6

THE APPEARANCE MANAGER

Demonstration Program: Appearance

History

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 and Chapter 4. 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 Chapters 7 and 14. Accordingly, as a preparation for what is to come, this chapter now formally introduces the Appearance Manager, a component of the system software which, on Mac OS 8/9, represented 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, dialogs, 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.

On Mac OS 8/9, in 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, dialogs, alerts, menus, controls, etc.), together with a highlight colour (for selected text) and a variation colour (for menus and controls). (As of Mac OS 9.1, Platinum remains the only appearance ever provided by Apple. It is by now certain that this situation will never change.)
- 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 spoke 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 Mac OS 8/9 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. 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 influence of the Appearance Manager is evident to a greater or lesser extent in many chapters of this book and in all of the associated demonstration programs. Amongst other things, it ensures consistency in the appearance of the standard human interface elements on both Mac OS 8/9 and Mac OS X. It also provides the means to:

- Ensure that the appearance of your application's custom human interface elements (if any) is consistent with the Platinum and Aqua "look".
- Draw anti-aliased text on Mac OS X.

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

Carbon fully supports the Appearance Manager.

New Definition Functions — Mac OS 8/9

To provide a system-wide coordination of look and behaviour on Mac OS 8/9, 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.

Colours, Patterns, and Appearance Primitives

The Appearance Manager provides **Appearance primitives**, and the means to set the colours and patterns, needed to draw consistently in the Platininum appearance on Mac OS 8/9 and with the Aqua "look" on Mac OS X. Using these drawing primitives, colours, and patterns makes it easier to create visual entities and custom human interface elements that are consistent with the Platinum appearance and Aqua "look".

Drawing Appearance Primitives

As will become apparent at Chapters 7 and 14, most of the Appearance primitives relate to certain controls. The 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 visual representation of that control.

Your application might use these primitives to, for example, draw an image of a placard, window header, edit text field frame, etc., when you don't want to use a control.

The following are examples of functions that draw Appearance primitives:

Function	*Description*
`DrawThemePrimaryGroup`	Draws a primary group box frame.
`DrawThemeSecondaryGroup`	Draws a secondary group box frame.
`DrawThemeSeparator`	Draws a separator line. The orientation of the rectangle determines how 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.
`DrawThemePlacard`	Draws a placard.
`DrawThemeEditTextFrame`	Draws an edit text field frame. 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. You should not use these frames for items other than edit text fields.
`DrawThemeListBoxFrame`	Draws a list box frame. 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.
`DrawThemeGenericWell`	Draws an image well frame. You can specify that the centre of the well be filled with white (Mac OS 8/9).
`DrawThemeFocusRegion`	Draws or erases a focus ring around a specified region.
`DrawThemeTabPane`	Draws a tab-pane.
`DrawThemeTab`	Draws a tab.

Fig 2 shows examples of images drawn in the active mode using the Appearance primitives.

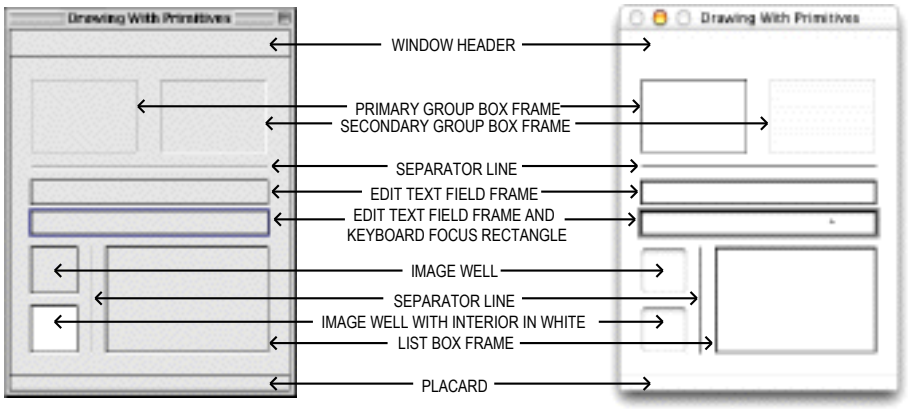

FIG 2 - IMAGES DRAWN WITH APPEARANCE DRAWING PRIMITIVES

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

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

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 Platinum Appearance and Aqua "Look"

The following functions are those used to draw using colours/**patterns** consistent with the Platinum appearance and Aqua "look". (Patterns are explained at Chapter 11.) The reference to colours *and* patterns reflects the fact that either a colour or a pattern may be used for the drawing.

