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.

Although introduced with Mac OS 8.0, the Appearance Manager's full impact on the Macintosh user experience was not scheduled to be realised until the release of Mac OS 8.5. Mac OS 8.5 was to be the first release to include several switchable **themes**, one of which (the Platinum theme) had, in fact, been included in Mac OS 8.0. The concept of switchable themes was the main driving force behind the creation of the Appearance Manager.

Essentially, a theme was intended to be an interface "look" that spanned all elements of the user interface (windows, menus, dialog boxes, controls, background colours, alert icons, etc), tying them together with a certain graphic design. Fig 1 shows the same window as it would have appeared in the three themes originally intended to be included in Mac OS 8.5. If one of these themes had been selected by the user, all elements of the user interface (menus, windows, controls, etc.) would have appeared in that theme.

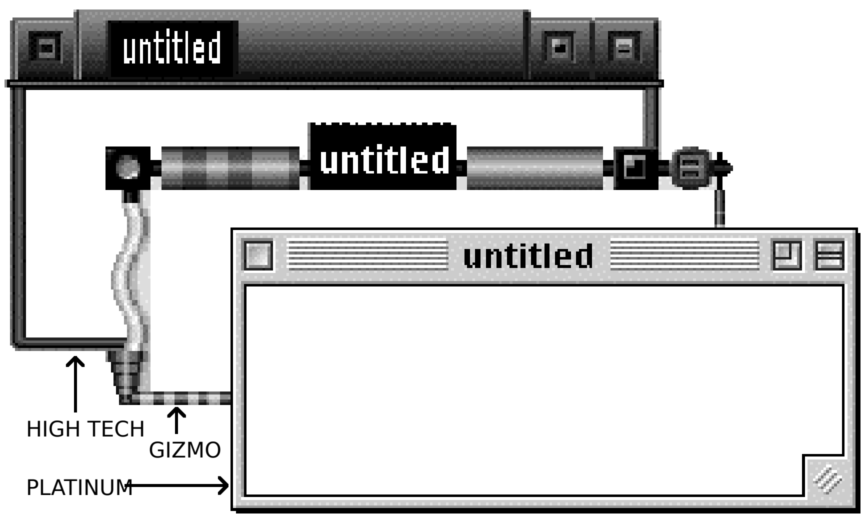

FIG 1 - WINDOWS IN THREE THEMES

The two additional themes (High Tech and Gizmo) shown at Fig 1 were included in pre-release versions of Mac OS 8.5; however, prior to final release, these two themes were deleted. The reasons for this decision by Apple remain tantalisingly obscure.

Themes — New Definition

Mac OS 8.5 did, in fact, introduce a theme scheme, though one of an entirely different flavour to that described above. This is reflected in the Appearance control panel introduced with Mac OS 8.5, in which the Platinum *theme* is now referred to as the Platinum *appearance*. An appearance (new definition) is now simply one component of a broader set of user preferences known as a theme (new definition). With the release of Mac OS 8.5, therefore, the term "theme" took on an entirely new meaning.

Under Mac OS 8.5 and later, an individual theme is a set of user preferences encompassing:

- An appearance (which unifies the look of human interface objects such as windows, dialog boxes, alert boxes, menus, controls, etc.), together with a highlight colour (for selected text) and a variation colour (for menus and controls). (As of Mac OS 8.6, Platinum remains the only appearance provided by Apple.)
- A large system font (for menus and headings), a small system font (for explanatory text and labels), a views font (for lists and icons), and an option to turn anti-aliasing of fonts on screen on or off.
- A desktop picture and desktop pattern.
- Sound preferences relating to opening menus and choosing items, dragging and resizing windows, interacting with controls, and clicking, dragging, and dropping in the Finder.
- Scrolling preference (smart scrolling on or off) and collapse-window preference (double-click title bar to collapse window on or off).

Theme-Compliance

Another significant terminological change ushered in by Mac OS 8.5 was that, whereas Apple documentation previously spoke of making applications **appearance-compliant**, documentation released following the release of Mac OS 8.5 speaks of making applications **theme-compliant**. It is assumed that the reason for this change is that, while the vast bulk of the measures required to make an application theme-compliant relate to unifying the look of the application's user interface elements (the province of an appearance), there are additional measures that the application may take, or may have to take:

- In response to the user changing the system and/or views fonts, using the Fonts tab of the Appearance control panel, while the application is running. (This consideration does not apply if the application uses standard human interface elements (that is, system-defined windows, controls, and menus), since the fonts used for these elements automatically change with the theme change. However, some applications may use custom human interface elements and may, for example, draw their own text in a dialog box. In such cases, the application must ensure that the fonts used match the corresponding system fonts in the current theme.)

- To cause theme-compliant sounds to accompany, for example, the opening and closing of the application's windows and the manipulation by the user of custom human interface elements.

