INSIDE CARBON

Programming With the Appearance Manager

For Appearance Manager 1.1

April 21, 1999
Technical Publications

Contents

Figures, Tables, and Listings

C H A P T E R 1

Introduction

This document describes how your program can use the Appearance Manager, through Appearance Manager version 1.1.

The Appearance Manager coordinates the look of human interface elements on the Mac OS and provides the underlying support for appearances and themes. **Appearances** unify the look of human interface elements in your program and across the system—including alert icons, controls, background colors, dialog boxes, menus, windows, and state transitions—thus giving the user a consistent experience. **Themes** bundle additional user preferences regarding such interface aspects as sounds, desktop pictures or patterns, and system fonts.

You can use the Appearance Manager to adapt any nonstandard interface elements in your program to the same coordinated look as the rest of the Mac OS. The Appearance Manager also provides many standard human interface elements, such as focus rings and group boxes, that can eliminate the need to create and maintain your own custom solutions.

Many programs exclusively use standard, system-defined interface elements. If yours is one of these, you may need only to register your program with the Appearance Manager and be prepared to respond to Appearance Manager Apple events in order to coordinate with the systemwide look, that is, to be **theme-compliant**. However, if your program uses any custom interface elements, you may need to use other features of the Appearance Manager for your program to be theme-compliant.

If your program has a user interface, you should read this document to learn how to give your program a look consistent with the system and other Mac OS programs. Documentation on related Mac OS human interface technologies is available at

<http://developer.apple.com/techpubs/macos8/HumanInterfaceToolbox/humaninterfacetoolbox.html>

The following chapters describe the Appearance Manager:

- "About the Appearance Manager" (page 13) introduces the Appearance Manager and its capabilities.
- "Using the Appearance Manager" (page 39) provides examples of how your program can use the Appearance Manager.
- "Appearance Manager Reference" (page 65) describes the complete Appearance Manager application programming interface (API) through version 1.1.
- "Document Version History" (page 239) provides a history of corrections and other changes to this document.

About the Appearance Manager

Contents

The Appearance Manager coordinates the look of standard Mac OS human interface elements and helps you adapt your custom interface elements to the coordinated systemwide look, that is, to be theme-compliant.

This chapter describes the Appearance Manager, through version 1.1, in the following sections:

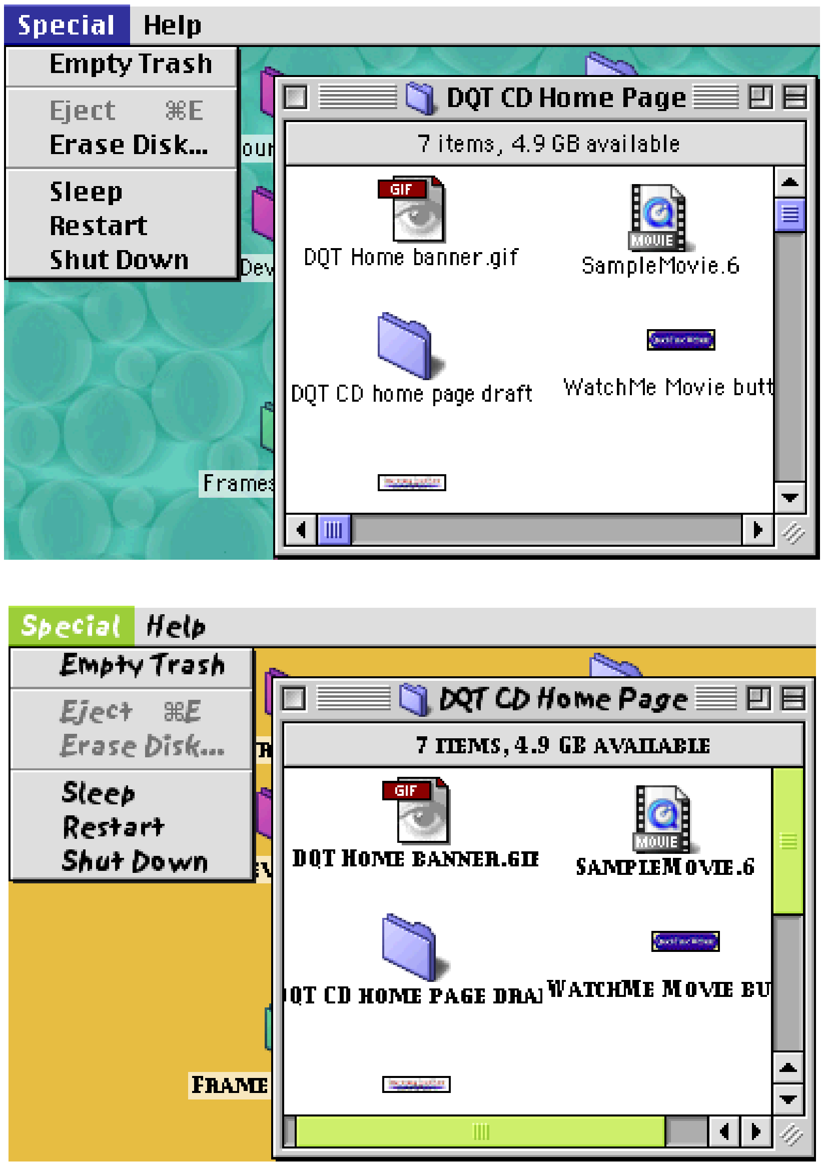
- "The Appearance Control Panel" (page 13) discusses appearances, themes, and the various other settings that the Appearance control panel affects.
- "A Theme-Compliant User Interface" (page 21) discusses the parts of the Mac OS user interface that are affected by the Appearance Manager.
- "Definition Function Mapping and Program Registration" (page 32) discusses how the Appearance Manager maps standard pre–Appearance Manager definition functions to their theme-compliant equivalents.
- "Appearance Manager Versions" (page 34) discusses the availability of the Appearance Manager under various versions of the Mac OS.
- "Appearance Manager Memory Requirements" (page 35) discusses the impact of the Appearance Manager on your program's memory usage.

The Appearance Control Panel

The Appearance control panel is the user interface for the Appearance Manager. Through the Appearance control panel, users can adapt their experience of the system's look and sound by changing the current theme. The changes that the user makes to the current theme apply not just to the Mac OS itself, but also to all theme-compliant programs that the user is currently running.

Figure 2-1 shows the same desktop, before and after a change of themes.

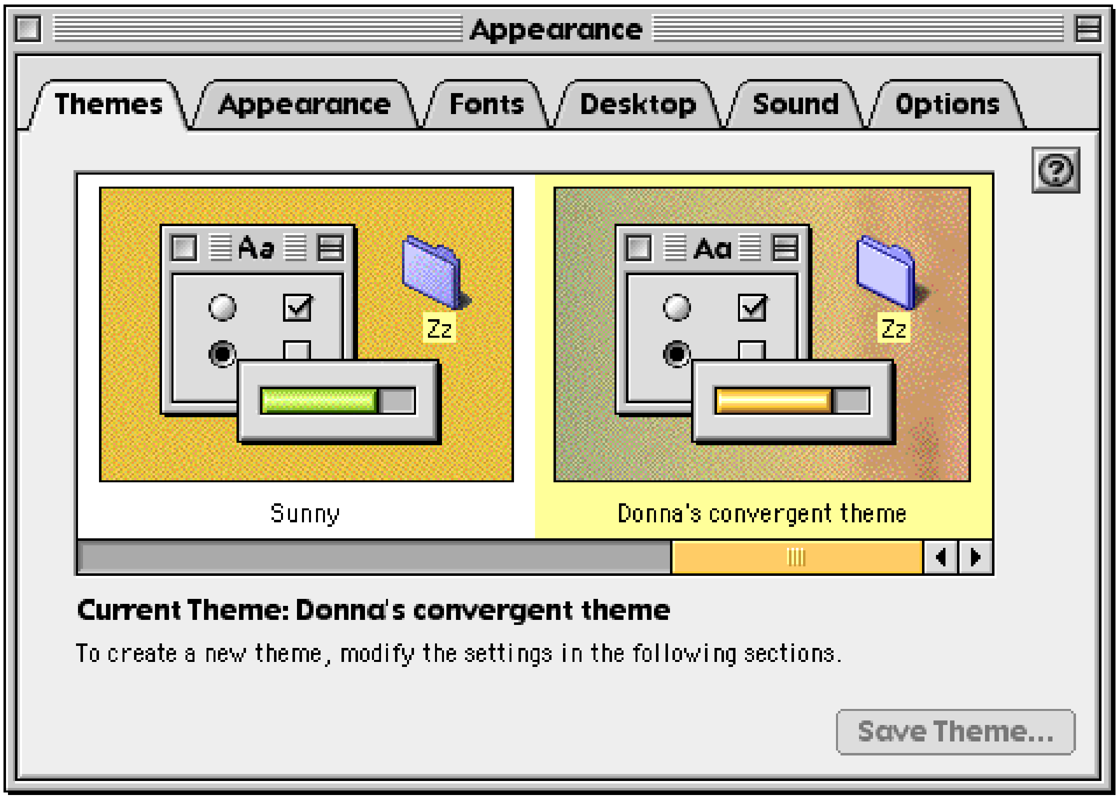
Figure 2-1 The same desktop in two different themes



As Figure 2-1 shows, changing a theme can mean changing various user preferences. A **theme** can contain preferences for the current desktop picture or pattern, the system fonts, any interface-related sounds, the current appearance, and other options.