Function	*Description*
`SetThemeWindowBackground`	Sets the 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 Brush Type Constants, below.
`SetThemeBackground`	Sets an element's background colour/pattern to comply with the Platinum appearance/Aqua "look". 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 Brush Type Constants, below.
`SetThemePen`	Sets an element's pen pattern or colour to comply with the Platinum appearance/Aqua "look". 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 Brush Type Constants, below.
`SetThemeTextColor`	Sets an element's foreground colour for drawing text to comply with the Platinum appearance/Aqua "look". See also Text Colour Constants, below.

Brush Type Constants

The following are examples of constants, of type `ThemeBrush`, which may be passed in the `inBrush` parameter of calls to `SetThemeWindowBackground`, `SetThemeBackground`, and `SetThemePen` to specify 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's background colour/ pattern.
kThemeBrushDialogBackgroundInactive	An inactive dialog's background colour or pattern.
kThemeBrushAlertBackgroundActive	An active alert's background colour/pattern.
kThemeBrushAlertBackgroundInactive	An inactive alert's background colour/pattern.
kThemeBrushModelessDialogBackgroundActive	An active modeless dialog's background colour/pattern.
kThemeBrushModelessDialogBackgroundInactive	An inactive modeless dialog'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. (Applicable on Mac OS 8/9 only.)
kThemeBrushListViewBackground	The background colour/pattern of a list view column that is not being sorted upon. (Applicable on Mac OS 8/9 only.)
kThemeBrushListViewSeparator	A list view separator's colour/pattern. (Applicable on Mac OS 8/9 only.)
kThemeBrushDocumentWindowBackground	A document window's background colour/pattern.

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.
kThemeListViewTextColor	Text colour for list view. (Applicable on Mac OS 8/9 only.)

Appearance Manager Text

The Appearance Manager function UseThemeFont may be used to set the font for the current graphics port.

Text drawn on Mac OS X using QuickDraw functions such as DrawString is not entirely satisfactory. You should therefore use the Appearance Manager function DrawThemeTextBox to draw text when your application is running on Mac OS X.

You pass a value of type ThemeFontID in the inFontID parameter of UseThemeFont and DrawThemeTextBox. The principal relevant constants are as follows:

Constant	Font on Mac OS 8/9	Font on Mac OS X
kThemeSystemFont	As set in Appearance control panel.	Lucida Grande Regular 13pt
kThemeEmphasizedSystemFont	System font, as set in Appearance control panel.	Lucida Grande Bold 13pt
kThemeSmallSystemFont	As set in Appearance control panel.	Lucida Grande Regular 11pt
kThemeSmallEmphasizedSystemFont	Small system font, as set in Appearance control panel, bold.	Lucida Grande Bold 11pt
kThemeApplicationFont	Geneva 12pt.	Lucida Grande Regular 13pt
kThemeLabelFont	System font, as set in Appearance control panel.	Lucida Grande Regular 10pt

Saving and Setting the Graphics Port Drawing State

Chapter 12 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. The associated functions are as follows:

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

Cursor Setting

The Appearance Manager introduced the following cursor-setting functions, the uses of which are addressed at Chapter 13:

Constant	Description
SetThemeCursor	Sets the cursor.
SetAnimatedThemeCursor	Sets an animated cursor.