- To support the proportional scroll boxes[1] the user expects when Smart Scrolling is selected on in the Options tab of the Appearance control panel.

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 human interface elements and provides the underlying support for themes, particularly appearances.

- Enables you to adapt your application's custom human interface elements (if any) to the coordinated system-wide look, that is, to make those elements theme-compliant.

Appearance Manager Versions

The version of the Appearance Manager delivered with Mac OS 8.5 was Version 1.1. This was the first version to be included in the System file. Previous versions were delivered as extensions. These extensions can be installed and used on Macintoshes and Power Macintoshes running System 7.1 through 7.6.1 (as well as Power Macintoshes running Mac OS 8.0 or 8.1), meaning that theme-compliant applications running on the System 7 systems can present their human interface elements in the chosen appearance.

The Appearance Manager versions delivered as extensions are Versions 1.0, 1.0.1, 1.0.2, and 1.0.3. Version 1.0.2 or, preferably, Version 1.0.3 must be used on Macintoshes and Power Macintoshes running System 7.1 through 7.6.1. Versions 1.0 and 1.0.1 were delivered with, respectively, Mac OS 8.0 and 8.1; however, it is advisable to upgrade to Version 1.0.2 or, preferably, Version 1.0.3 on such systems.

The only difference between Versions 1.0.1 and 1.0.2 is that Version 1.0.2 contains extra code (for backward compatibility) and the ".Keyboard" font. The ".Keyboard" font is used to display keyboard glyphs in menus.

The only difference between Versions 1.0.2 and 1.0.3 is that Version 1.0.3 no longer contains the ".Keyboard" font. In Version 1.0.3, this font is delivered as a separate suitcase, which should be installed into the Fonts folder in the System folder. The purpose of this latter is to avoid a font ID conflict between the ".Keyboard" font and Microsoft Internet Explorer's Arial font.

[1] Proportional scroll boxes are scroll boxes which vary in size according to the proportion of the document visible in the window.

New Definition Functions

To provide a system-wide coordination of look and behaviour, new theme-compliant definition functions were introduced with the Appearance Manager to replace the old pre-Appearance Manager definition functions for menu bars, menus, windows, and controls. In addition, many new theme-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.

Mapping of Pre-Appearance Manager Definition Functions

Another way in which the Appearance Manager achieved a unified look and behaviour was by **mapping** the following standard pre-Appearance Manager definition functions to their theme-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**. The mappers have the same resource ID as the pre-Appearance definition functions to which they relate.

Under Mac OS 8.5 (Appearance Version 1.1), mapping on a system-wide basis is permanently on. In earlier versions of the Appearance Manager:

- Mapping on a system-wide basis only occurs when the user has selected system-wide Appearance on in the Appearance control panel.
- You can ensure that mapping on an individual application basis will occur when the user has selected system-wide Appearance off in the Appearance control panel by calling the function RegisterAppearanceClient within your application.

Of course, no mapping occurs if your application specifies the new theme-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 theme-compliant control definition function being called directly.

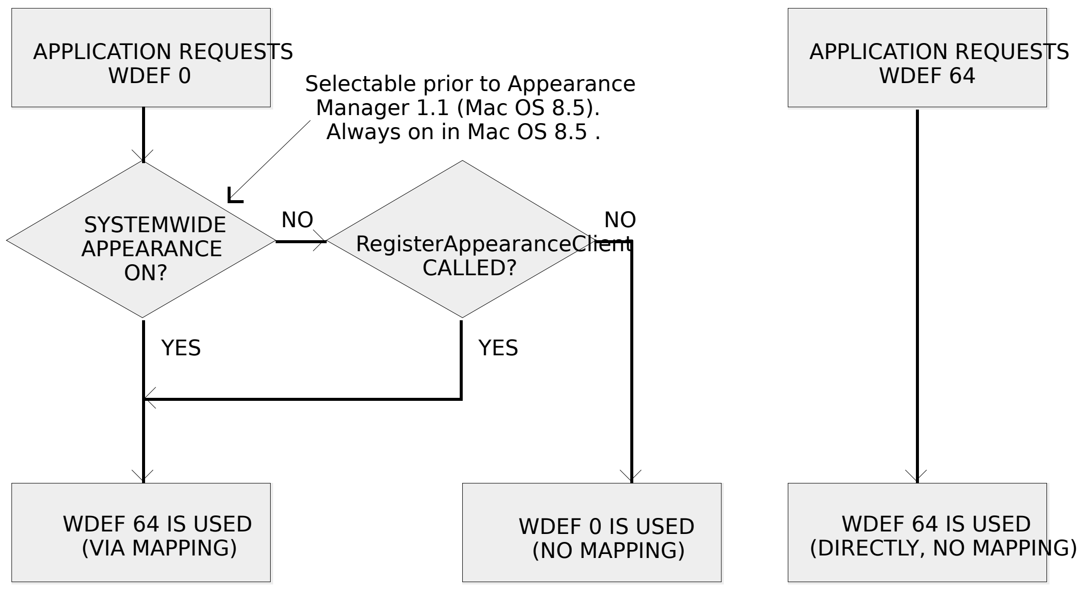
APPLICATION REQUESTS WDEF 0
Selectable prior to Appearance Manager 1.1 (Mac OS 8.5). Always on in Mac OS 8.5 .
SYSTEMWIDE APPEARANCE ON?
NO
RegisterAppearanceClient CALLED?
NO
YES
YES
WDEF 64 IS USED (VIA MAPPING)
WDEF 0 IS USED (NO MAPPING)
APPLICATION REQUESTS WDEF 64
WDEF 64 IS USED (DIRECTLY, NO MAPPING)

