

6

THE APPEARANCE MANAGER

Includes Demonstration Program AppearanceDemo

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.

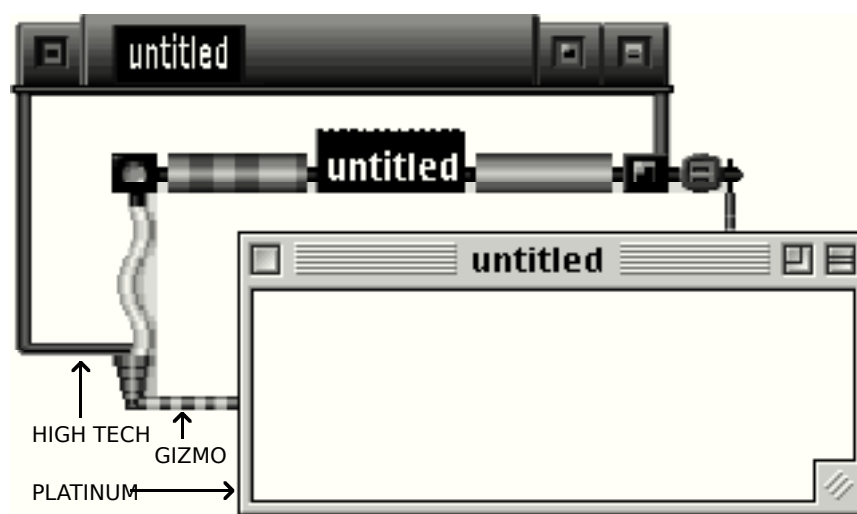


FIG 1 - WINDOWS IN THREE THEMES

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. At the time of writing (March 1999), the reasons for this decision remain tantalisingly obscure, and there is now at least some doubt as to whether multiple theme choices will be offered by Apple in any future Mac OS release.

Terminological Confusion — Themes and Appearances

Unfortunately, the Apple decision to delete all but the Platinum theme from the release version of Mac OS 8.5 has resulted in a certain amount of terminological confusion. This arises from the fact that Mac OS 8.5 did, in fact, introduce a theme scheme, though one of an entirely different flavour to that described above. The themes that may be chosen in the Themes tab in the Mac OS 8.5 Appearance control panel are nothing more than simple collections of desktop patterns and pictures, system fonts, highlight colours, system sounds, etc. Nowadays, therefore, the term "theme" has a meaning which differs markedly from that which applied immediately prior to the release of Mac OS 8.5.

The same now applies to the term "**appearance**". In the Appearance tab in the Mac OS 8.5 Appearance control panel, the pop-up menu labelled Appearance would have been labelled Theme in the originally envisaged scheme of things. Thus an "appearance", not a "theme", is now the thing that unifies the look of human interface elements in your application. The fact that only one item (Apple platinum) appears in the Mac OS 8.5 Appearance pop-up menu bears testimony to the current uncertainty regarding the future of what must now be referred to as switchable appearances.

Without a knowledge of this background, the beginning programmer could be forgiven for being somewhat confused by the fact that "Theme" features in so many Appearance Manager constant, data type, and function names (eg., `kThemeDragHiliteBrush`, `ThemeDrawState`, `DrawThemePlacard`). To alleviate this confusion, and in the light of the history outlined above, it might be advisable to mentally substitute "Appearance" for "Theme" in these constant, data type, and function names.

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 appearance.
- 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 for appearances and appearance switching.

Appearance Manager Versions

The version of the Appearance Manager delivered with Mac OS 8.5 (Version 1.1) is 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 Appearance-compliant applications running on the System 7 systems can present their human interface elements in the Platinum 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.

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.

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**. 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 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.