Getting Menu Bar Height

The Appearance Manager introduced the function `GetThemeMenuBarHeight`. In most instances, the value returned by this function and `GetMBarHeight` are the same. However, when the menu bar is hidden, `GetMBarHeight` produces a value of `0`, whereas `GetThemeMenuBarHeight` still returns the height of the (hidden) menu bar.

Appearance Manager Apple Events

On Mac OS 8/9, 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. Your application is advised of font changes via **Appearance Manager Apple events**. Appearance Manager Apple events are addressed at Chapter 10.

> ***Carbon Note***
>
> Prior to CarbonLib 1.1, it was necessary to call `RegisterAppearanceClient` at program launch in order for your application to receive Appearance Manager Apple events. However, in CarbonLib 1.1 and later, the CarbonLib initialisation routine calls `RegisterAppearanceClient` on behalf of your application, and there is thus no requirement for your application to call this function.

Constants

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
kThemeBrushBlack                            = -1
kThemeBrushWhite                            = -2
```

Theme-Compliant Text Colour Constants

```
kThemeTextColorWindowHeaderActive           = 7
kThemeTextColorWindowHeaderInactive         = 8
kThemeTextColorPlacardActive                = 9
kThemeTextColorPlacardInactive              = 10
kThemeTextColorPlacardPressed               = 11
kThemeTextColorListView                     = 22
kThemeTextColorBlack                        = -1
kThemeTextColorWhite                        = -2
```

Theme-Compliant Draw State Constants (For Primitives)

```
kThemeStateInactive                         = 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
```

Font Constants

```
kThemeSystemFont                            = 0
kThemeSmallSystemFont                       = 1
kThemeSmallEmphasizedSystemFont             = 2
kThemeViewsFont                             = 3
kThemeEmphasizedSystemFont                  = 4
kThemeApplicationFont                       = 5
kThemeLabelFont                             = 6
```

```
kThemeCurrentPortFont                       = 200
```

Data Types

```
typedef UInt32 ThemeDrawState;
typedef SInt16 ThemeBrush;
typedef SInt16 ThemeTextColor;
typedef UInt32 ThemeCursor;
typedef struct OpaqueThemeDrawingState* ThemeDrawingState;
```

Functions

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 Platinum/Aqua

```
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);
```

Setting and Getting the Graphics Port Font

```
OSStatus  UseThemeFont(ThemeFontID inFontID,ScriptCode inScript);
OSStatus  GetThemeFont(ThemeFontID inFontID,ScriptCode inScript,Str255 outFontName,
           SInt16 *outFontSize,Style *outStyle);
```

Drawing Text

```
OSStatus  DrawThemeTextBox(CFStringRef inString,ThemeFontID inFontID,ThemeDrawState inState,
          Boolean inWrapToWidth,const Rect *inBoundingBox,SInt16 inJust,void *inContext);
OSStatus  TruncateThemeText(CFMutableStringRef inString,ThemeFontID inFontID,
          ThemeDrawState inState,SInt16 inPixelWidthLimit,TruncCode inTruncWhere,
          Boolean *outTruncated);
OSStatus  GetThemeTextDimensions(CFStringRef inString,ThemeFontID inFontID,
          ThemeDrawState inState,Boolean inWrapToWidth,Point *ioBounds,SInt16 *outBaseline);
OSStatus  GetThemeTextShadowOutset(ThemeFontID inFontID,ThemeDrawState inState,
          Rect * outOutset);
```

Saving and Setting the 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);
```

Getting Menu Bar Height

```
OSStatus  GetThemeMenuBarHeight(SInt16 *outHeight);
```

