeft = topLeft(structureRect);
    contentTopLeft = topLeft(contentRect);
    difference = DeltaPoint(contentTopLeft, structureTopLeft);

    contentRectLeft += (*(Point*) &difference).h;
    contentRectTop += (*(Point*) &difference).v;

    menuBarHeight = LMGetMBarHeight();
    contentRectTop += menuBarHeight;

    MoveWindow(windowPtr, contentRectLeft, contentRectTop, bringToFront);
}

// ***** doDrawThemePrimitives
void doDrawThemePrimitives(ThemeDrawState inState)
{
    Rect theRect;

    SetRect(&theRect, -1, -1, 261, 26);
    DrawThemeWindowHeader(&theRect, inState);

    SetRect(&theRect, 20, 46, 119, 115);
    DrawThemePrimaryGroup(&theRect, inState);

    SetRect(&theRect, 140, 46, 239, 115);
    DrawThemeSecondaryGroup(&theRect, inState);

    SetRect(&theRect, 20, 127, 240, 128);
    DrawThemeSeparator(&theRect, inState);

    SetRect(&theRect, 20, 141, 239, 162);
    DrawThemeEditTextFrame(&theRect, inState);

    SetRect(&theRect, 20, 169, 239, 190);
    DrawThemeEditTextFrame(&theRect, inState);

    SetRect(&theRect, 20, 203, 62, 245);
    DrawThemeGenericWell(&theRect, inState, false);

    SetRect(&theRect, 20, 258, 62, 300);
    DrawThemeGenericWell(&theRect, inState, true);

    SetRect(&theRect, 75, 202, 76, 302);
    DrawThemeSeparator(&theRect, inState);

    SetRect(&theRect, 90, 203, 239, 300);
    DrawThemeListBoxFrame(&theRect, inState);

    SetRect(&theRect, -1, 321, 261, 337);
    DrawThemePlacard(&theRect, inState);
}

// ***** doDrawThemeCompliantText
void doDrawThemeCompliantText(WindowPtr windowPtr, ThemeDrawState inState)
{
    SInt16    windowWidth, stringWidth;
    Str255    message = "\pBalloon help is available";

```

```

if(inState == kThemeStateActive)
    SetThemeTextColor(kThemeActiveWindowHeaderTextColor, gPixelDepth, gIsColourDevice);
else
    SetThemeTextColor(kThemeInactiveWindowHeaderTextColor, gPixelDepth, gIsColourDevice);

windowWidth = (windowPtr->portRect.right - (windowPtr->portRect.left);
stringWidth = StringWidth(message);
MoveTo((windowWidth / 2) - (stringWidth / 2), 17);
DrawString("\pBalloon help is available");
}

// ***** doDrawListView

void doDrawListView(WindowPtr windowPtr)
{
    Rect    theRect;
    Sint16  a;

    theRect = windowPtr->portRect;

    SetThemeBackground(kThemeListViewBackgroundBrush, gPixelDepth, gIsColourDevice);
    EraseRect(&theRect);

    theRect.left += 130;

    SetThemeBackground(kThemeListViewSortColumnBackgroundBrush, gPixelDepth, gIsColourDevice);
    EraseRect(&theRect);

    SetThemePen(kThemeListViewSeparatorBrush, gPixelDepth, gIsColourDevice);

    theRect = windowPtr->portRect;
    for(a=theRect.top; a<=theRect.bottom; a+=18)
    {
        MoveTo(theRect.left, a);
        LineTo(theRect.right - 1, a);
    }

    SetThemeTextColor(kThemeListViewTextColor, gPixelDepth, gIsColourDevice);

    for(a=theRect.top; a<=theRect.bottom + 18; a+=18)
    {
        MoveTo(theRect.left, a - 5);
        DrawString("\p  List View Background          List View Sort Column");
    }
}

// ***** doChangeKeyboardFocus

void doChangeKeyboardFocus(Point mouseXY)
{
    Rect    edit1Rect, edit2Rect, listRect;

    DrawThemeFocusRect(&gCurrentRect, false);
    DrawThemeEditTextFrame(&gCurrentRect, kThemeStateActive);