Each pane of the Appearance control panel allows the user to specify preferences for selected aspects of a theme. As shown in Figure 2-2, the Themes pane is the first presented to the user. In this pane, the user can select the current theme from among various system-supplied themes, or the user can choose a theme that they themselves have previously created.

Figure 2-2 The Themes pane of the Appearance control panel

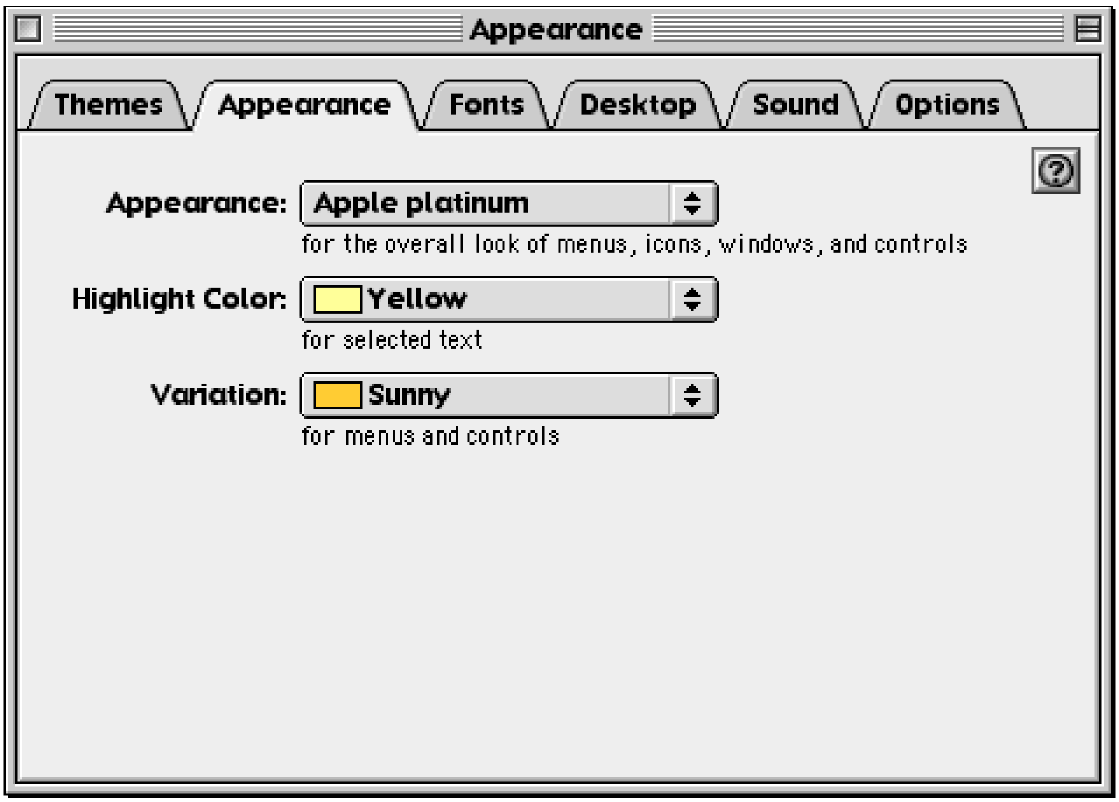


Current Theme: Donna's convergent theme

To create a new theme, modify the settings in the following sections.

Figure 2-3 shows the Appearance pane of the Appearance control panel. In this pane, the user can select highlight and variation colors for an appearance. An **appearance** unifies the look of human interface objects on the system, including alert icons, controls, background colors, dialog boxes, menus, windows, and state transitions.

Figure 2-3 The Appearance pane of the Appearance control panel



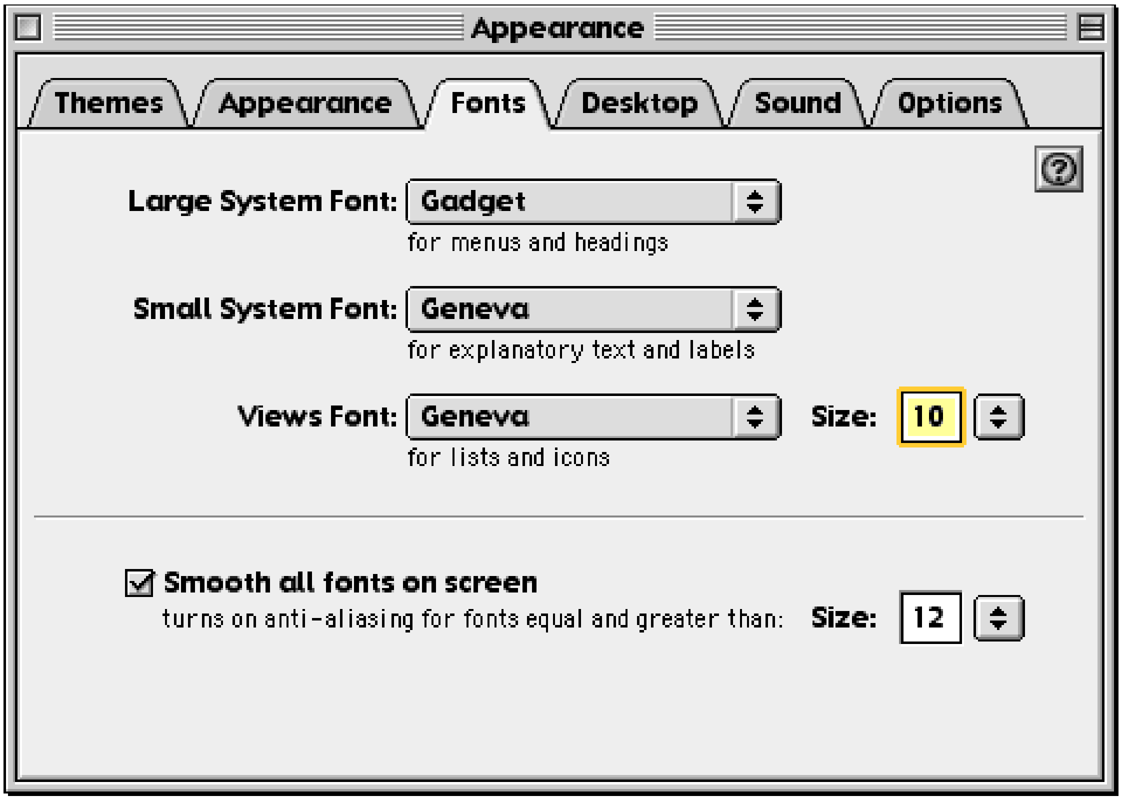
Appearance: Apple platinum
for the overall look of menus, icons, windows, and controls

Highlight Color: Yellow
for selected text

Variation: Sunny
for menus and controls

In the Fonts pane of the Appearance control panel, shown in Figure 2-4, the user can select the preferences for the system fonts in a theme. Note that the Appearance Manager distinguishes between large and small system fonts, as well as providing the user with the option to choose a separate views font for lists and labels, such as those used in Finder windows.

Figure 2-4 The Fonts pane of the Appearance control panel



Large System Font: Gadget
for menus and headings

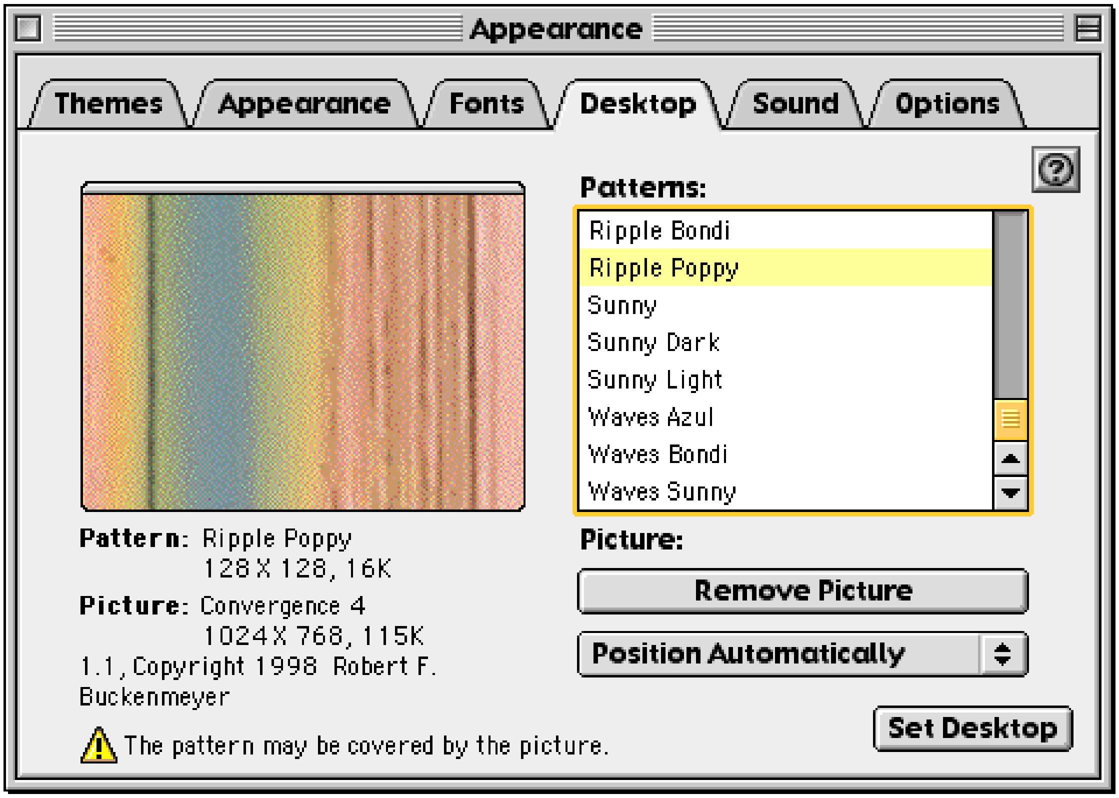
Small System Font: Geneva
for explanatory text and labels