Demonstration Program Appearance Listing

```
// ****************************************************************************************
// Appearance.c                                                          CLASSIC EVENT MODEL
// ****************************************************************************************
//
// This program opens two kWindowDocumentProc windows containing:
//
// •  In the first window:
//
//    •  On Mac OS 8/9, a theme-compliant list view.
//
//    •  On Mac OS X, some text drawn with Appearance Manager functions.
//
// •  In the second window, various images drawn with Appearance primitives and, on Mac OS
//    8/9, text drawn in the window header 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 utilises the following resources:
//
// •  A 'plst' resource.
//
// •  An 'MBAR' resource, and 'MENU' resources for Apple, File, and Edit menus (preload,
//    non-purgeable).
//
// •  Two 'WIND' resources (purgeable) (initially not visible).
//
// •  A 'STR#' list resource (purgeable) containing text drawn in the first window.
//
// •  'hrct' and 'hwin' resources (both purgeable), which provide help balloons describing the
//     contents of the windows (Mac OS 8/9).
//
// •  A 'SIZE' resource with the acceptSuspendResumeEvents,  canBackground,
//    doesActivateOnFGSwitch, and isHighLevelEventAware flags set.
//
// ****************************************************************************************

// ..... includes

#include <Carbon.h>

// ..... defines

#define rMenubar          128
#define rNewWindow1       128
#define rNewWindow2       129
#define mAppleApplication 128
#define  iAbout           1
#define mFile             129
#define  iQuit            12
#define sDescription      128
#define MAX_UINT32        0xFFFFFFFF

// ..... global variables

Boolean   gRunningOnX = false;
Boolean   gDone;
Boolean   gInBackground;
WindowRef gWindowRef1, gWindowRef2;
SInt16    gPixelDepth;
Boolean   gIsColourDevice = false;
Rect      gCurrentRect;
Rect      gEditText1Rect = { 141, 20, 162, 239 };
Rect      gEditText2Rect = { 169, 20, 190, 239 };
```

```
Rect      gListBoxRect   = { 203, 90, 300, 239 };

// ..... function prototypes

void  main                          (void);
void  doPreliminaries               (void);
OSErr quitAppEventHandler           (AppleEvent *,AppleEvent *,SInt32);
void  doEvents                      (EventRecord *);
void  doUpdate                      (EventRecord *);
void  doActivate                    (EventRecord *);
void  doActivateWindow              (WindowRef,Boolean);
void  doOSEvent                     (EventRecord *);
void  doDrawAppearancePrimitives    (ThemeDrawState);
void  doDrawThemeCompliantTextOn9 (WindowRef,ThemeDrawState);
void  doDrawListViewOn9             (WindowRef);
void  doDrawThemeTextOnX            (WindowRef,ThemeDrawState);
void  doChangeKeyBoardFocus         (Point);
void  doGetDepthAndDevice           (void);

// ********************************************************************************** main

void  main(void)
{
  MenuBarHandle menubarHdl;
  SInt32        response;
  MenuRef       menuRef;
  SInt16        fontNum;
  EventRecord   EventStructure;

  // ..... do preliminaries

  doPreliminaries();

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

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

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

    gRunningOnX = true;
  }

  // ..... open first window, set font and font size, show window

  if(!(gWindowRef1 = GetNewCWindow(rNewWindow1,NULL,(WindowRef)-1)))
    ExitToShell();

  SetPortWindowPort(gWindowRef1);
  if(!gRunningOnX)
    UseThemeFont(kThemeSmallSystemFont,smSystemScript);
  else
  {
    GetFNum("\pAmerican Typewriter",&fontNum);
    TextFont(fontNum);
    TextSize(11);
  }
```

```
  if(!gRunningOnX)
    SetWTitle(gWindowRef1,"\pList Views");
  else
    SetWTitle(gWindowRef1,"\pTheme Text");

  ShowWindow(gWindowRef1);

  // ……..…… open second window, set font, set background colour/pattern, show window


  if(!(gWindowRef2 = GetNewCWindow(rNewWindow2,NULL,(WindowRef)-1)))
    ExitToShell();

  SetPortWindowPort(gWindowRef2);
  UseThemeFont(kThemeSmallSystemFont,smSystemScript);

  SetThemeWindowBackground(gWindowRef2,kThemeBrushDialogBackgroundActive,false);

  ShowWindow(gWindowRef2);

  // ……..…… get pixel depth and whether colour device for certain Appearance Manager functions

  if(!gRunningOnX)
    doGetDepthAndDevice();

// ……..…… set top edit text field rectangle as target for initial keyboard focus frame

  gCurrentRect = gEditText1Rect;

  // ……..…… enter eventLoop

  gDone = false;

  while(!gDone)
  {
    if(WaitNextEvent(everyEvent,&EventStructure,MAX_UINT32,NULL))
      doEvents(&EventStructure);
  }
}

// ********************************************************************** doPreliminaries

void  doPreliminaries(void)
{