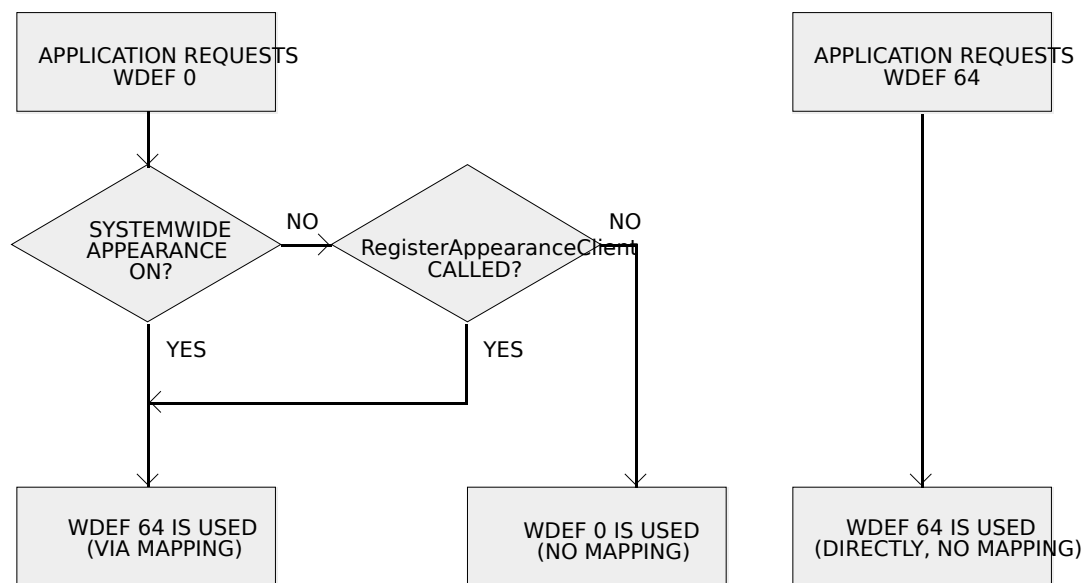


FIG 2 - MAPPING A STANDARD PRE-APPEARANCE DEFINITION FUNCTION TO ITS APPEARANCE-COMPLIANT EQUIVALENT AND CALLING AN APPEARANCE-COMPLIANT DEFINITION FUNCTION DIRECTLY

The RegisterAppearanceClient Function

The following describes the `RegisterAppearanceClient` function.

Function	Description
<code>RegisterAppearanceClient</code>	<p>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.</p> <p>Under Appearance Manager 1.0.3 and earlier, applications that call this function will continue to have the Platinum appearance when system-wide appearance is selected off in the Appearance control panel.</p> <p>This function automatically maps standard pre-Appearance definition functions to their Appearance-compliant equivalents. Although they will not make use of mapping, applications that specify Appearance-compliant definition function IDs directly should also call this function in order to receive Appearance Manager Apple events.</p>

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.

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 (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.

```
OsError : OSErr;
response : SInt32;
gAppearancePresent : boolean;
gAppearance101present : boolean;
gAppearance110present : boolean;
gInCompatibilityMode : boolean;
```

...

```

gAppearancePresent := false;
gAppearance101present := false;
gAppearance110present := false;
gInCompatibilityMode := 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) and BitTst(response, gestaltAppearanceExists)) then
    begin
        // At least Version 1.0 is present. Set a flag.

        gAppearancePresent := 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, gestaltAppearanceCompatMode)) then
            begin
                gInCompatibilityMode := true;
            end;

        { Call Gestalt again with the gestaltAppearanceVersion selector. }

        osError := 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 = $00000101) then
            begin
                gAppearance101present := true;
            end
        else if (response = $00000110) then
            begin
                gAppearance110present := true;
            end
        end
    else begin
        { Nil-Appearance error alert presented here, then exit. }
    end;
end;

```

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 the Platinum appearance. Using these drawing primitives, colours, and patterns makes it easier to create visual entities and custom definition functions 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 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 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
<code>DrawThemePrimaryGroup</code>	Draws a primary group box frame consistent with the current appearance.
<code>DrawThemeSecondaryGroup</code>	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.
<code>DrawThemeSeparator</code>	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.
<code>DrawThemeWindowHeader</code>	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.
<code>DrawThemeWindowListViewHeader</code>	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.
<code>DrawThemePlacard</code>	Draws a placard consistent with the current appearance.
<code>DrawThemeEditTextFrame</code>	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 <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 appearance. 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>DrawThemeFocusRect</code>	Draws or erases a focus ring around a specified rectangle. 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>DrawThemeModelessDialogFrame</code>	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 Appearance-compliant. The purpose of the modeless dialog frame is to assist in making modeless dialog windows visually distinguishable from normal document windows.
<code>DrawThemeGenericWell</code>	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 center of the well be filled with white.
<code>DrawThemeFocusRegion</code>	Draws or erases an Appearance-compliant focus ring around a specified region.
<code>DrawThemeTabPage</code>	Draws a tab-pane consistent with the current appearance.
<code>DrawThemeTab</code>	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.