    SetRect(&edit1Rect, 20, 141, 239, 162);
    SetRect(&edit2Rect, 20, 169, 239, 190);
    SetRect(&listRect, 90, 203, 239, 300);

    GlobalToLocal(&mouseXY);

    if(PtInRect(mouseXY, &edit1Rect))
        SetRect(&gCurrentRect, 20, 141, 239, 162);
    else if(PtInRect(mouseXY, &edit2Rect))
        SetRect(&gCurrentRect, 20, 169, 239, 190);
    else if(PtInRect(mouseXY, &listRect))
        SetRect(&gCurrentRect, 90, 203, 239, 300);

    DrawThemeFocusRect(&gCurrentRect, true);
}

// ***** doGetDepthAndDevice

void doGetDepthAndDevice(void)
{
    GDHandle  deviceHdl;

```

```

deviceHdl = LMGetMainDevice();
gPixelDepth = (*(deviceHdl)->gdPMap)->pixelSize;
if(BitTst(&(deviceHdl)->gdFlags, gdDevType))
    gIsColourDevice = true;
}

// *****

```

Demonstration Program Comments

When this program is run, the user should:

- First drag the top window to a position where the content of the bottom window is visible.
- Choose Show Balloons from the Help menu and move the cursor over the frames in the window titled "Drawing With Primitives" window (when active), and the left and right sides of the window titled "Theme-Compliant List View" (when active), noting the descriptions in the balloons.
- With the "Drawing With Primitives" window frontmost, click in the edit text field frame not currently outlined with the keyboard focus frame, or in the list box frame, so as to move the keyboard focus frame to that rectangle.
- Click on the desktop to send the application to the background and note the changed appearance of the frames and text in the "Drawing With Primitives" window. Note also that there is no change to the appearance of the content region of the "Theme-Compliant List View" window. Click on the "Drawing With Primitives" window to bring the application to the foreground with that window active, noting the changed appearance of the frames and text.

In the following, reference is made to graphics devices and pixel depth. Graphics devices and pixel depth are explained at Chapter 11 – QuickDraw Preliminaries.

#define

The first block establishes constants representing menu IDs, resources, and menu items, and window and menu bar resources.

MAXLONG is defined as the maximum possible long value, and is used in the WaitNextEvent function. The last line defines a common macro which converts the top and left fields of a Rect structure to a Point.

Global Variables

gInCompatibilityMode will be assigned true if the machine on which the demonstration is running is in compatibility mode. gVersion101Present will be assigned the version number of the Appearance Manager present on the machine.

gDone, when set to true, causes the main event loop to be exited and the program to terminate. gInBackground relates to foreground/background switching. gWindowPtr1 and gWindowPtr2 will be assigned window pointers.

gPixelDepth will be assigned the pixel depth of the main device. gIsColourDevice will be assigned true if the graphics device is a colour device and false if it is a monochrome device. The values in these two variables are required by certain Appearance functions. gCurrentRect will be assigned the rectangle which is to be the current target for the keyboard focus frame.

main

After the system software managers are initialised, Gestalt is called to determine whether the Appearance Manager and its functions are present. If so, bit 1 in the response is tested to determine whether the machine on which the program is running is currently in compatibility mode, and Gestalt is called again to determine whether the version of the Appearance Manager present is 1.0.1 or later. If the Appearance Manager and its functions are not present, the program simply terminates.

Note that the assignments to the global variables `gInCompatibilityMode` and `gVersion101Present` are for demonstration purposes only; the program does not use these variables for any purpose. (In a real application, you might, for example, use `gVersion101Present` to bypass calls to `DrawThemeModellessDialogFrame`, `DrawThemeGenericWell`, `DrawThemeFocusRegion`, `SetThemeTextColor`, `GetThemeAccentColors`, and `IsThemeInColor`, since those functions must not be called unless Version 1.0.1 or later is present.)

The call to `RegisterAppearanceClient` means that the new Appearance-compliant menu bar definition function (resource ID 63) will be used regardless of whether system-wide Appearance is selected on or off in the Appearance control panel.

After the menus are set up, each window is created. After each window is created, its graphics port is set as the current port and the text size for that port is set to 10pt, the window is shown, and an application-defined function is called to move the window to a position which accounts for varying window frame widths and title bar heights under themes.

`SetThemeWindowBackground` sets an Appearance-compliant colour/pattern for the "Drawing With Primitives" window's content area. This means that the content area will be automatically repainted with that colour/pattern when required with no further assistance from the application. When true is passed in the third parameter, the content region of the window is invalidated and the content region is repainted immediately.