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

The RegisterAppearanceClient Function

The following describes the RegisterAppearanceClient function.

Function	***Description***
RegisterAppearanceClient	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. Under Appearance Manager 1.0.3 and earlier, applications that call this function will continue to have the chosen appearance when system-wide appearance is selected off in the Appearance control panel. This function automatically maps standard pre-Appearance definition functions to their theme-compliant equivalents. Although they will not make use of mapping, applications that specify theme-compliant definition function IDs directly should also call this function in order to receive Appearance Manager Apple events (see Chapter 10 — Apple Events).

Disadvantages of Calling a Definition Function Via a Mapper

When a theme-compliant definition function is called via the mappers, the associated object may have a slightly different look and behaviour than is the case when it is 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 a theme-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 a theme-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 theme-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 theme-compliant WDEF is called directly, the three-pixel-wide space is banished.

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

Mapping of Custom Definition Functions

Custom definition functions cannot be automatically mapped to theme-compliant equivalents. However, the Appearance Manager does provide ways to coordinate custom user interface elements with the current appearance. For example, calling DrawThemeListBoxFrame would create a theme-compliant frame for a custom list box.

Checking For the Presence of the 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 (see Chapter 23 — Miscellany) may be used to acquire a wide range of information about the operating environment, and may be used to determine:

- Whether the Appearance Manager is 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. (This applies only under Appearance Manager 1.0.3 and earlier.)
- 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 example shows how to, in sequence, check that the Appearance Manager is present, determine whether system-wide Appearance is on, and determine the version of the Appearance Manager that is present.

```
OSErr    osError;
SInt32      response;
Boolean  appearancePresent              = false;
Boolean  appearance101present           = false;
Boolean  appearance110present           = false;
Boolean  inCompatibilityMode            = false;

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)))
{
   // At least Version 1.0 is present.  Set a flag.

   appearancePresent = true;

   // If the bit in response represented by the constant gestaltAppearanceCompatMode
   // is set, system-wide Appearance is off.  The result of this check will be
   // relevant only where Versions 1.0 through 1.0.3 are present.

   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, 1.0.2, or 1.0.3 is
   // present. If the low order word in response is 0x0110, Version 1.1 is available.
```

```
        if(response == 0x00000101)
            gAppearance101present = true;
        else if(response == 0x00000110)
            gAppearance110present = true;
    }
    else
    {
        // Present nil-Appearance error alert presented here, then exit.
    }
```

Colours, Patterns, and the Current Appearance

The Appearance Manager provides drawing primitives, and the means to set the colours and patterns, needed to draw consistently in a given appearance. Using these drawing primitives, colours, and patterns makes it easier to create visual entities and custom human interface elements that are consistent with the current appearance.

Drawing Appearance Primitives

The Appearance Manager 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 theme-compliant visual representation of that control.

Your application might use these primitives to, for example:

- Draw a theme-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 theme-compliant by using DrawThemeListBoxFrame to draw the frame and DrawThemeFocusRect to draw the focus ring.

The following are examples of functions that draw Appearance primitives. Those appearing on a light gray background are available only in Appearance Version 1.0.1 through 1.0.3 and later. Those appearing on a dark gray background are available only in Appearance Version 1.1 and later.