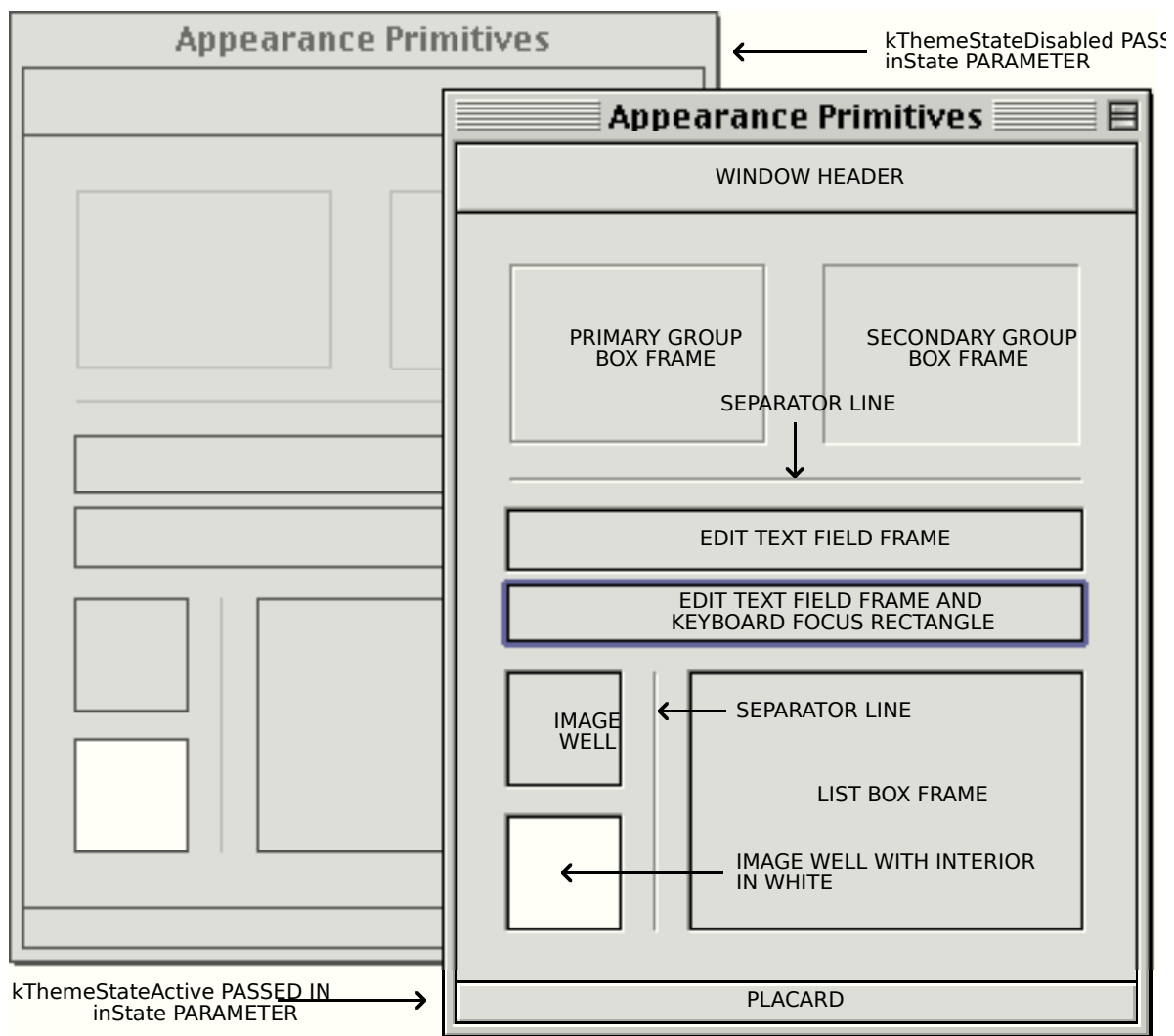


FIG 3 - IMAGES DRAWN WITH OTHER APPEARANCE DRAWING PRIMITIVE

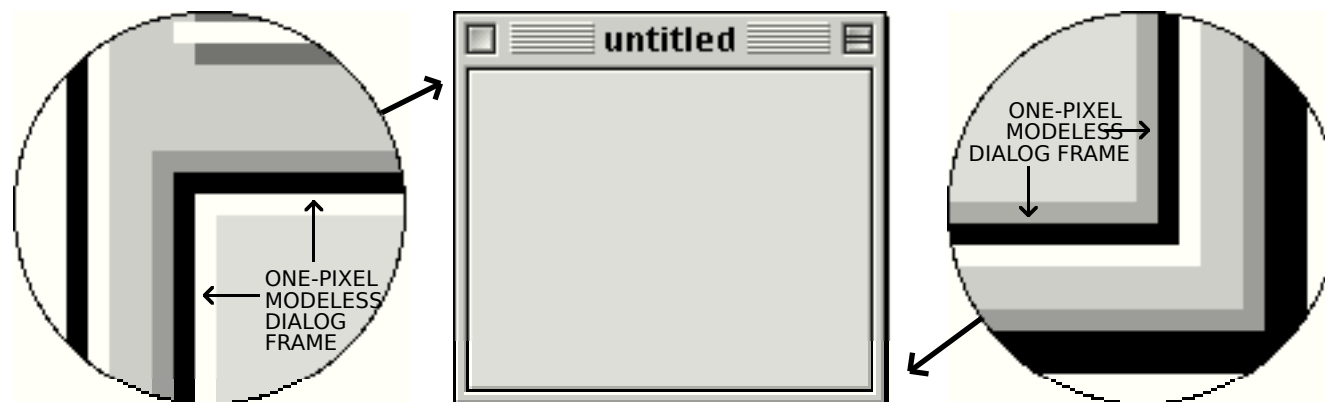


FIG 4 - MODELESS DIALOG FRAME DRAWN WITH APPEARANCE DRAWING PRIMITIVE

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.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.

Drawing in Colours and Patterns Consistent With the Current Theme

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

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

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

Appearance-Compliant Text Colour Constants

Constants of type `ThemeTextColor` may be passed in the `inColor` parameter of the function `SetThemeTextColor` 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
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.

Setting Appearance Cursors

Version 1.1 of the Appearance Manager introduced new functions for setting the cursor shape. These 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-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/patterns you need to draw consistently with the current appearance, 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 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.)

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

<code>gestaltAppearanceAttr</code>	= 'appr'
<code>gestaltAppearanceExists</code>	= 0
<code>gestaltAppearanceCompatMode</code>	= 1
<code>gestaltAppearanceVersion</code>	= 'apvr'

Appearance-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

Appearance-Compliant Text Colour Constants

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