  OSErr osError;

  MoreMasterPointers(32);
  InitCursor();
  FlushEvents(everyEvent,0);

  osError = AEInstallEventHandler(kCoreEventClass,kAEQuitApplication,
                            NewAEEventHandlerUPP((AEEventHandlerProcPtr) quitAppEventHandler),
                            0L,false);
  if(osError != noErr)
    ExitToShell();
}

// ********************************************************************** doQuitAppEvent

OSErr  quitAppEventHandler(AppleEvent *appEvent,AppleEvent *reply,SInt32 handlerRefcon)
{
  OSErr    osError;
  DescType returnedType;
  Size     actualSize;

  osError = AEGetAttributePtr(appEvent,keyMissedKeywordAttr,typeWildCard,&returnedType,NULL,0,
                              &actualSize);
```

```
  if(osError == errAEDescNotFound)
  {
    gDone = true;
    osError = noErr;
  }
  else if(osError == noErr)
    osError = errAEParamMissed;

  return osError;
}

// ********************************************************************************** doEvents

void  doEvents(EventRecord *eventStrucPtr)
{
  SInt32         menuChoice;
  MenuID         menuID;
  MenuItemIndex  menuItem;
  WindowPartCode partCode;
  WindowRef      windowRef;

  switch(eventStrucPtr->what)
  {
    case kHighLevelEvent:
      AEProcessAppleEvent(eventStrucPtr);
      break;

    case keyDown:
      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,&windowRef))
      {
        switch(partCode)
        {
          case inMenuBar:
            menuChoice = MenuSelect(eventStrucPtr->where);
            menuID = HiWord(menuChoice);
            menuItem = LoWord(menuChoice);

            if(menuID == 0)
              return;

            switch(menuID)
            {
              case mAppleApplication:
                if(menuItem == iAbout)
                {
                  SysBeep(10);
                  HiliteMenu(0);
                }
                break;

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

          case inContent:
```

```
            if(windowRef != FrontWindow())
              SelectWindow(windowRef);
            else
            {
              if(FrontWindow() == gWindowRef2)
              {
                SetPortWindowPort(gWindowRef2);
                doChangeKeyBoardFocus(eventStrucPtr->where);
              }
            }
            break;

          case inDrag:
            DragWindow(windowRef,eventStrucPtr->where,NULL);
            break;
        }
      }
      break;

    case updateEvt:
      doUpdate(eventStrucPtr);
      break;

    case activateEvt:
      doActivate(eventStrucPtr);
      break;

    case osEvt:
      doOSEvent(eventStrucPtr);
      break;
  }
}

// ********************************************************************************** doUpdate

void  doUpdate(EventRecord *eventStrucPtr)
{
  WindowRef windowRef;

  windowRef = (WindowRef) eventStrucPtr->message;

  BeginUpdate(windowRef);

  SetPortWindowPort(windowRef);

  if(windowRef == gWindowRef2)
  {
    if(gWindowRef2 == FrontWindow() && !gInBackground)
    {
      doDrawAppearancePrimitives(kThemeStateActive);
      DrawThemeFocusRect(&gCurrentRect,true);
      if(!gRunningOnX)
        doDrawThemeCompliantTextOn9(windowRef,kThemeStateActive);
    }
    else
    {
      doDrawAppearancePrimitives(kThemeStateInactive);
      if(!gRunningOnX)
        doDrawThemeCompliantTextOn9(windowRef,kThemeStateInactive);
    }
  }

  if(windowRef == gWindowRef1)
  {
    if(!gRunningOnX)
      doDrawListViewOn9(windowRef);
  }

  EndUpdate(windowRef);
```

```
}

// ********************************************************************************** doActivate

void  doActivate(EventRecord *eventStrucPtr)
{
  WindowRef windowRef;
  Boolean   becomingActive;

  windowRef = (WindowRef) eventStrucPtr->message;
  becomingActive = ((eventStrucPtr->modifiers & activeFlag) == activeFlag);
  doActivateWindow(windowRef,becomingActive);
}

// ****************************************************************************** doActivateWindow

void  doActivateWindow(WindowRef windowRef,Boolean becomingActive)
{
  if(windowRef == gWindowRef2)
  {
    SetPortWindowPort(gWindowRef2);
    doDrawAppearancePrimitives(becomingActive);
    DrawThemeFocusRect(&gCurrentRect,becomingActive);

    if(!gRunningOnX)
      doDrawThemeCompliantTextOn9(windowRef,becomingActive);

  }
  else if(windowRef == gWindowRef1 && gRunningOnX)
  {
    SetPortWindowPort(gWindowRef1);
    doDrawThemeTextOnX(windowRef,becomingActive);
  }
}

// ************************************************************************************ doOSEvent

void  doOSEvent(EventRecord *eventStrucPtr)
{
  switch((eventStrucPtr->message >> 24) & 0x000000FF)
  {
    case suspendResumeMessage:
      if((eventStrucPtr->message & resumeFlag) == 1)
      {
        SetThemeCursor(kThemeArrowCursor);
        gInBackground = false;
      }
      else
        gInBackground = true;
      break;