Views Font: Geneva Size: 10
for lists and icons

☑ Smooth all fonts on screen
turns on anti-aliasing for fonts equal and greater than: Size: 12

As shown in Figure 2-5, the user can change the current desktop picture or pattern via the Desktop pane of the Appearance control panel.

Figure 2-5 The Desktop pane of the Appearance control panel



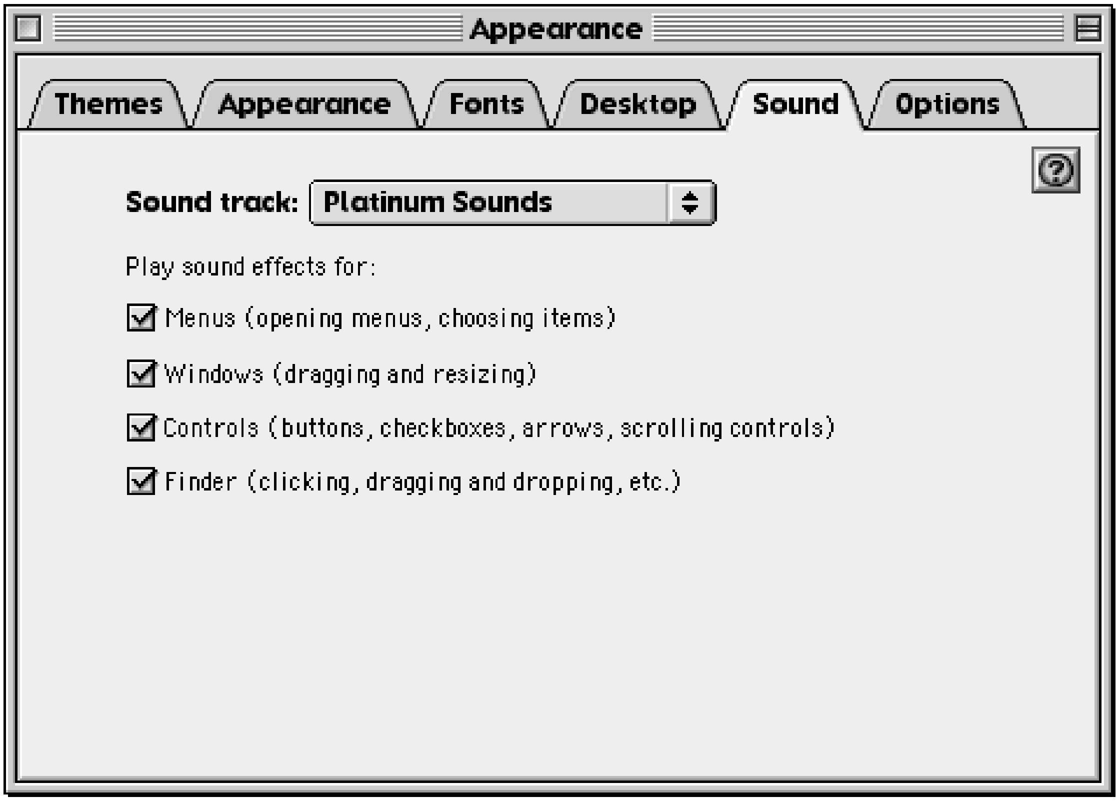
Patterns: Ripple Bondi, Ripple Poppy, Sunny, Sunny Dark, Sunny Light, Waves Azul, Waves Bondi, Waves Sunny

Pattern: Ripple Poppy
128 X 128, 16K
Picture: Convergence 4
1024 X 768, 115K
1.1, Copyright 1998 Robert F. Buckenmeyer

The pattern may be covered by the picture.

Figure 2-6 shows the Sound pane of the Appearance control panel. Users can choose a "sound track" for any or all interface aspects in the current theme, or they can choose to eliminate interface sounds from the theme entirely.

Figure 2-6 The Sound pane of the Appearance control panel



Sound track: Platinum Sounds

Play sound effects for:

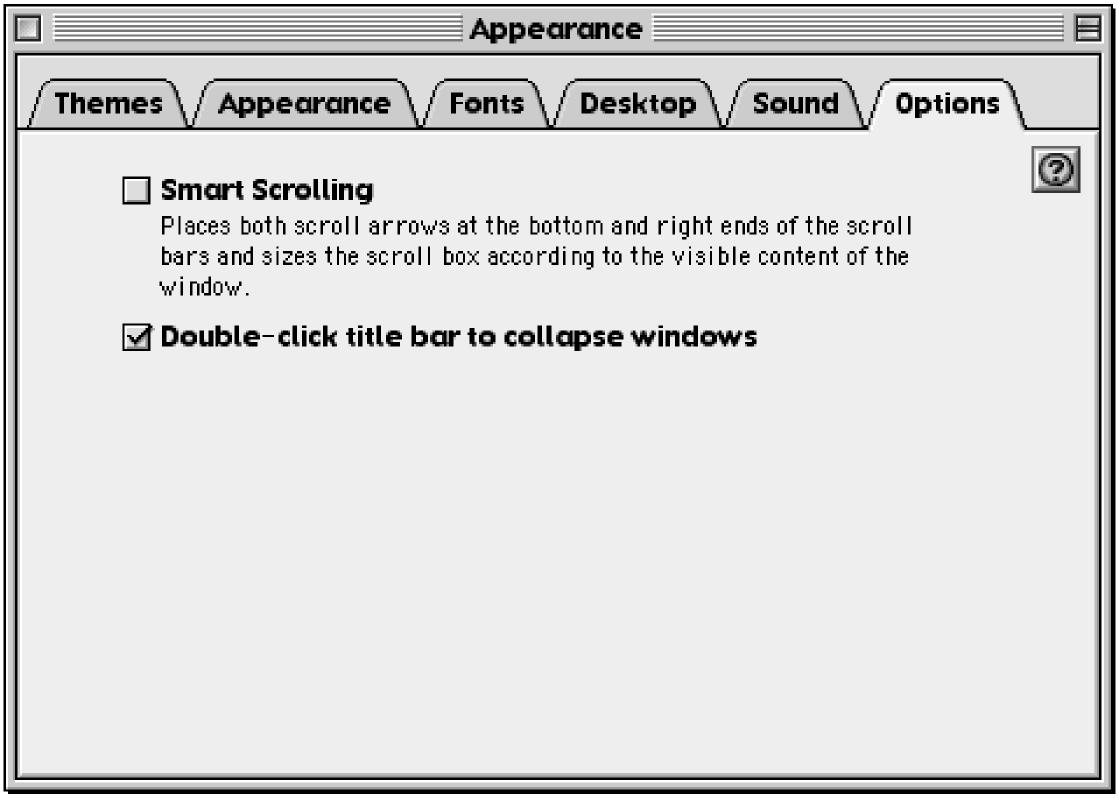
- [✓] Menus (opening menus, choosing items)
- [✓] Windows (dragging and resizing)
- [✓] Controls (buttons, checkboxes, arrows, scrolling controls)
- [✓] Finder (clicking, dragging and dropping, etc.)

In the Options pane of the Appearance control panel, shown in Figure 2-7, the user can select scroll bar preferences and choose the window collapsing behavior for a theme.

When the user selects Smart Scrolling, double scroll bar arrows are used at one end of a scroll bar. For vertical scroll bars, the double arrows are located at the lower end of the scroll bar. For horizontal scroll bars, the double arrows are located at the right end of the scroll bar. The scroll box (also known as a “scroll indicator” or “thumb”) is proportional in size to the amount of a window’s visible content with smart scrolling.

If the user does not select smart scrolling, a single scroll bar arrow is used at each end of a scroll bar and the scroll box is of fixed size.

Figure 2-7 The Options pane of the Appearance control panel



☐ **Smart Scrolling**
Places both scroll arrows at the bottom and right ends of the scroll bars and sizes the scroll box according to the visible content of the window.