Appearance-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

```
ThemeDrawState : UInt32;
ThemeBrush : SInt16;
ThemeTextColor : SInt16;
```

Routines**Initialising the Appearance Manager**

```
FUNCTION RegisterAppearanceClient: OSStatus;
```

Drawing Appearance Primitives

```
FUNCTION DrawThemeWindowHeader({CONST}VAR inRect: Rect; inState: ThemeDrawState): OSStatus;
FUNCTION DrawThemeWindowListViewHeader({CONST}VAR inRect: Rect;
    inState: ThemeDrawState) : OSStatus;
FUNCTION DrawThemePlacard({CONST}VAR inRect: Rect; inState: ThemeDrawState): OSStatus;
FUNCTION DrawThemeEditTextFrame({CONST}VAR inRect: Rect; inState: ThemeDrawState): OSStatus;
FUNCTION DrawThemeListBoxFrame({CONST}VAR inRect: Rect; inState: ThemeDrawState): OSStatus;
FUNCTION DrawThemeFocusRect({CONST}VAR inRect: Rect; inHasFocus: BOOLEAN): OSStatus;
FUNCTION DrawThemePrimaryGroup({CONST}VAR inRect: Rect; inState: ThemeDrawState): OSStatus;
FUNCTION DrawThemeSecondaryGroup({CONST}VAR inRect: Rect; inState: ThemeDrawState): OSStatus;
```

```

FUNCTION DrawThemeSeparator({CONST}VAR inRect: Rect; inState: ThemeDrawState): OSStatus;
FUNCTION DrawThemeModelessDialogFrame({CONST}VAR inRect: Rect;
    inState: ThemeDrawState): OSStatus;
FUNCTION DrawThemeGenericWell({CONST}VAR inRect: Rect; inState: ThemeDrawState;
    inFillCenter: BOOLEAN): OSStatus;
FUNCTION DrawThemeFocusRegion(inRegion: RgnHandle; inHasFocus: BOOLEAN): OSStatus;
FUNCTION DrawThemeTab({CONST}VAR inRect: Rect; inStyle: ThemeTabStyle;
    inDirection: ThemeTabDirection; labelProc: ThemeTabTitleDrawUPP;
    userData: UInt32): OSStatus;
FUNCTION DrawThemeTabPage({CONST}VAR inRect: Rect; inState: ThemeDrawState): OSStatus;