Function	***Description***
DrawThemePrimaryGroup	Draws a primary group box frame consistent with the current appearance.
DrawThemeSecondaryGroup	Draws a secondary group box frame consistent with the current appearance. Allows you to nest a secondary group box frame within the primary group box frame.
DrawThemeSeparator	Draws a separator line consistent with the current appearance. 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.
DrawThemeWindowHeader	Draws a window header consistent with the current appearance. This function draws a window header such as that used by the Finder. The window header is drawn inside the rectangle that is passed.
DrawThemeWindowListViewHeader	Draws a window list view header, such as that used by the Finder, consistent with the current appearance. 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.
DrawThemePlacard	Draws a placard consistent with the current appearance.
DrawThemeEditTextFrame	Draws an edit text field frame consistent with the current appearance. The rectangle passed in should be the same as the one passed in the function DrawThemeFocusRect (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.
DrawThemeListBoxFrame	Draws a list box frame consistent with the current appearance. The rectangle passed in should be the same as the one passed into the function DrawThemeFocusRect (see below) so that you get the correct focus look for your list box.
DrawThemeFocusRect	Draws or erases a focus ring around a specified rectangle. To achieve the right look, you should first call DrawThemeEditTextFrame or DrawThemeListBoxFrame and then call DrawThemeFocusRect, passing the same rectangle in the inRect parameter. If you use DrawThemeFocusRect 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.
DrawThemeModelessDialogFrame	Draws a modeless dialog box frame, like the one drawn by the Dialog Manager, consistent with the current appearance. This function may be used to make a custom modeless dialog box theme-compliant. The purpose of the modeless dialog frame is to assist in making modeless dialog windows visually distinguishable from normal document windows.
DrawThemeGenericWell	Draws an image well frame consistent with the current appearance. Image well frames are for use with custom image well controls. You can specify that the centre of the well be filled with white.
DrawThemeFocusRegion	Draws or erases a theme-compliant focus ring around a specified region.
DrawThemeTabPane	Draws a tab-pane consistent with the current appearance.
DrawThemeTab	Draws a tab consistent with the current appearance.