☑ **Double-click title bar to collapse windows**

While the Appearance control panel allows users to adapt their experience of the system's look and sound, some programs may also wish to set their own theme preferences, thus creating a custom theme environment in which to run. This is useful for some programs, such as games, that need to control the entire user environment while they are active. See "Creating Custom Themes" (page 56) for more details on this process.

The Appearance Manager saves the preferences that describe a theme in a theme file in the System Folder. The Appearance Manager provides the following functions for working with theme files:

- `GetTheme` (page 70) obtains a collection containing data describing the current theme.
- `SetTheme` (page 73) sets a specified collection as the current theme.
- `IterateThemes` (page 73) iterates over all themes installed on a system.
- `IsValidAppearanceFileType` (page 72) returns whether the system can interpret files of a given file type as appearance files.

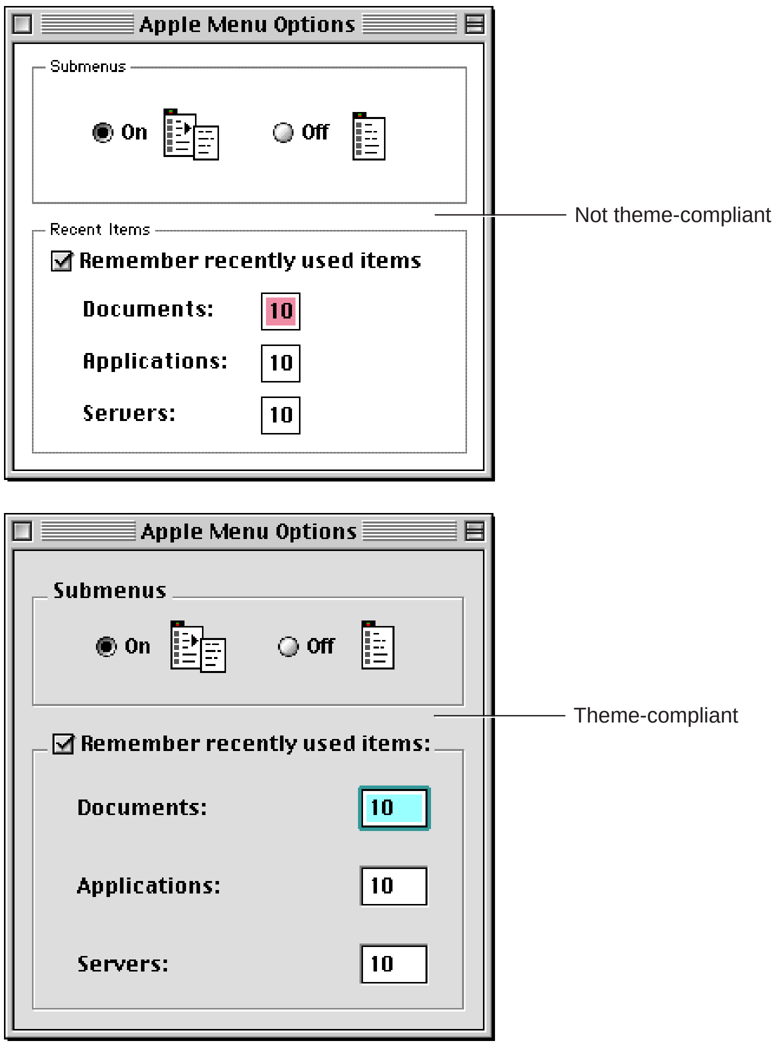
A Theme-Compliant User Interface

A **theme-compliant** program is either one that uses only standard Mac OS human interface elements (that is, controls, windows, and other elements created from standard definition functions) or one that contains custom interface elements but uses the Appearance Manager to be theme-compliant.

Theme-compliant interface elements automatically coordinate with the rest of the user interface under any theme. An interface element that is not theme-compliant may appear to be visually incongruous or may not provide the same behavior (such as not having smart scrolling features) as a theme-compliant element.

Figure 2-8 shows the same dialog box, before and after being made theme-compliant. Comparing the two dialog boxes, you can see that the theme-compliant version uses the correct background color (gray) for the platinum appearance. Another difference is that the theme-compliant dialog box uses standard system-defined primary group boxes; these group boxes have a beveled look in the platinum appearance and allow you to use a checkbox item for the title of the group box. Finally, the theme-compliant dialog box also makes use of a standard focus ring for the editable text field into which the user may currently type.

Figure 2-8 The same dialog box, before and after being made theme-compliant



Apple Menu Options

Submenus

[✓ On] Off

Recent Items

[✓] Remember recently used items

Documents: 10

Applications: 10

Servers: 10

Apple Menu Options

Submenus

[✓ On] Off

[✓] Remember recently used items:

Documents: 10

Applications: 10

Servers: 10

The key to making your program theme-compliant is to allow the system to do as much of your interface work for you as is possible. Using the standard, system-defined interface elements is the biggest step you can take toward theme-compliance. However, if your program uses custom interface elements, you must then use the Appearance Manager to adapt these nonstandard elements to the same coordinated look as the rest of the Mac OS. See "Case Studies for Making Custom Interface Elements Theme-Compliant" (page 46) for examples of making various custom control elements theme-compliant.

Of course, the specific actions necessary to achieve theme-compliance vary from program to program, so you should use the checklist provided in "A Checklist for Creating a Theme-Compliant Program" (page 39) to determine what you need to do to make your program theme-compliant.

The following sections discuss the elements of a theme-compliant user interface in more detail:

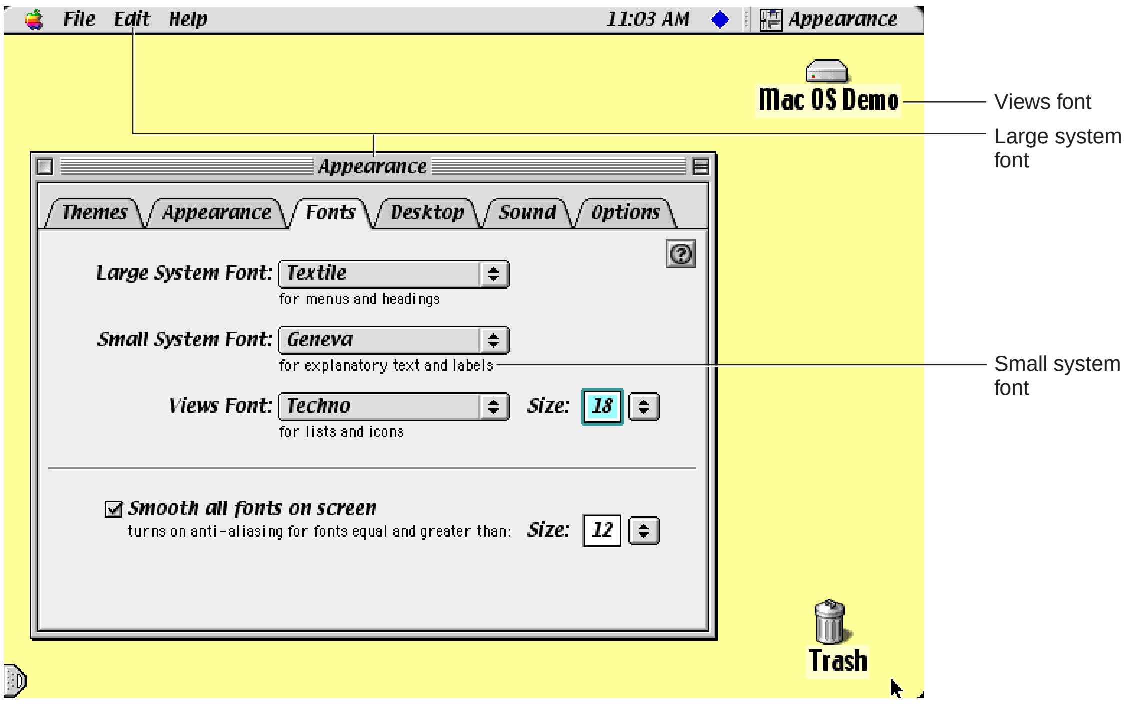
- "Theme-Compliant Fonts" (page 23)
- "Theme-Compliant Sounds" (page 24)
- "Theme-Compliant Cursors" (page 25)
- "Theme-Compliant Controls" (page 25)
- "Theme-Compliant Windows" (page 28)
- "Theme-Compliant Menus" (page 29)
- "Theme-Compliant Colors and Patterns" (page 30)

Theme-Compliant Fonts

As shown in Figure 2-9, the user can use the Appearance control panel to select the preferences for the system fonts in a theme. Because of this, system fonts may change with a theme change while your program is running. If you are using standard interface elements (that is, system-defined windows, controls, and menus), the fonts used for these elements automatically change with a theme change.

Some programs may not use standard interface elements in all instances, however. For example, a program may draw its own text into a dialog box. In such cases, to ensure that the fonts you use match the corresponding system fonts in the current theme, you should use the Appearance Manager to determine the fonts that you use. The Appearance Manager provides the following functions for working with theme fonts:

Figure 2-9 Examples of the views font, the large system font, and the small system font



Large System Font: Textile
for menus and headings

Small System Font: Geneva
for explanatory text and labels

Views Font: Techno
for lists and icons
Size: 18

☑ Smooth all fonts on screen
turns on anti-aliasing for fonts equal and greater than: Size: 12

- `GetThemeFont` (page 71) obtains information about a system font in the current theme.
- `UseThemeFont` (page 74) sets the font of the current graphics port to one of the current theme's system fonts.