  }
}

// ************************************************************** doDrawAppearancePrimitives

void  doDrawAppearancePrimitives(ThemeDrawState inState)
{
  Rect theRect;

  if(gRunningOnX)
  {
    GetWindowPortBounds(gWindowRef2,&theRect);
    EraseRect(&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);

  DrawThemeEditTextFrame(&gEditText1Rect,inState);

  DrawThemeEditTextFrame(&gEditText2Rect,inState);

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

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

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

  DrawThemeListBoxFrame(&gListBoxRect,inState);

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

// ************************************************************ doDrawThemeCompliantTextOn9

void  doDrawThemeCompliantTextOn9(WindowRef windowRef,ThemeDrawState inState)
{
  SInt16 windowWidth, stringWidth;
  Rect   portRect;
  Str255 message = "\pBalloon help is available";

  if(inState == kThemeStateActive)
    SetThemeTextColor(kThemeTextColorWindowHeaderActive,gPixelDepth,gIsColourDevice);
  else
    SetThemeTextColor(kThemeTextColorWindowHeaderInactive,gPixelDepth,gIsColourDevice);

  GetWindowPortBounds(windowRef,&portRect);
  windowWidth = portRect.right - portRect.left;
  stringWidth = StringWidth(message);
  MoveTo((windowWidth / 2) - (stringWidth / 2), 17);
  DrawString(message);
}

// ************************************************************************ doDrawListViewOn9

void  doDrawListViewOn9(WindowRef windowRef)
{
  Rect   theRect;
  SInt16 a;

  GetWindowPortBounds(windowRef,&theRect);

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

  theRect.left += 130;

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

  SetThemePen(kThemeBrushListViewSeparator,gPixelDepth,gIsColourDevice);

  GetWindowPortBounds(windowRef,&theRect);

  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");
  }
}

// ********************************************************************** doDrawThemeTextOnX

void  doDrawThemeTextOnX(WindowRef windowRef,ThemeDrawState inState)
{
  Rect        portRect;
  Str255      theString;
  CFStringRef stringRef;

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

  if(inState == kThemeStateActive)
    TextMode(srcOr);
  else
    TextMode(grayishTextOr);

  SetRect(&portRect,portRect.left,30,portRect.right,50);
  DrawThemeTextBox(CFSTR("System Font"),kThemeSystemFont,inState,true,
                   &portRect,teJustCenter,NULL);
  SetRect(&portRect,portRect.left,60,portRect.right,80);
  DrawThemeTextBox(CFSTR("Emphasized System Font"),kThemeEmphasizedSystemFont,inState,true,
                   &portRect,teJustCenter,NULL);
  SetRect(&portRect,portRect.left,90,portRect.right,105);
  DrawThemeTextBox(CFSTR("Small System Font"),kThemeSmallSystemFont,inState,true,
                   &portRect,teJustCenter,NULL);
  SetRect(&portRect,portRect.left,120,portRect.right,135);
  DrawThemeTextBox(CFSTR("Small Emphasized System Font"),kThemeSmallEmphasizedSystemFont,
                   inState,true,&portRect,teJustCenter,NULL);
  SetRect(&portRect,portRect.left,150,portRect.right,170);
  DrawThemeTextBox(CFSTR("Application Font"),kThemeApplicationFont,inState,true,
                   &portRect,teJustCenter,NULL);
  SetRect(&portRect,portRect.left,180,portRect.right,190);
  DrawThemeTextBox(CFSTR("Label Font"),kThemeLabelFont,inState,true,
                   &portRect,teJustCenter,NULL);

  GetIndString(theString,sDescription,1);
  stringRef = CFStringCreateWithPascalString(NULL,theString,CFStringGetSystemEncoding());
  SetRect(&portRect,portRect.left + 20,210,portRect.right - 20,300);
  DrawThemeTextBox(stringRef,kThemeCurrentPortFont,inState,true,&portRect,teJustCenter,NULL);

  if(stringRef != NULL)
    CFRelease(stringRef);
}

// ********************************************************************** doChangeKeyBoardFocus

void  doChangeKeyBoardFocus(Point mouseXY)
{
  Rect portRect;

  DrawThemeFocusRect(&gCurrentRect,false);
  DrawThemeEditTextFrame(&gCurrentRect,kThemeStateActive);
  SetPortWindowPort(gWindowRef2);
  GlobalToLocal(&mouseXY);
```

```
  if(PtInRect(mouseXY,&gEditText1Rect))
    gCurrentRect = gEditText1Rect;
  else if(PtInRect(mouseXY,&gEditText2Rect))
    gCurrentRect = gEditText2Rect;
  else if(PtInRect(mouseXY,&gListBoxRect))
    gCurrentRect = gListBoxRect;

  GetWindowPortBounds(gWindowRef2,&portRect);
  InvalWindowRect(gWindowRef2,&portRect);
}

// ************************************************************************** doGetDepthAndDevice

void doGetDepthAndDevice(void)
{
  GDHandle deviceHdl;

  deviceHdl = GetMainDevice();
  gPixelDepth = (*(*deviceHdl)->gdPMap)->pixelSize;
  if(((1 << gdDevType) & (*deviceHdl)->gdFlags) != 0)
    gIsColourDevice = true;
}

// ****************************************************************************************************
```