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

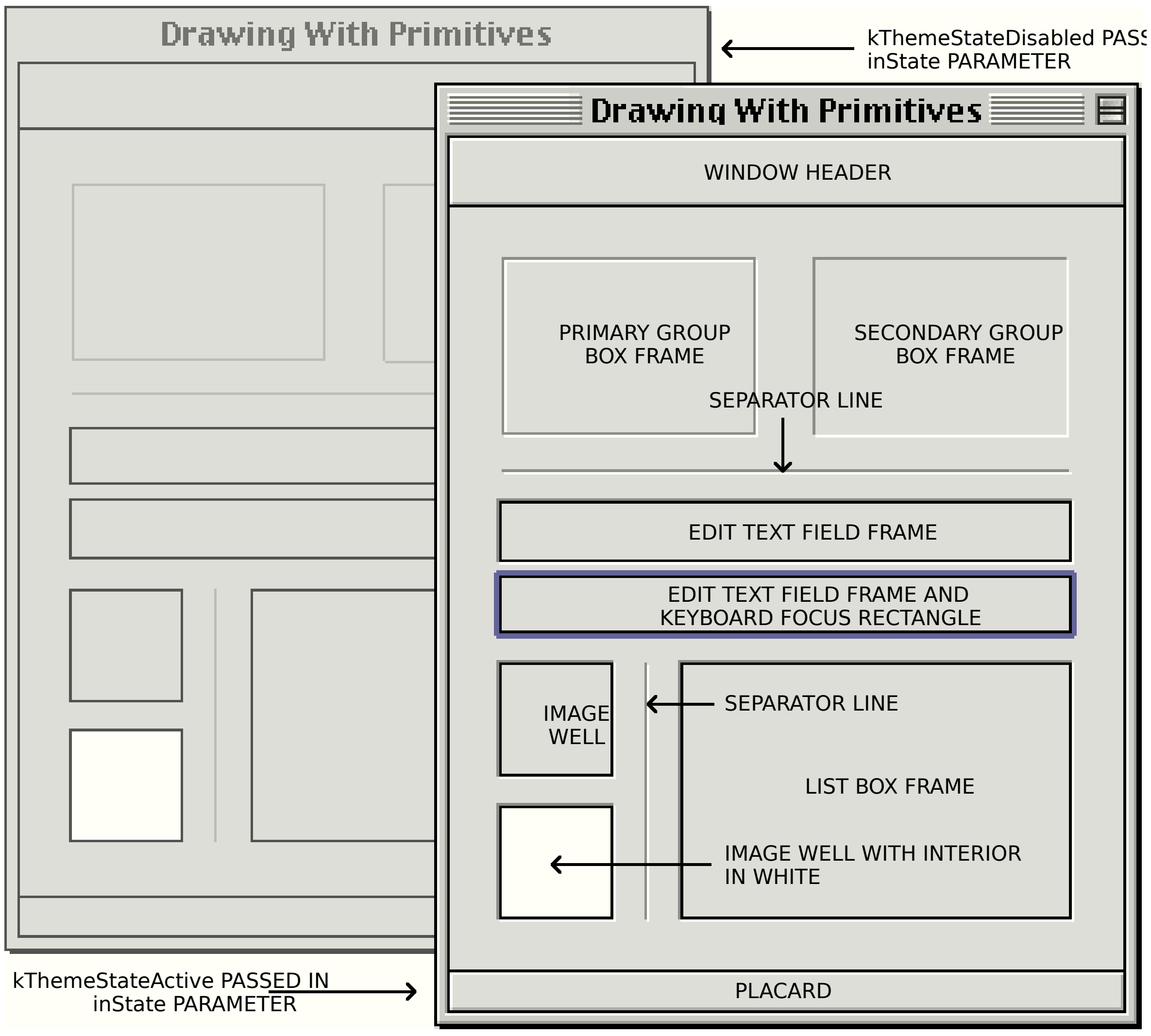

FIG 3 - IMAGES DRAWN WITH OTHER APPEARANCE DRAWING PRI

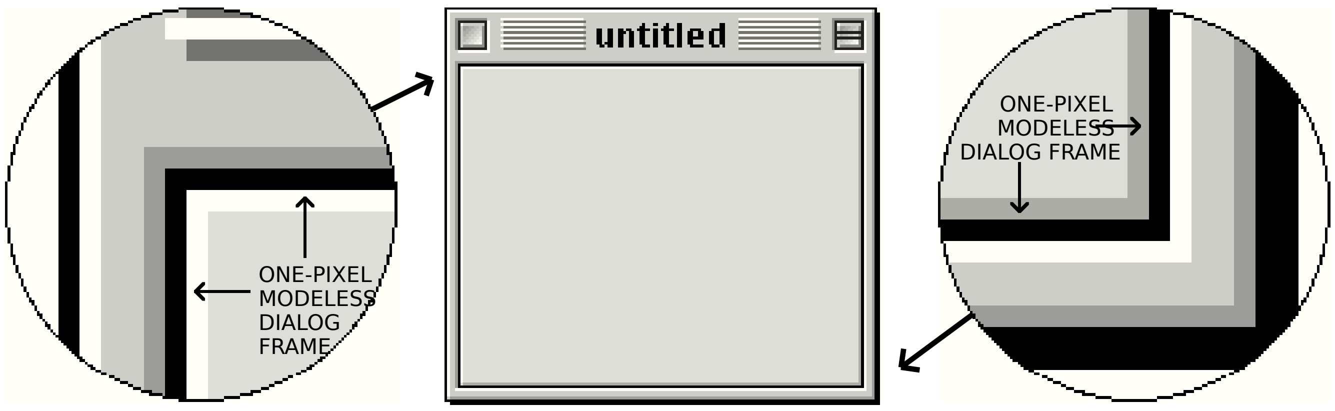
untitled
ONE-PIXEL MODELESS DIALOG FRAME
ONE-PIXEL MODELESS DIALOG FRAME

FIG 4 - MODELESS DIALOG FRAME DRAWN WITH APPEARANCE DRAWING P

Draw State Constants

The following constants are passed in the inState parameter of the functions that draw Appearance primitives (except DrawThemeFocusRect and DrawThemeFocusRegion) to specify whether the primitive should be drawn in the active or deactivated mode.[2]

Constant	***Value***	***Description***
kThemeStateDisabled	0	Draw the primitive in the inactive mode.
kThemeStateActive	1	Draw the primitive in the active mode.

2 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.

Drawing in Colours and Patterns Consistent With the Current Appearance

The following functions are those used to draw using colours/**patterns** consistent with the current appearance. (Patterns are explained at Chapter 11 — QuickDraw Preliminaries.) The reference to colours *and* patterns reflects the fact that, depending on the current appearance, either a colour or a pattern may be used for the drawing.

Function	*Description*
SetThemeWindowBackground	Sets the theme-compliant colour/pattern that the window background will be repainted to when PaintOne is called. This function sets the colour/pattern to which the Window Manager will erase the window background. See also Theme-Compliant Brush Type Constants, below.
SetThemeBackground	Sets an element's background colour/pattern to comply with the current appearance. 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 Theme-Compliant Brush Type Constants, below.
SetThemePen	Sets an element's pen pattern or colour to comply with the current appearance. 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 Theme-Compliant Brush Type Constants, below.
SetThemeTextColor	Sets an element's foreground colour for drawing text to comply with the current appearance. This function is typically used inside a DeviceLoop drawing procedure to set the foreground colour for drawing text in order to coordinate with the current appearance. See also Theme-Compliant Text Colour Constants, below.

Theme-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 theme-compliant colours/patterns for user interface elements. For reasons explained above, these constants can represent either a straight colour or a pattern.