```

Drawing in Colours and Patterns Consistent With the Current Theme

```

FUNCTION SetThemeWindowBackground(inWindow: WindowPtr; inBrush: ThemeBrush;
    inUpdate: BOOLEAN): OSStatus;
FUNCTION SetThemePen(inBrush: ThemeBrush; inDepth: SInt16;
    inIsColorDevice: BOOLEAN): OSStatus;
FUNCTION SetThemeBackground(inBrush: ThemeBrush; inDepth: SInt16;
    inIsColorDevice: BOOLEAN): OSStatus;
FUNCTION SetThemeTextColor(inColor: ThemeTextColor; inDepth: SInt16;
    inIsColorDevice: BOOLEAN): OSStatus;

```

Saving and Setting the Colour Graphics Port Drawing State

```

FUNCTION NormalizeThemeDrawingState: OSStatus;
FUNCTION GetThemeDrawingState(VAR outState: ThemeDrawingState): OSStatus;
FUNCTION SetThemeDrawingState(inState: ThemeDrawingState; inDisposeNow: BOOLEAN): OSStatus;
FUNCTION DisposeThemeDrawingState(inState: ThemeDrawingState): OSStatus;

```

Setting Appearance Cursors

```

FUNCTION SetThemeCursor(inCursor: ThemeCursor): OSStatus;
FUNCTION SetAnimatedThemeCursor(inCursor: ThemeCursor; inAnimationStep: UInt32): OSStatus;

```

Demonstration Program

```

{
  // AppearanceDemo.p
  //
  // 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 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.
  //
  // }
}

program AppearanceDemo;

//
.....
..... interfaces

```

Version 2.1

uses

```
{ Universal Interfaces. }
Appearance, Devices, Fonts, GestaltEq, Menus, Processes, Sound, ToolUtils, LowMem,
SegLoad;
```

//

..... constants

const

```
rMenubar = 128;
rNewWindow1 = 128;
rNewWindow2 = 129;
mApple = 128;
iAbout = 1;
mFile = 129;
iQuit = 11;
```

```
MAXLONG = $7FFFFFFF;
```

//

..... global variables

var

```
gAppearancePresent : boolean;
gAppearance101present : boolean;
gAppearance110present : boolean;
gInCompatibilityMode : boolean;
gDone : boolean;
gInBackground : boolean;
gWindowPtr1, gWindowPtr2 : WindowPtr;
gPixelFormat : SInt16;
gIsColourDevice : boolean;
gCurrentRect : Rect;
```

//

... main program variables

```
osError : OSErr;
response : SInt32;
menubarHdl : Handle;
menuHdl : MenuHandle;
mainEvent : EventRecord;
```