Theme-Compliant Sounds

As shown in Figure 2-6 (page 19), the user can select preferences for the use of sounds in a theme. Therefore, not only may users choose to play sounds associated with various aspects of your program's user interface, but the sounds associated with various interface elements may change with a theme change. If you are using standard interface elements (that is, system-defined windows, controls, and menus), the system automatically plays the appropriate sounds, if any, for these elements in the current theme.

Some programs may not use standard interface elements in all instances, however. In such cases, to ensure that your program's sounds match those used in the current theme, you should use the Appearance Manager to determine the sounds that your program uses. The Appearance Manager provides the following functions for playing theme sounds:

- `PlayThemeSound` (page 89) plays an asynchronous sound associated with the specified state change.
- `BeginThemeDragSound` (page 88) continuously plays a theme-specific sound associated with the user's movement of a given interface object.
- `EndThemeDragSound` (page 88) terminates the playing of a sound associated with the user's movement of a given interface object.

Theme-Compliant Cursors

Appearance Manager 1.1 introduces cursors that can change appearance with a theme change. In order to be theme-compliant, your program should use these theme-specific cursors whenever possible, instead of the classic black-and-white or color cursors. To obtain theme-compliant cursors, you must use the Appearance Manager to draw cursors in your program, rather than the QuickDraw cursor utilities.

Because the Appearance Manager cursors are color cursors, they currently cannot be set from interrupt time. Therefore, if you support animated cursors that are changed at interrupt time you should continue to use your own cursors for now. The Appearance Manager provides the following functions for specifying theme-compliant cursors:

- `SetThemeCursor` (page 91) sets the cursor to a version of the specified cursor type that is consistent with the current theme.
- `SetAnimatedThemeCursor` (page 90) animates a version of the specified cursor type that is consistent with the current theme.

Theme-Compliant Controls

Controls are graphical objects, such as buttons, scroll bars, or tabs, that the user can manipulate to take an immediate action or change settings to modify a future action. Your program can use the Control Manager to create standard Mac OS controls. To be theme-compliant, your program should either use standard controls or use the Appearance Manager to adapt its custom control elements. For examples and descriptions of the standard Mac OS 8.*x* controls, see the *Mac OS 8 Human Interface Guidelines* at

<http://developer.apple.com/techpubs/mac/HIGOS8Guide/thig-2.html>

Although your program (and the Control Manager) may typically define many different types of buttons, for simplicity the Appearance Manager treats various types of buttons—including push buttons, checkboxes, radio buttons, arrow buttons, pop-up menu buttons, disclosure triangles, increment/decrement buttons, and bevel buttons—within a single concept of "buttons." The Appearance Manager provides the following functions for creating theme-compliant custom buttons:

- `DrawThemeButton` (page 94) draws a button.
- `DrawThemePopupArrow` (page 102) draws a pop-up arrow.
- `GetThemeButtonBackgroundBounds` (page 113) obtains the rectangle that contains a button.
- `GetThemeButtonContentBounds` (page 114) obtains the rectangle where content can be drawn for a button.
- `GetThemeButtonRegion` (page 115) obtains the region occupied by a button.
- `GetThemeCheckBoxStyle` (page 116) obtains the system preference for the type of mark to use in a checkbox.

The Appearance Manager also treats various types of rectangular controls—including scroll bars, sliders, and progress bars—as a single concept of "tracks." The Appearance Manager provides the following functions for creating theme-compliant custom tracks:

- `DrawThemeTrack` (page 110) draws a track.
- `DrawThemeTrackTickMarks` (page 111) draws tick marks for a track.
- `DrawThemeTickMark` (page 109) draws a tick mark.
- `DrawThemeScrollBarArrows` (page 104) draws scroll bar arrows consistent with the current system preferences.
- `GetThemeTrackBounds` (page 120) obtains the bounding rectangle of a track.

- `GetThemeTrackDragRect` (page 121) obtains the area in which the user may drag a track's indicator.
- `GetThemeTrackLiveValue` (page 122) obtains the current value of a track's indicator, given its relative position.
- `GetThemeTrackThumbPositionFromOffset` (page 123) obtains the relative position of a track's indicator, given an offset from its prior position.
- `GetThemeTrackThumbPositionFromRegion` (page 124) obtains the relative position of a track's indicator, given its current position.
- `GetThemeTrackThumbRgn` (page 124) obtains the region containing a track's indicator.
- `GetThemeScrollBarTrackRect` (page 118) obtains the area containing the track portion of a scroll bar.
- `HitTestThemeTrack` (page 127) returns whether the user clicked upon the specified track.
- `HitTestThemeScrollBarArrows` (page 125) returns whether the user clicked upon the specified scroll bar's arrows.
- `GetThemeScrollBarArrowStyle` (page 116) obtains the system preference for the type of scroll bar arrows to be used.
- `GetThemeScrollBarThumbStyle` (page 117) obtains the system preference for the type of scroll box to be used.

The Appearance Manager provides the following functions for creating theme-compliant custom tabs:

- `DrawThemeTab` (page 107) draws a tab.
- `DrawThemeTabPane` (page 109) draws a tab pane.
- `GetThemeTabRegion` (page 119) obtains the region occupied by a tab.

The Appearance Manager provides the following other functions for creating theme-compliant custom controls:

- `DrawThemeChasingArrows` (page 96) draws an asynchronous arrows indicator.
- `DrawThemeEditTextFrame` (page 97) draws an editable text frame.
- `DrawThemeFocusRect` (page 98) draws or erases a focus ring around a specified rectangle.

- `DrawThemeFocusRegion` (page 99) draws or erases a focus ring around a specified region.
- `DrawThemeGenericWell` (page 100) draws an image well frame.
- `DrawThemeListBoxFrame` (page 100) draws a list box frame.
- `DrawThemePlacard` (page 101) draws a placard.
- `DrawThemePrimaryGroup` (page 103) draws a primary group box frame.
- `DrawThemeSecondaryGroup` (page 105) draws a secondary group box frame.
- `DrawThemeSeparator` (page 106) draws a separator line.

Theme-Compliant Windows

Mac OS applications typically interact with users via windows on the screen. You can use the Window Manager to create, display, and manage the drawing and behavior of standard Mac OS windows. Dialog boxes and alert boxes are specific types of windows that are used to present information to and solicit information from the user. You can use the Dialog Manager to readily implement standard Mac OS dialog boxes and alert boxes. To be theme-compliant, your program should either use standard windows or use the Appearance Manager to adapt your custom window, dialog box, and alert box elements. For examples and descriptions of standard Mac OS 8.*x* windows, see the *Mac OS 8 Human Interface Guidelines* at

<http://developer.apple.com/techpubs/mac/HIGOS8Guide/thig-2.html>

The Appearance Manager provides the following functions for applying theme-compliant colors and patterns to custom windows:

- `SetThemeWindowBackground` (page 143) associates a theme-compliant color or pattern with the background of a window.
- `SetThemeTextColorForWindow` (page 142) sets a window’s foreground color to a theme-compliant color.

The Appearance Manager provides the following functions for drawing theme-compliant custom windows:

- `DrawThemeModelessDialogFrame` (page 129) draws a beveled outline inside the content area of a modeless dialog box.
- `DrawThemeScrollBarDelimiters` (page 130) outlines a window’s scroll bars.

- `DrawThemeStandaloneGrowBox` (page 131) draws a size box.
- `DrawThemeStandaloneNoGrowBox` (page 132) draws a fill image for use in the corner space between scroll bars.
- `DrawThemeTitleBarWidget` (page 133) draws a close box, zoom box, or collapse box.
- `DrawThemeWindowFrame` (page 135) draws a window frame.
- `DrawThemeWindowHeader` (page 136) draws a window header.
- `DrawThemeWindowListViewHeader` (page 137) draws a window list view header.

The Appearance Manager provides the following functions for obtaining window region information:

- `GetThemeStandaloneGrowBoxBounds` (page 138) obtains the bounds of a size box.
- `GetThemeWindowRegion` (page 139) obtains the specified window region.
- `GetThemeWindowRegionHit` (page 140) obtains the part of the window that the user clicked upon.

Theme-Compliant Menus

Menus allow users to view or choose from a list of choices and commands that your application provides. You can use the Menu Manager to create, display, and manage standard Mac OS menus. To be theme-compliant, your program should either use standard menus or use the Appearance Manager to adapt your custom menu and menu bar elements. For examples and descriptions of standard Mac OS 8.*x* menus, see the *Mac OS 8 Human Interface Guidelines* at

<http://developer.apple.com/techpubs/mac/HIGOS8Guide/thig-2.html>

The Appearance Manager provides the following functions for drawing theme-compliant custom menus:

- `DrawThemeMenuBackground` (page 145) draws a menu background.
- `GetThemeMenuBackgroundRegion` (page 150) obtains the background region for a menu.