The call to the application-defined function `doGetDepthAndDevice` determines the current pixel depth of the graphics port, and whether the current graphics device is a colour device, and assigns the results to the global variables `gPixelDepth` and `gIsColourDevice`.

The call to `SetRect` establishes the initial target for the keyboard focus frame. This is the rectangle used by the first edit text field frame.

doEvents

At the `MouseDown` case, the `inContent` case within the `partCode` switch is of relevance to the demonstration.

If the mouse-down was within the content region of a window, and if that window is not the front window, `SelectWindow` is called to bring that window to the front and activate it.

However, if the window is the front window, and if that window is the "Drawing With Primitives" window, that window's graphics port is set as the current graphics port for drawing, and the application-defined function `doChangeKeyboardFocus` is called. That function determines whether the mouse-down was within one of the edit text field frames or the list box frame, and moves the keyboard focus if necessary.

doUpdate

Within the `doUpdate` function, if the window to which the update event relates is the "Drawing With Primitives" window, and if that window is currently the front window:

- Application-defined functions are called to draw the primitives and the window header text in the active mode.
- `DrawThemeFocusRect` is called to draw the keyboard focus frame using the rectangle currently assigned to the global variable `gCurrentRect`.

If, however, the "Drawing With Primitives" window is not the front window, the same calls are made but with the primitives and text being drawn in the inactive mode. Note that no call is required to erase the keyboard focus frame because this will already have been erased when the window was deactivated (see below).

If the window to which update event relates is the "Theme-Compliant List View" window, an application-defined function for drawing the window's content area is called. Note that, for this window, there is no differentiation between active and inactive modes. This is because, for list views, the same brush type constants are used regardless of whether the window is active or inactive.

doActivateWindow

When an activate event is received for the "Drawing With Primitives" window, the application-defined functions for drawing the primitives and the window header text, together with the Appearance function which draws and erases the keyboard focus rectangle, are called. To eliminate the necessity for if/else coding, the `becomingActive` value is used to ensure that, firstly, the primitives and text are drawn in the appropriate mode and, secondly, that the keyboard focus frame is either drawn or erased, depending on whether the window is coming to the front or being sent to the back.

Once again, the "Theme-Compliant List View" window is treated differently because the list view brush constants to be used are the same regardless of whether the window is activated and deactivated.

doMoveWindowForThemes

`doMoveWindowForThemes` is called immediately after the windows are opened to move them to a position which accounts for the differing widths of the frame, and the differing height of the title bar, depending on the current theme. This function is just one of several approaches you might take to accommodating differing window border widths under themes.

For the purposes of the demonstration, the 'WIND' resource for the "Theme-Compliant List View" window establishes the top left of the content region at 0,0 (global coordinates), and the 'WIND' resource for the "Drawing With Primitives" window establishes the top left of the content region at 20,20 (global coordinates). No positioning constants are used.

The first line copies the window's port rectangle (in effect, the bounding rectangle of the content region) to a local variable of type `Rect`. This rectangle will be in local coordinates, so the call to `LocalToGlobal` changes the values in the fields of the `Rect` structure to global coordinates. The top and left fields of the `Rect` are then copied to two local variables.

The next two lines copy the bounding rectangles of the window's structure region and content region to two local `Rect` variables. This is for the purpose of finding the width of the window's frame and the height of its title bar.

At the next two lines, the macro `topLeft` is used to extract the values in the top and left fields of the specified `Rect` structures, which are then assigned to the `v` and `h` fields of two local variables of type `Point`. The call to `DeltaPoint` calculates the distance between the two points, returning a 32-bit value with the vertical distance in the high-order word and the horizontal distance in the low-order word. The horizontal distance represents the width of the window's frame and the vertical distance represents the height of its title bar.