Demonstration Program Appearance Comments

When this program is run, the user should:

- With the "Drawing With Primitives" window frontmost, click in the edit text 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. On Mac OS 8/9, note also that there is no change to the appearance of the content region of the "List Views" window. On Mac OS X, note the changed appearance of the text in the "Theme Text" 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. Click on the "Theme Text" window to make it active and note the changed appearance of the text.

- On Mac OS 8/9, 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 "List Views" (when active), noting the descriptions in the balloons.

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

Global Variables

gWindowRef1 and gWindowRef2 will be assigned references to window objects.

gPixelDepth will be assigned the pixel depth of the main device (screen). 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 Manager functions.

gCurrentRect will be assigned the rectangle which is to be the current target for the keyboard focus frame. gEditText1Rect, gEditText2Rect, and gListBoxRect are assigned the rectangles for the two edit text frames and the list box frame.

main

After each window is created, its graphics port is set as the current port before the port's font is set. For the first window's graphics port, if the program is running on Mac OS 8/9, the Appearance Manager function UseThemeFont is called to set the font to the small system font. Otherwise the Font Manager function GetFNum is called to get the font number for American Typewriter, which is then set as the port's font, at 11 points, by the QuickDraw functions TextFont and TextSize. For the second window's graphics port, UseThemeFont is called to set the font to the small system font.