//

..... routine declarations

```
procedure DoInitManagers; forward;
procedure DoEvents({const} var theEvent : EventRecord); forward;
procedure DoUpdate({const} var theEvent : EventRecord); forward;
procedure DoActivate({const} var theEvent : EventRecord); forward;
procedure DoActivateWindow(theWindowPtr : WindowPtr; becomingActive : boolean); forward;
procedure DoOSEvent({const} var theEvent : EventRecord); forward;
procedure DoDrawAppearancePrimitives(themeState : ThemeDrawState); forward;
procedure DoDrawAppearanceCompliantText(theWindowPtr : WindowPtr;
    themeState : ThemeDrawState); forward;
procedure DoDrawListView(theWindowPtr : WindowPtr); forward;
procedure DoChangeKeyboardFocus(mouseXY : Point); forward;
procedure DoGetDepthAndDevice; forward;
```

```
// DoInitManagers DoInitManagers
```

```
procedure DoInitManagers;
```

```
var
osError : OSErr;
```

```
begin
MaxApplZone;
MoreMasters;
```

```
InitGraf(@qd.thePort);
InitFonts;
InitWindows;
InitMenus;
```



```

        HiliteMenu(0);
        end;

inContent: begin
    if (theWindowPtr <> FrontWindow) then
        begin
            SelectWindow(theWindowPtr);
        end
    else begin
        if (FrontWindow = gWindowPtr2) then
            begin
                SetPort(gWindowPtr2);
                DoChangeKeyBoardFocus(theEvent.where);
            end;
        end;
    end;

inDrag: begin
    if (theWindowPtr <> FrontWindow) then
        begin
            SelectWindow(theWindowPtr);
        end;
        DragWindow(theWindowPtr, theEvent.where, qd.screenBits.bounds);
    end;
end;
    { of case statement }
end;
end;

updateEvt: begin
    DoUpdate(theEvent);
end;

activateEvt: begin
    DoActivate(theEvent);
end;

osEvt: begin
    DoOSEvent(theEvent);
    HiliteMenu(0);
end;

otherwise begin
    end;
end;
    { of case statement }
end;
    { of procedure DoEvents }

//  DoUpdate

procedure DoUpdate({const} var theEvent : EventRecord);
var
    ignoredErr : OSErr;
    theWindowPtr : WindowPtr;

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

    SetPort(theWindowPtr);

    if (theWindowPtr = gWindowPtr2) then
        begin
            if ((gWindowPtr2 = FrontWindow) and (not gInBackground)) then
                begin
                    DoDrawAppearancePrimitives(kThemeStateActive);
                    DoDrawAppearanceCompliantText(theWindowPtr, kThemeStateActive);
                    ignoredErr := DrawThemeFocusRect(gCurrentRect, true);
                end
            else begin
                DoDrawAppearancePrimitives(kThemeStateDisabled);
                DoDrawAppearanceCompliantText(theWindowPtr, kThemeStateDisabled);
            end;
        end;

    if (theWindowPtr = gWindowPtr1) then
        begin
            DoDrawListView(theWindowPtr);
        end;
    end;
end;

```



```

    end;

    EndUpdate(theWindowPtr);
end;
    { of procedure DoUpdate }

// DoActivate
procedure DoActivate({const} var theEvent : EventRecord);
    var
        theWindowPtr : WindowPtr;
        becomingActive : boolean;

    begin
        theWindowPtr := WindowPtr(theEvent.message);
        becomingActive := (BAnd(theEvent.modifiers, activeFlag) = activeFlag);
        DoActivateWindow(theWindowPtr, becomingActive);
    end;
    { of procedure DoActivate }

// DoActivateWindow
procedure DoActivateWindow(theWindowPtr : WindowPtr; becomingActive : boolean);
    var
        ignoredErr : OSErr;

    begin
        if (theWindowPtr = gWindowPtr2) then
            begin
                SetPort(gWindowPtr2);

                DoDrawAppearancePrimitives(integer(becomingActive));
                DoDrawAppearanceCompliantText(theWindowPtr, integer(becomingActive));
                ignoredErr := DrawThemeFocusRect(gCurrentRect, becomingActive);
            end;
        end;
    { of procedure DoActivateWindow }

// DoOSEvent
procedure DoOSEvent({const} var theEvent : EventRecord);
    begin
        case BAnd(BSR(theEvent.message, 24), $000000FF) of

            suspendResumeMessage: begin
                gInBackground := BAnd(theEvent.message, resumeFlag) = 0;
                DoActivateWindow(FrontWindow, not gInBackground);
            end;

            otherwise begin
                end;
        end;
    { of case statement }
end;
    { of procedure DoOSEvent }

// DoDrawAppearancePrimitives
procedure DoDrawAppearancePrimitives(inState : ThemeDrawState);
    var
        ignoredErr : OSErr;
        theRect : Rect;

    begin
        SetRect(theRect, -1, -1, 261, 26);
        ignoredErr := DrawThemeWindowHeader(theRect, inState);

        SetRect(theRect, 20, 46, 119, 115);
        ignoredErr := DrawThemePrimaryGroup(theRect, inState);

        SetRect(theRect, 140, 46, 239, 115);
        ignoredErr := DrawThemeSecondaryGroup(theRect, inState);

        SetRect(theRect, 20, 127, 240, 128);
        ignoredErr := DrawThemeSeparator(theRect, inState);

        SetRect(theRect, 20, 141, 239, 162);
        ignoredErr := DrawThemeEditTextFrame(theRect, inState);

        SetRect(theRect, 20, 169, 239, 190);
    end;