Constant	*Description*
kThemeBrushDialogBackgroundActive	An active dialog box's background colour/ pattern.
kThemeBrushDialogBackgroundInactive	An inactive dialog box's background colour or pattern.
kThemeBrushAlertBackgroundActive	An active alert box's background colour/pattern.
kThemeBrushAlertBackgroundInactive	An inactive alert box's background colour/pattern.
kThemeBrushModelessDialogBackgroundActive	An active modeless dialog box's background colour/pattern.
kThemeBrushModelessDialogBackgroundInactive	An inactive modeless dialog box's background colour/pattern.
kThemeBrushUtilityWindowBackgroundActive	An active utility window's background colour/pattern.
kThemeBrushUtilityWindowBackgroundInactive	An inactive utility window's background colour/pattern.
kThemeBrushListViewSortColumnBackground	The background colour/pattern of the column upon which a list view is sorted.
kThemeBrushListViewBackground	The background colour/pattern of a list view column that is not being sorted upon.

kThemeBrushListViewSeparator	A list view separator's colour/pattern.
kThemeBrushDocumentWindowBackground	A document window's background colour/pattern.
kThemeBrushFinderWindowBackground	A Finder window's background colour/pattern. Generally, you should not use this constant unless you are trying to create a window that matches the Finder window.

Theme-Compliant Text Colour Constants

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

Constant	Description
kThemeTextColorWindowHeaderActive	Text colour for active window header.
kThemeTextColorWindowHeaderInactive	Text colour for inactive window header.
kThemeTextColorPlacardActive	Text colour for active placard.
kThemeTextColorPlacardInactive	Text colour for inactive placard.
kThemeTextColorPlacardPressed	Text colour for highlighted placard.
kThemeListViewTextColor	Text colour for list view.

Saving and Setting the Colour Graphics Port Drawing State

Chapter 12 — Drawing With QuickDraw addresses certain measures which need to be taken consequential to the fact that both colours and patterns can be used by the Appearance functions SetThemeWindowBackground, SetThemeBackground, and SetThemePen. These measures have to do with saving, restoring, and normalising the drawing state of the graphics port.

Version 1.1 of the Appearance Manager introduced new functions which simplify this process. These functions, the uses of which are addressed at Chapter 12, are as follows:

Constant	Description
GetThemeDrawingState	Obtain the drawing state of the current colour graphics port.
SetThemeDrawingState	Set the drawing state of the current graphics port.
NormalizeThemeDrawingState	Set the current colour graphics port to the default drawing state.
DisposeThemeDrawingState	Release the memory associated with a reference to a graphics port's drawing state.

Theme-Compliant Cursors

Version 1.1 of the Appearance Manager introduced cursors that can change appearance with an appearance change. The associated functions, the uses of which are addressed at Chapter 13 — Offscreen Graphics Worlds, Pictures, Cursors, and Icons, are as follows:

Constant	Description
SetThemeCursor	Sets the cursor to a version of the cursor consistent with the current appearance.
SetAnimatedThemeCursor	Animates a version of the cursor that is consistent with the current appearance.

Appearance Manager Apple Events

As previously stated, your application may need to respond to the user changing the system and/or views fonts using the Fonts tab in the Appearance control panel. In addition, because an appearance change might result in a change to menu bar height, window structure dimensions, and colours and patterns currently in use, your application may also need to respond to the user changing the current appearance using the Appearance tab in the Appearance control panel..

Provided you have called RegisterAppearanceClient, your application is advised of font and appearance changes via **Appearance Manager Apple events**. Appearance Manager Apple events are addressed at Chapter 10 — Apple Events.

Theme-Compliant Applications

Making a New Application Theme-Compliant

The main consideration involved in making your application theme-compliant is to allow the system to do as much of your interface work as possible. The following lists the actions required to make a new application theme-compliant:

- Call RegisterAppearanceClient early in your application code, prior to drawing any on-screen elements or invoking any definition functions. For many applications, this will be the only action required to render the application fully theme-compliant.
- Use the system-supplied theme-compliant menu and window definition functions.
- As will be explained at Chapter 7 — Introduction to Controls and at Chapter 14 — More on Controls, use the system-supplied theme-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 theme-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.
- Where you absolutely cannot use the system-supplied definition functions, use Appearance Manager functions in your custom definition functions to ensure that your custom human interface elements are theme-compliant.
- Use Appearance Manager functions and constants to get any colours/patterns you need to draw consistently with the current appearance, and to draw theme-compliant visual entities such as window headers when you don't want to use a control of that type.
- Because the measurements of standard interface objects may vary from appearance to appearance, make no assumptions about the dimensions of menus, windows, or controls. (For example, if you assume an unchanging menu bar height in order to position your windows, you could end up with the menu bar overlapping your windows after an appearance change.) Use Appearance Manager functions such as GetThemeMenuBarHeight to obtain the measurements used in the current appearance.
- Use the Appearance Manager functions SetThemeCursor and SetAnimatedThemeCursor to ensure that your application's cursors are theme-compliant.

Making Old Applications Theme-Compliant

Ultimately, the task of making an old non-theme-compliant application fully theme-compliant will involve all of the steps listed at Making a New Application Theme-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 under Versions 1.0 through 1.0.3 of the Appearance Manager, 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.)