If the program is running on OS 8/9, SetThemeWindowBackground is called to set 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 false is passed in the third parameter, the content region of the window is not invalidated. (Passing true in this instance is not appropriate because the window is not yet showing.)

If the program is running on OS 8/9, doGetDepthAndDevice is called to determine the current pixel depth of the graphics port, and whether the current graphics device is a colour device, and assign the results to the global variables gPixelDepth and gIsColourDevice. These values are required by certain Appearance Manager functions which, in this program, are not called if the program is running on Mac OS X.

The next line sets the initial target for the keyboard focus frame. This is the rectangle used by the first edit text field frame.

doEvents

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

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

However, if the window is the front window, and if that window is the "Drawing With Primitives" window, that window's graphics port is set as the current graphics port for drawing, and 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, if that window is currently the front window, and if the application is not currently in the background:

- Functions are called to draw the primitives and, on Mac OS 8/9 only, 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, on Mac OS 8/9 only, 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 the update event relates is the "List Views" window, doDrawListViewOn9 is called to draw the window's content area. 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, functions are called to draw the primitives and, on Mac OS 8/9 only, the window header text. In addition, an Appearance Manager function which draws and erases the keyboard focus rectangle is called. The value passed in the becomingActive parameter of these calls ensures ensure that, firstly, the primitives and text are drawn in the appropriate mode and, secondly, 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.

If the activate event is for the first window and the program is running on Mac OS X, doDrawThemeTextOnX is called to draw some text in the window in either the active or inactive mode.

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.

doDrawThemeCompliantTextOn9

doDrawThemeCompliantTextOn9, which is called only if the program is running on Mac OS 8/9, 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 Manager 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 Platinum 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 four 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.

doDrawListViewOn9

doDrawListViewOn9, which is called only if the program is running on Mac OS 8/9, draws a 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.

doDrawThemeTextOnX

doDrawThemeTextOnX is called only if the program is running on Mac OS X. It draws anti-aliased text in the first window.

GetWindowPortBounds is called to copy the port rectangle to a local variable of type Rect. EraseRect is then called to erase the port rectangle, a necessary precursor to drawing over existing anti-aliased text on Mac OS X using the Appearance Manager function DrawThemeTextBox.

As was done in the function doDrawThemeCompliantTextOn9, SetThemeTextColor could be used here to set the text colour according to the value received in the inState formal parameter. However, in this instance the alternative of calling TextMode is used. The so-called transfer modes passed in the calls to TextMode are explained at Chapter 12. srcOr is the default transfer mode for text, and causes the colour of the glyph (the visual representation of a character) to be determined by the graphics port's foreground colour. The non-standard mode grayishTextOr is used to draw text in the deactivated state.

Before each call to DrawThemeTextBox, SetRect is called to adjust the top and bottom fields of the Rect variable portRect. This controls the vertical positioning of the text in the window, being passed in DrawThemeTextBox's inBoundingBox parameter. teJustCenter is passed in DrawThemeTextBox's inJust parameter to cause the text to be centre-justified within the rectangle. The Appearance Manager constants passed in the inFontID parameter determine the size and style of the drawn text.

At the last block, a string is retrieved from a 'STR#' Resource. After being converted to a CFString, that string is drawn by DrawThemeTextBox in the bottom of the window. Note that kThemeCurrentPort passed in the inFontID parameter so as to cause the text to be drawn using the window's graphics port font, which was set in main.

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 Manager 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 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.

The call to InvalWindowRect ensures that the keyboard focus frame will be drawn by the call to DrawThemeFocusRect in the function doUpdate.

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 the Appearance Manager functions SetThemeTextColor, SetThemeBackground, and SetThemePen.