```



```

    MoveTo(theRect.left, a - 5);
    DrawString(' List View Background List View Sort Column');
    end;
end;
{ of procedure DoDrawListView }

// DoChangeKeyBoardFocus
procedure DoChangeKeyBoardFocus(mouseXY : Point);
var
  ignoredErr : OSErr;
  edit1Rect, edit2Rect, listRec : Rect;

begin
  ignoredErr := DrawThemeFocusRect(gCurrentRect, false);
  ignoredErr := DrawThemeEditTextFrame(gCurrentRect, kThemeStateActive);

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

  GlobalToLocal(mouseXY);

  if PtInRect(mouseXY, edit1Rect) then
    begin
      SetRect(gCurrentRect, 20, 141, 239, 162);
    end
  else if PtInRect(mouseXY, edit2Rect) then
    begin
      SetRect(gCurrentRect, 20, 169, 239, 190);
    end
  else if PtInRect(mouseXY, listRec) then
    begin
      SetRect(gCurrentRect, 90, 203, 239, 300);
    end;

  ignoredErr := DrawThemeFocusRect(gCurrentRect, true);
end;
{ of procedure DoChangeKeyBoardFocus }

// DoGetDepthAndDevice
procedure DoGetDepthAndDevice;
var
  deviceHdl : GDHandle;

begin
  deviceHdl := LMGetMainDevice;
  gPixelDepth := deviceHdl^^.gdPMap^^.pixelSize;
  if (BitTst(Ptr(@deviceHdl^^.gdFlags), gdDevType)) then
    begin
      glsColourDevice := true;
    end;
end;
{ of procedure DoGetDepthAndDevice }

// main program
begin
  // ..... initialise
  global variables

  gAppearancePresent := false;
  gAppearance101present := false;
  gAppearance110present := false;
  gInCompatibilityMode := false;

  glsColourDevice := false;

  //
  .....
  initialise managers

  DoInitManagers;

  // ..... check for Appearance and functions, compatibility mode, Appearance version
  osError := Gestalt(gestaltAppearanceAttr, response);

```

Version 2.1

```
if ((osError = noErr) and BitTst(response, gestaltAppearanceExists)) then
  begin
    gAppearancePresent := true;

    if (BitTst(response, gestaltAppearanceCompatMode)) then
      begin
        gInCompatibilityMode := true;
      end;

    osError := Gestalt(gestaltAppearanceVersion, response);

    if (response = $00000101) then
      begin
        gAppearance101present := true;
      end
    else if (response = $00000110) then
      begin
        gAppearance110present := true;
      end
    end
  else begin
    SysBeep(10);
    ExitToShell;
  end;

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

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

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

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

  gWindowPtr1 := GetNewCWindow(rNewWindow1, nil, WindowPtr(-1));
  if (gWindowPtr1 = nil) then
    begin
      ExitToShell;
    end;

  SetPort(gWindowPtr1);
  TextSize(10);
  ShowWindow(gWindowPtr1);

  gWindowPtr2 := GetNewCWindow(rNewWindow2, nil, WindowPtr(-1));
  if (gWindowPtr2 = nil) then
    begin
      ExitToShell;
    end;

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

  // ..... set Appearance-compliant colour/pattern for second window
  osError := 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 (not gDone) do
  begin
    if (WaitNextEvent(everyEvent, mainEvent, MAXLONG, nil)) then
      begin
        DoEvents(mainEvent);
      end;
    end;
  end.
  { of main program. }

// .....
```

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 "Appearance-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.

constants

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.

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.

dolnitManagers

DolnitManagers 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 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.

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 routine DoChangeKeyboardFocus is called. That routine 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 routine, 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 routines 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 "Appearance-Compliant List View" window, an application-defined routine 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 routines for drawing the primitives and the window header text, together with the Appearance routine which draws and erases the keyboard focus rectangle, are called. To eliminate the necessity for if/then/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 " Appearance-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.

DoDrawThemePrimitives

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.

If "Drawing With Primitives" is the active window, SetThemeTextColor is called with the kThemeActiveWindowHeaderTextColor text colour constant passed in the first parameter. 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. `SetThemeTextColour` 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.

main program block

`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, 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 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.