Memory Requirements

Some appearances may use simple rectangular shapes, and some may use complex, non-rectangular shapes, for their interface elements. Accordingly, the variable length Region structures in which the descriptions of these shapes are stored will occupy more or less memory depending on the current appearance. If your application's memory usage is fine-tuned on the basis of an appearance which uses simple rectangular shapes for its interface elements, you will need to increase the heap size to account for the possibility that the user will choose an appearance whose interface element shapes are more complex.

Main Constants, Data Types, and Functions

In the following:

- Those items appearing on a light gray background are available only with Appearance Version 1.0.1 through 1.0.3 and later.
- Those items appearing on a dark gray background are available only with Appearance Version 1.1 and later.

Constants

Checking For Appearance, Appearance Functions, and Version

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

Theme-Compliant Brush Type Constants

```
kThemeBrushDialogBackgroundActive           = 1
kThemeBrushDialogBackgroundInactive         = 2
kThemeBrushAlertBackgroundActive            = 3
kThemeBrushAlertBackgroundInactive          = 4
kThemeBrushModelessDialogBackgroundActive   = 5
kThemeBrushModelessDialogBackgroundInactive = 6
kThemeBrushUtilityWindowBackgroundActive    = 7
kThemeBrushUtilityWindowBackgroundInactive  = 8
kThemeBrushListViewSortColumnBackground     = 9
```

kThemeBrushListViewBackground	= 10
kThemeBrushListViewSeparator	= 12
kThemeBrushDocumentWindowBackground	= 15
kThemeBrushFinderWindowBackground	= 16

Theme-Compliant Text Colour Constants

kThemeTextColorWindowHeaderActive	= 7
kThemeTextColorWindowHeaderInactive	= 8
kThemeTextColorPlacardActive	= 9
kThemeTextColorPlacardInactive	= 10
kThemeTextColorPlacardPressed	= 11
kThemeTextColorListView	= 22

Theme-Compliant Draw State Constants (For Primitives)

kThemeStateDisabled	= 0
kThemeStateActive	= 1
kThemeStatePressed	= 2

Theme Cursor Constants

kThemeArrowCursor	= 0	
kThemeCopyArrowCursor	= 1	
kThemeAliasArrowCursor	= 2	
kThemeContextualMenuArrowCursor	= 3	
kThemeIBeamCursor	= 4	
kThemeCrossCursor	= 5	
kThemePlusCursor	= 6	
kThemeWatchCursor	= 7	// Can animate
kThemeClosedHandCursor	= 8	
kThemeOpenHandCursor	= 9	
kThemePointingHandCursor	= 10	
kThemeCountingUpHandCursor	= 11	// Can animate
kThemeCountingDownHandCursor	= 12	// Can animate
kThemeCountingUpAndDownHandCursor	= 13	// Can animate
kThemeSpinningCursor	= 14	// Can animate
kThemeResizeLeftCursor	= 15	
kThemeResizeRightCursor	= 16	
kThemeResizeLeftRightCursor	= 17	

Data Types

```
typedef UInt32 ThemeDrawState;
typedef SInt16 ThemeBrush;
typedef SInt16 ThemeTextColor;
```

Functions

Initialising the Appearance Manager

```
OSStatus  RegisterAppearanceClient    (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    DrawThemeModelessDialogFrame(const Rect *inRect,ThemeDrawState inState);
OSStatus    DrawThemeGenericWell(const Rect *inRect,ThemeDrawState inState,
            Boolean inFillCenter);
OSStatus    DrawThemeFocusRegion(RgnHandle inRegion,Boolean inHasFocus);
OSStatus    DrawThemeTab(const Rect *inRect,ThemeTabStyle inStyle,ThemeTabDirection inDirection,
            ThemeTabTitleDrawUPP labelProc,UInt32 userData);
OSStatus    DrawThemeTabPane(const Rect *inRect,ThemeDrawState inState);
```

Drawing in Colours/Patterns Consistent With the Current Appearance

```
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);
```

Saving and Setting the Colour Graphics Port Drawing State

```
OSStatus    NormalizeThemeDrawingState(void);
OSStatus    GetThemeDrawingState(ThemeDrawingState *outState);
OSStatus    SetThemeDrawingState(ThemeDrawingState inState,Boolean inDisposeNow);
OSStatus    DisposeThemeDrawingState(ThemeDrawingState inState);
```

Setting Appearance Cursors

```
OSStatus    SetThemeCursor(ThemeCursor inCursor);
OSStatus    SetAnimatedThemeCursor(ThemeCursorinCursor,UInt32 inAnimationStep);
```