The Appearance Manager provides the following functions for drawing theme-compliant custom menu titles:

- `DrawThemeMenuTitle` (page 149) draws a menu title.

- `GetThemeMenuTitleExtra` (page 154) obtains a measurement of the space to either side of a menu title.

The Appearance Manager provides the following functions for drawing theme-compliant custom menu items:

- `DrawThemeMenuItem` (page 146) draws a menu item.
- `DrawThemeMenuSeparator` (page 148) draws a menu item separator line.
- `GetThemeMenuItemExtra` (page 152) obtains a measurement of the space surrounding a menu item.
- `GetThemeMenuSeparatorHeight` (page 153) obtains the height of a menu separator line.

The Appearance Manager provides the following functions for drawing theme-compliant custom menu bars:

- `DrawThemeMenuBarBackground` (page 146) draws a menu bar background.
- `GetThemeMenuBarHeight` (page 151) obtains the optimal height of a menu bar.

Theme-Compliant Colors and Patterns

The colors used for interface elements may vary from theme to theme. If you are using standard interface elements (that is, system-defined windows, controls, and menus), the colors used for these elements automatically change with a theme change.

Some programs may not use standard interface elements in all instances, however. In such cases, to ensure that your interface elements coordinate with the current theme, you should use the Appearance Manager to determine the colors (and patterns) that your program uses. See “Using Theme-Compliant Colors and Patterns” (page 43) for more details on this process.

To be theme-compliant, you should not use any set values for the colors of interface objects in your program. For example, compare the background colors of the two dialog boxes shown in Figure 2-8 (page 22). The non-theme-compliant dialog box uses a set background color of white, which contrasts with the theme-compliant dialog box, whose background color is automatically drawn in gray, the correct color for the current theme.

The Appearance Manager provides the following functions for setting the foreground or background of the current graphics port:

- `ApplyThemeBackground` (page 76) sets the background color or pattern of the current port to be consistent with that of an embedding object.
- `SetThemeBackground` (page 83) applies a theme-compliant color or pattern to the background of the current port.
- `SetThemePen` (page 85) applies a theme-compliant color or pattern to the foreground of the current port.

The Appearance Manager provides the following functions for working with text colors:

- `GetThemeTextColor` (page 81) obtains the text color used for a specified element under the current theme.
- `SetThemeTextColor` (page 86) sets the current text color to be consistent with that of a specified element.

The Appearance Manager provides the following functions for obtaining theme color information:

- `GetThemeAccentColors` (page 78) obtains a copy of a theme's accent colors.
- `GetThemeBrushAsColor` (page 79) obtains the color that corresponds to a given theme brush type under the current theme.
- `IsThemeInColor` (page 82) returns whether the current theme would draw in color in the given environment.

The Appearance Manager provides the following functions for working with the drawing state of the current graphics port:

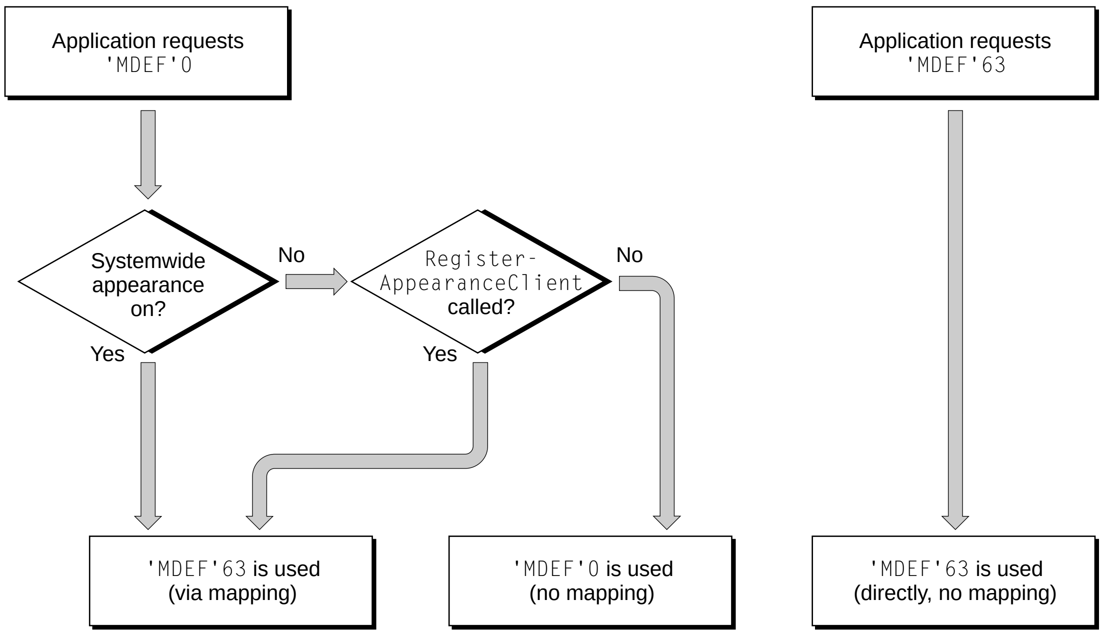
- `GetThemeDrawingState` (page 80) obtains the drawing state of the current graphics port.
- `SetThemeDrawingState` (page 84) sets the drawing state of the current graphics port.
- `DisposeThemeDrawingState` (page 78) releases the memory associated with a reference to a graphics port's drawing state.
- `NormalizeThemeDrawingState` (page 83) sets the current graphics port to a default drawing state.

Definition Function Mapping and Program Registration

One way the Appearance Manager coordinates the system's look and behavior is by mapping standard pre–Appearance Manager definition functions (the `'MBDF' 0`, `'MDEF' 0`, `'WDEF' 0`, `'WDEF' 124`, `'CDEF' 0`, `'CDEF' 1`, and `'CDEF' 63` resources) to their theme-compliant equivalents. With Appearance Manager 1.1, mapping always occurs systemwide. Prior to Appearance Manager 1.1, the user can turn off systemwide appearance and, therefore, systemwide mapping. Programs can ensure that their standard interface elements are mapped—with any version of the Appearance Manager—by registering with the Appearance Manager. See "Becoming a Client of the Appearance Manager" (page 41) for more details on registering your program.

Figure 2-10 shows how the Appearance Manager determines whether mapping occurs for standard definition functions.

Figure 2-10 Mapping of standard definition functions


Application requests 'MDEF'0
Systemwide appearance on?
No
Yes
Register-AppearanceClient called?
No
Yes
Application requests 'MDEF'63
'MDEF'63 is used (via mapping)
'MDEF'0 is used (no mapping)
'MDEF'63 is used (directly, no mapping)

Some mapped definition functions have a slightly different look and behavior than if they were specified directly. For example, since a standard pre–Appearance Manager window definition function can't specify the inclusion of a horizontal zoom box, when the old resource is mapped to a new one, the resulting window still won't have a horizontal zoom box. For this reason (and to eliminate the time spent going through the mapping layer), it's recommended that you specify theme-compliant definition function IDs directly.

Note
Custom definition functions cannot be mapped automatically to theme-compliant equivalents. However, the Appearance Manager does provide functions that you can use to coordinate specific custom interface elements with themes.

The Appearance Manager provides the following functions for registering your program:

- `RegisterAppearanceClient` (page 68) registers your program with the Appearance Manager.
- `UnregisterAppearanceClient` (page 69) informs the Appearance Manager that your program is no longer its client.
- `IsAppearanceClient` (page 67) returns whether a given process is currently registered as a client of the Appearance Manager.

Appearance Manager Versions

A number of versions of the Appearance Manager are currently available. Table 2-1 specifies

- the version of the Appearance Manager that must or may be installed on a given version of the Mac OS
- the version of the API provided by that Appearance Manager version
- the version number produced by passing `gestaltAppearanceVersion` to the `Gestalt` function

Table 2-1 Appearance Manager versions

System version	Appearance Manager version	API version	gestaltAppearanceVersion
System 7.1 through Mac OS 7.6.1	must install 1.0.2 or 1.0.3	1.0.1	1.0.1
Mac OS 8	1.0 (bundled)	1.0	none
Mac OS 8.1	1.0.1 (bundled)	1.0.1	1.0.1
Mac OS 8 and Mac OS 8.1	may install 1.0.2 or 1.0.3	1.0.1	1.0.1
Mac OS 8.5	1.1 (built into the System file)	1.1	1.1.0

Version 1.0 of the Appearance Manager does not advertise its version via `Gestalt`; if the `gestaltAppearanceVersion` selector is not present but the `gestaltAppearanceAttr` selector is present, you can assume version 1.0 of the API is available.

Appearance Manager 1.0.2 uses `Gestalt` to advertise version 1.0.1 of the API. The only differences between Appearance Manager 1.0.1 and 1.0.2 are that 1.0.2 contains extra code for backward compatibility and the ".Keyboard" font, which you can detect by calling the Font Manager function `GetFNum`

Appearance Manager 1.0.3 uses `Gestalt` to advertise version 1.0.1 of the API. The only difference between Appearance Manager 1.0.2 and 1.0.3 is that 1.0.3 no longer contains the ".Keyboard" font. In the 1.0.3 SDK, the font is delivered as a separate suitcase that should be installed into the Fonts folder. The font can still be detected with the function `GetFNum`.