The next two lines add these distances to the values in the two local variables previously assigned the location of the top/left of the content region rectangle in global coordinates. At the first line, and within the brackets, the address of difference is cast as a pointer to a `Point` structure. (A `Point` structure comprises two 16-bit fields - `h` and `v`). This is dereferenced to get the address of the data. Outside the brackets, the `.h` persuades the program that it is now looking at the `h` field of a `Point` structure.

The call to `LMGetMBarHeight` gets the height of the menu bar and adds this to the value in `contentRectTop`.

Finally, `MoveWindow` is called to move the window right by a distance equal to the width of the window frame and down by a distance equal to the height of the title bar plus the height of the menu bar.

doDrawThemePrimitives

`doDrawThemePrimitives` uses Appearance Manager functions for drawing Appearance primitives, and is called to draw the various frames in the "Drawing With Primitives" window. The mode in which the primitives are drawn (active or inactive) is determined by the Boolean value passed in the `inState` parameter.

doDrawThemeCompliantText

`doDrawThemeCompliantText` is called to draw some advisory text in the window header of the "Drawing With Primitives" window. The QuickDraw drawing function `DrawString` does the drawing; however, before the drawing begins, the Appearance function `SetThemeTextColor` is used to set the foreground colour for drawing text, in either the active or inactive modes, so as to comply with the current theme.

At the first two lines, if "Drawing With Primitives" is the active window, `SetThemeTextColor` is called with the `kThemeActiveWindowHeaderTextColor` text colour constant passed in the first parameter. At the next two lines, if the window is inactive, `SetThemeTextColor` is called with `kThemeInactiveWindowHeaderTextColor` passed in the first parameter. Note that `SetThemeTextColor` requires the pixel depth of the graphics port, and whether the graphics device is a colour device or a monochrome device, passed in the second and third parameters.

The next three lines simply adjust QuickDraw's pen location so that the text is drawn centered laterally in the window header frame. The call to `DrawString` draws the specified text.

doDrawListView

doDrawListView draws an Appearance-compliant list view background in the specified window.

The first line copies the window's port rectangle to a local variable of type Rect.

The call to SetThemeBackground sets the background colour/pattern to the colour/pattern represented by the Appearance-compliant brush type constant kThemeListViewBackgroundBrush. The QuickDraw function EraseRect fills the whole of the port rectangle with this colour/pattern.

The next line adjusts the Rect variable's left field so that the rectangle now represents the right half of the port rectangle. The same drawing process is then repeated, but this time with kThemeListViewSortColumnBackgroundBrush passed in the first parameter of the SetThemeBackground call.

SetThemePen is then called with the colour/pattern represented by the constant kThemeListViewSeparatorBrush passed in the first parameter. The rectangle for drawing is then expanded to equate with the port rectangle before the following five lines draw one-pixel-wide horizontal lines, at 18-pixel intervals, from the top to the bottom of the port rectangle.

Finally, some text is drawn in the list view in the Appearance-compliant colour for list views. SetThemeTextColor is called with the kThemeListViewTextColor passed in, following which a for loop draws some text, at 18-pixel intervals, from the top to the bottom of the port rectangle.

doChangeKeyBoardFocus

doChangeKeyBoardFocus is called when a mouse-down occurs in the content region of the "Drawing With Primitives" window.

At the first two lines, Appearance functions are used to, firstly, erase the keyboard focus frame from the rectangle around which it is currently drawn and, secondly, redraw an edit text field frame around that rectangle.

The next three lines make three local variables of type Rect equal to the rectangles for the two edit text field frames and the list box frame.

The call to GlobalToLocal converts the coordinates of the mouse-down to the local coordinates required by the following calls to PtInRect. PtInRect returns true if the mouse-down is within the rectangle passed in the second parameter. If one of the calls to PtInRect returns true, that rectangle is made the current rectangle for keyboard focus by assigning it to the global variable gCurrentRect.

Whatever rectangle is assigned to gCurrentRect, the call to DrawThemeFocusRect draws a Theme-compliant keyboard focus frame around that rectangle.

doGetDepthAndDevice

doGetDepthAndDevice determines the pixel depth of the graphics port, and whether the graphics device is a colour device or a monochrome device, and assigns the results to two global variables. This information is required by certain Appearance functions.