Demonstration Program

```
// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊
// Appearance.c
// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊
//
// This program opens two kWindowDocumentProc windows containing:
//
// •   In the first window, a theme-compliant list view.
//
// •   In the second window, various images drawn with Appearance primitives and window
//     header text drawn in the correct appearance colour.
//
// 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 <Gestalt.h>
#include <LowMem.h>
#include <Sound.h>
#include <ToolUtils.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       gAppearancePresent    = false;
Boolean       gInCompatibilityMode   = false;
Boolean       gAppearance101present  = false;
Boolean       gAppearance110present  = 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  doDrawAppearancePrimitives       (ThemeDrawState);
void  doDrawAppearanceCompliantText    (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;

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

    osError = Gestalt(gestaltAppearanceAttr,&response);

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

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

        Gestalt(gestaltAppearanceVersion,&response);

        if(response == 0x00000101)
            gAppearance101present = true;
        else if(response >= 0x00000110)
            gAppearance110present = true;
    }
    else
    {
        SysBeep(10);
        ExitToShell();
    }

    // 
..................................................................................................................................................................
... initialise managers

    doInitManagers();
```

```
  // ........................................................................................................................................................ 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);

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

  SetPort(gWindowPtr2);
  TextSize(10);
  ShowWindow(gWindowPtr2);

  // ........................................................................... set theme-compliant colour/pattern for second window

  SetThemeWindowBackground(gWindowPtr2,kThemeBrushDialogBackgroundActive,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);

  RegisterAppearanceClient();
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ 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);
            HiliteMenu(0);
            break;
    }
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doUpdate

void  doUpdate(EventRecord *eventStrucPtr)
{
    WindowPtr    windowPtr;

    windowPtr = (WindowPtr) eventStrucPtr->message;

    BeginUpdate(windowPtr);

    SetPort(windowPtr);

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

        doDrawAppearancePrimitives(becomingActive);
        doDrawAppearanceCompliantText(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;
    }
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doDrawAppearancePrimitives

void  doDrawAppearancePrimitives(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);

    if(gAppearance101present || gAppearance110present)
    {
        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);
}

// ◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊◊ doDrawAppearanceCompliantText

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

    if(inState == kThemeStateActive)
        SetThemeTextColor(kThemeTextColorWindowHeaderActive,gPixelDepth,gIsColourDevice);
    else
        SetThemeTextColor(kThemeTextColorWindowHeaderInactive,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(kThemeBrushListViewBackground,gPixelDepth,gIsColourDevice);
    EraseRect(&theRect);
    theRect.left += 130;

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

    SetThemePen(kThemeBrushListViewSeparator,gPixelDepth,gIsColourDevice);

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

    SetThemeTextColor(kThemeTextColorListView,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, listRec;

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

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

    SetPort(gWindowPtr2);
    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,&listRec))
        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

gAppearancePresent will be assigned true if at least Version 1.0 of the Appearance Manager is present. gInCompatibilityMode will be assigned true if the machine on which the demonstration is running is in compatibility mode (applicable only to Versions 1.0 through 1.0.3 only). gAppearance101present will be assigned true if Versions 1.0.1, 1.0.2, or 1.0.3 are present. gAppearance110present will be assigned true if Version 1.1 is present.

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

Gestalt is called to determine whether some version of the Appearance Manager is present. If so, bit 1 in response is tested to determine whether the machine on which the program is running is currently in compatibility mode (relevant only where Appearance Manager Versions 1.0 through 1.0.3 are present), and Gestalt is called again to determine whether Version 1.0.1 through 1.0.3, or Version 1.1 or later, is present. If the Appearance Manager is not present, the system alert sound is played and the program simply terminates.

Note that the assignment to the global variable gInCompatibilityMode is for demonstration purposes only; the program does not use this variables for any purpose.

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.

SetThemeWindowBackground sets a theme-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.

doInitManagers

DoInitManagers is called from main immediately after it has been determined that the Appearance Manager is present. In this demonstration program, and in all subsequent demonstration programs, a call to RegisterAppearanceClient has been added to this function.

If this program is run under Appearance Manager Versions 1.0 through 1.0.3, one effect of the call to RegisterAppearanceClient is that the new theme-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.

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.

doDrawAppearancePrimitives

doDrawAppearancePrimitives 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.

Note that DrawThemeGenericWell, which was introduced with Version 1.0.1 of the Appearance Manager, is called only if Versions 1.0.1 through 1.0.3, or Version 1.1, are present.

doDrawAppearanceCompliantText

doDrawAppearanceCompliantText 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 appearance.

At the first two lines, if "Drawing With Primitives" is the active window, SetThemeTextColor is called with the kThemeTextColorWindowHeaderActive text colour constant passed in the first parameter. At the next two lines, if the window is inactive, SetThemeTextColor is called with kThemeTextColorWindowHeaderInactive 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 a theme-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 theme-compliant brush type constant kThemeBrushListViewBackground. 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 kThemeBrushListViewSortColumnBackground passed in the first parameter of the SetThemeBackground call.

SetThemePen is then called with the colour/pattern represented by the constant kThemeBrushListViewSeparator 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 theme-compliant colour for list views. SetThemeTextColour is called with the kThemeTextColorListView 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.