Do not attempt to install a version of the Appearance Manager bundled with the Mac OS onto any other version of the Mac OS. Apple has not tested such configurations and does not recommend or support such configurations.

Appearance Manager Memory Requirements

Because appearance design is intended to be very flexible, some appearances may apply complex, nonrectangular shapes to their interface elements. Because data describing these shapes is saved in the form of QuickDraw `Region` structures—which are of variable length, depending upon the complexity of the shapes being described—the amount of memory your application requires may increase when some appearances are active. If your program's memory usage is finely tuned according to assumptions about the amount of memory consumed on versions of the Mac OS prior to Mac OS 8.5, and in particular the amount of memory consumed by the Window Manager for each window, you may wish to increase your heap size to accommodate appearance-specific memory usage variations.

C H A P T E R 3

Using the Appearance Manager

Contents

Many programs exclusively use standard, system-defined interface elements. If yours is one of these, you may only need to register your program with the Appearance Manager and be prepared to respond to Appearance Manager Apple events in order to be theme-compliant. However, if your program uses any custom interface elements, you may need to use other features of the Appearance Manager for your program to be theme-compliant.

This chapter discusses how you can use the Appearance Manager, through version 1.1, in the following sections:

- "A Checklist for Creating a Theme-Compliant Program" (page 39) presents the main steps you should take to make your program theme-compliant.
- "Becoming a Client of the Appearance Manager" (page 41) describes the process of checking for and registering with the Appearance Manager.
- "Using Theme-Compliant Colors and Patterns" (page 43) discusses how to work with color in your interface.
- "Case Studies for Making Custom Interface Elements Theme-Compliant" (page 46) presents examples of making custom interface elements theme-compliant.
- "Creating Custom Themes" (page 56) describes how to set up a custom environment for your program.

A Checklist for Creating a Theme-Compliant Program

The key to making your program theme-compliant is to allow the system to do as much of your interface work for you as possible. Using the standard, system-defined interface elements is one major step toward theme-compliance. Avoiding making hard-coded assumptions about dimensions and colors is another major step. Both steps save you engineering effort and repay you with a theme-compliant interface. The specific actions necessary to achieve theme-compliance vary from program to program, so you should use the checklist below to determine what you need to do to make your program theme-compliant.

- ☐ Register with the Appearance Manager. See "Becoming a Client of the Appearance Manager" (page 41) for more details.

- ❑ Whenever possible, use the system-defined windows supplied by the Window Manager instead of creating your own. With Mac OS 8.5 and 8.6, the Window Manager provides extensive support for a variety of window features, including floating windows, that developers previously had to implement on their own.
- ❑ Whenever possible, use the system-defined menus provided by the Menu Manager instead of creating your own. With Mac OS 8 and 8.5, the Menu Manager supports a variety of long-requested features, including: an extended selection of available modifier keys to use for keyboard equivalents, different fonts for individual menu items, a wider selection of icon data formats, and menu item command IDs.
- ❑ Whenever possible, use the wide assortment of system-defined controls available with the Mac OS 8 Control Manager instead of creating your own. At a minimum, you should revise any custom controls in your program so that they work in control hierarchies.
- ❑ Make your dialog boxes and alert boxes theme-compliant. The Mac OS 8 Dialog Manager introduces two new resources—the `'dlgx'` and `'alrx'` resources—that you can use to specify theme-compliant features for your dialog boxes and alert boxes, respectively.
- ❑ Where you absolutely cannot use standard interface elements, use Appearance Manager functions to adapt your custom elements to a theme-compliant look. See “Case Studies for Making Custom Interface Elements Theme-Compliant” (page 46) for some examples of using the Appearance Manager to make custom control elements theme-compliant.
- ❑ Remove color table resources for windows, controls, menus, dialog boxes, and alert boxes from your program. Because they limit theme-compliance, these resources—typically used to specify custom color information for interface elements—are now mostly ignored by the system.
- ❑ Make no assumptions about color values for your interface. Instead of hard-coding color values, use the Appearance Manager’s `ThemeBrush` and `ThemeTextColor` constants, described in “Theme Brush Constants” (page 180) and “Theme Text Color Constants” (page 225). When you use these constants in your program, the Appearance Manager automatically applies the correct color or pattern for a given interface element in the current theme. See “Using Theme-Compliant Colors and Patterns” (page 43) for more details on handling color in your program’s interface.
- ❑ Because the measurements of standard interface objects may change in a given appearance, you should make no assumptions about the dimensions of menus, windows, or controls. For example, relying on an unchanging

menu bar height in order to position your windows means that you could end up with the menu bar overlapping your windows after a theme change. Instead of relying on hard-coded dimension values, you should call Appearance Manager functions to obtain the measurements that are used in the current theme.

- □ Call Appearance Manager functions to use theme-compliant color cursors in your program.

Becoming a Client of the Appearance Manager

Before calling any Appearance Manager functions, you should check that the Appearance Manager is present. Listing 3-1 shows an example of a function that simply checks to see whether the Appearance Manager is present. You may wish to perform a more extended check to determine, for example, the version of the Appearance Manager that is installed.

Listing 3-1 Determining whether the Appearance Manager is present

```
static pascal OSStatus MyIsAppearancePresent (Boolean *haveAppearance)
{
    OSStatus err = noErr;

    long response;

    // Attempt to call Gestalt; if we succeed, test for presence of Appearance Mgr
    if (!(err = Gestalt (gestaltAppearanceAttr,&response)))
        *haveAppearance = response & (1 << gestaltAppearanceExists) ? true : false;
    // If the Appearance Mgr selector is undefined, the Appearance Mgr is not present
    else if (err == gestaltUndefSelectorErr)
    {
        *haveAppearance = false;
        err = noErr;
    }

    return err;
}
```

After determining that the Appearance Manager is present, but prior to initializing or drawing any onscreen elements or invoking any definition functions, you should register your program as client of the Appearance Manager by calling the function `RegisterAppearanceClient` (page 68), which is all many programs need to do to be theme-compliant.

You should call `RegisterAppearanceClient` in order to receive Appearance Manager Apple events. With Appearance Manager 1.1 and later, when the user changes the current appearance (that is, when a theme switch occurs), the Appearance Manager sends Apple events to all running applications that are registered as clients of the Appearance Manager and which are high-level event aware. Because typical results of a theme switch might include a change in menu bar height or window structure dimensions, as well as changes to the system fonts, colors, and patterns currently in use, you should listen for and, under most circumstances, respond to the Appearance Manager Apple events. See "Appearance Manager Apple Event Constants" (page 177) for more details on the Appearance Manager Apple events.

Note
The Appearance Manager Apple events are notifications your program receives after the theme switch has occurred, so you should not depend on receiving an Apple event before being asked to redraw in the new theme.

When your program calls `RegisterAppearanceClient`, the Appearance Manager also automatically maps standard pre–Appearance Manager definition functions to their theme-compliant equivalents for your program, whether or not systemwide appearance is active; see "Definition Function Mapping and Program Registration" (page 32) for more details on this process.

You are also encouraged to register plug-ins with the Appearance Manager in order to receive definition function mapping when systemwide appearance is off. You can use the function `IsAppearanceClient` (page 67) to determine if the process containing your plug-in is registered as a client of the Appearance Manager. If it is not, you can register your plug-in on entry by calling `RegisterAppearanceClient` (page 68), and you can unregister it on exit by calling the function `UnregisterAppearanceClient` (page 69).

Using Theme-Compliant Colors and Patterns

As mentioned in "A Checklist for Creating a Theme-Compliant Program" (page 39), one of the first steps in becoming theme-compliant is to remove color table resources for windows, controls, menus, dialog boxes, and alert boxes from your program. The system no longer fully supports the `'wctb'`, `'ictb'`, `'mctb'`, `'dctb'`, `'actb'`, and `'cctb'` resources, which typically have been used to specify custom color information for interface elements. In some cases, using these resources can inhibit the ability of your user interface to integrate with the current theme. For theme-compliant dialog boxes and alert boxes, instead use the following Mac OS 8 Dialog Manager resources: the dialog font table resource (`'dftb'`), the extended dialog resource (`'dlgx'`), and the extended alert resource (`'alrx'`).

Other steps you can take to ensure that your program's use of color is theme-compliant are described in the following sections:

- "Using Theme Brushes" (page 43)
- "Using Theme Text Colors" (page 44)
- "Saving and Restoring the Drawing Environment" (page 44)
- "Obtaining Device Color and Depth Information" (page 45)

Using Theme Brushes

One of the main things that you can do to make your program theme-compliant is to avoid using "fixed" color values for your interface. Instead of hard-coding color values, use the Appearance Manager `ThemeBrush` constants, described in "Theme Brush Constants" (page 180), for painting the background of a window or control. Theme brushes are an abstract mechanism that allows colors and patterns to be coordinated with the current theme. A theme brush may specify either an RGB color or a pixel pattern, depending on the theme. You can pass constants of type `ThemeBrush` in the `inBrush` parameter of the functions `SetThemeBackground` (page 83), `SetThemePen` (page 85), and `SetThemeWindowBackground` (page 143) to specify that the Appearance Manager substitute whatever the appropriate color or pattern is for a given human interface element in the current theme. Using these brushes makes your existing user interface integrate more smoothly with the current theme.

The `SetThemeBackground` function applies a theme-compliant color or pattern to the background of the current graphics port. Your application should call the `SetThemeBackground` function each time you wish to draw a background in a particular brush type. Note that the `SetThemeBackground` function aligns patterns with local coordinates (0,0) in the current port. To apply a theme-compliant color or pattern to the foreground of the current port, use the `SetThemePen` function. Your application should call the `SetThemePen` function each time you wish to draw a foreground element in a specified brush constant. For use specifically with windows, not ports, the `SetThemeWindowBackground` function sets the theme-compliant color or pattern value to which the Window Manager erases a window's background.

Because `ThemeBrush` constants can represent a color or pattern, depending on the current theme, your application must save and restore the current drawing state of the graphics port around calls to `SetThemeBackground`, `SetThemePen`, and `SetThemeWindowBackground`. For details on this process, see "Saving and Restoring the Drawing Environment" (page 44).

Using Theme Text Colors

The Appearance Manager also provides a wide variety of text colors for drawing the text of a control. Again, instead of hard-coding color values for text, use the Appearance Manager's `ThemeTextColor` constants, described in "Theme Text Color Constants" (page 225), which identify a particular context in which text is used. You can pass a constant of type `ThemeTextColor` to the function `SetThemeTextColor` (page 86) to specify that the Appearance Manager substitute whatever the appropriate text color is for a given context under the current theme. You can use the function `GetThemeTextColor` (page 81) to obtain the actual color in use under the current theme for the specified `ThemeTextColor` constant. When you use the `ThemeTextColor` constants in your program, the Appearance Manager automatically applies the correct color for text in the current theme.

Saving and Restoring the Drawing Environment

You may have existing code that saves the state of the graphics port before changing the background or pen, and restores the state after drawing. When adopting theme brushes, you should convert your code to use Appearance Manager functions to save and restore all graphics port state values that can be modified by the theme brush, including pixel patterns and other state information that is not typically accessible to applications.

To obtain the current graphics port state values, you can call the function `GetThemeDrawingState` (page 80) before performing an operation that modifies the drawing state of a graphics port. To return the graphics port to its previous drawing state and release the memory allocated for the drawing state reference, you can call `SetThemeDrawingState` (page 84), providing the reference obtained in the `outState` parameter of `GetThemeDrawingState`. You can also call `DisposeThemeDrawingState` (page 78) to release the allocated memory.

The function `NormalizeThemeDrawingState` (page 83) sets the current graphics port to a default drawing state. `NormalizeThemeDrawingState` sets the background of a graphics port to white; the pen of the port to a size of 1 pixel by 1 pixel, a pattern mode of `patCopy`, and a pattern of black; and the text mode of the port to `srcOr`. The `NormalizeThemeDrawingState` function also flushes from memory any color foreground or background patterns saved in the port's `GrafPort.pnPat` or `GrafPort.bkPat` fields, respectively.

Obtaining Device Color and Depth Information

To be truly theme-compliant, your program should use the QuickDraw function `DeviceLoop` with all of its drawing. `DeviceLoop` automatically supplies your application with the color and depth information that you need to supply to many Appearance Manager functions.

If your program is not drawing via `DeviceLoop`, your program should obtain the color and depth information itself. You typically do this by calling the QuickDraw function `GetGDevice`. Your program then examines the `GDevice` structure for the current device's color and depth information, as shown in Listing 3-2. Note that the example shown may not produce optimal results when you are drawing across multiple monitors with different bit depths.

Listing 3-2 Obtaining color and depth information for the current device

```
// Is the current device a color device?
static pascal Boolean MyGraphicDeviceIsColor (GDHandle gdh)
{
    if (!gdh) gdh = GetGDevice ( );
    return ((1 << gdDevType) & (**gdh).gdFlags) != 0;
}
```

```
// What is the bit depth of the current device?
static pascal short MyGetGraphicDeviceDepth (GDHandle gdh)
{
    if (!gdh) gdh = GetGDevice ( );
    return (**((**gdh).gdPMap)).pixelSize;
}
```

Case Studies for Making Custom Interface Elements Theme-Compliant

As discussed in "A Checklist for Creating a Theme-Compliant Program" (page 39), the easiest way to have your program's interface be theme-compliant is to use the system-defined menus, windows, and controls rather than creating your own. For example, the Mac OS 8 Control Manager supports a variety of new control definitions, as well as enhancements to existing controls, all of which are completely theme-compliant. In the case of your program's controls, if you find that you cannot use the system definitions supplied by Control Manager for all your needs, you then should use the Appearance Manager to ensure that those custom control elements in your program are theme-compliant.

The following four "before and after" case studies show how you can use the Appearance Manager to make a custom control element theme-compliant. Each case presents both a non-theme-compliant, custom control element (in our examples, a frame for an editable text field) and the theme-compliant version of the same element. The cases differ in the ways in which the non-theme-compliant elements have originally been implemented, which, in turn, affects how one makes the elements theme-compliant with the Appearance Manager. You may note that—even in these simple examples—in each case drawing the element with the Appearance Manager requires even less code than is needed for the original, non-theme-compliant drawing.

- "Making an Object Drawn With QuickDraw Theme-Compliant" (page 47)
- "Making a Dialog User Item Theme-Compliant" (page 48)
- "Making a Control User Pane Theme-Compliant" (page 50)
- "Making a Custom Definition Function Theme-Compliant" (page 52)

For a discussion of drawing theme-compliant tracks in your program, see "Drawing Tracks" (page 55).

Making an Object Drawn With QuickDraw Theme-Compliant

Listing 3-3 shows a sample function called `MyEditTextFrameDraw`, which, depending upon the presence of the Appearance Manager, branches between two functions, each of which draws a frame for an editable text field. Prior to drawing, the `MyEditTextFrameDraw` function calls the `MyIsAppearancePresent` function, described in "Becoming a Client of the Appearance Manager" (page 41), to determine whether the Appearance Manager is present.

If the Appearance Manager is not present, `MyEditTextFrameDraw` calls the non-theme-compliant function `MyClassicEditTextFrameDraw`. `MyClassicEditTextFrameDraw` draws a frame by setting the dimensions of the rectangle and its color with calls to QuickDraw. However, an editable text frame drawn in this manner maintains a "fixed" look in any appearance and cannot adapt to theme switches.

If the Appearance Manager is present, however, `MyEditTextFrameDraw` calls the `MyAppearanceSavvyEditTextFrameDraw` function. `MyAppearanceSavvyEditTextFrameDraw` then passes the appropriate Appearance Manager constant for the drawing state (`kThemeStateActive` or `kThemeStateInactive`) to the function `DrawThemeEditTextFrame` (page 97). `DrawThemeEditTextFrame` draws the frame appropriately for the activity state and the current theme. And, when a theme switch occurs, the frame automatically takes on a look consistent with the current theme.

Listing 3-3 Moving from QuickDraw to the Appearance Manager

```
static pascal OSStatus MyClassicEditTextFrameDraw (
                                                const Rect *bounds,
                                                Boolean active)
{
    Rect frame = *bounds;
    InsetRect (&frame,-1,-1);
    // We're pre-Appearance Mgr here, so always draw in black...
    PenNormal ( );
    // unless the editable text field is inactive; in that case, draw in gray
    if (!active) PenPat (&(qd.gray));
    FrameRect (&frame);
    return noErr;
}

static pascal OSStatus MyAppearanceSavvyEditTextFrameDraw (
                                                const Rect *bounds,
                                                Boolean active)
{
    DrawThemeEditTextFrame (bounds,
        active ? kThemeStateActive : kThemeStateInactive);
    return noErr;
}

static pascal OSStatus MyEditTextFrameDraw (const Rect *bounds, Boolean active)
{
    OSStatus err = noErr;

    Boolean haveAppearance;

    if (!(err = MyIsAppearancePresent (&haveAppearance)))
    {
        if (haveAppearance)
            err = MyAppearanceSavvyEditTextFrameDraw (bounds, active);
        else
            err = MyClassicEditTextFrameDraw (bounds, active);
    }

    return err;
}
```

Making a Dialog User Item Theme-Compliant

Listing 3-4 shows another sample function, `MyEditTextFrameUserItemProc`, which, depending upon the presence of the Appearance Manager, branches between two functions, each of which draws a frame for an editable text field. In this example, the editable text frame is defined as a dialog user item. Again, prior to drawing, the `MyEditTextFrameUserItemProc` function calls the `MyIsAppearancePresent` function, described in “Becoming a Client of the

Appearance Manager" (page 41), to determine whether the Appearance Manager is present.

If the Appearance Manager is not present, `MyEditTextFrameUserItemProc` calls the function `MyClassicEditTextFrameUserItemProc`. `MyClassicEditTextFrameUserItemProc` first obtains the frame's rectangle from the Dialog Manager, then supplies the rectangle and a color to QuickDraw to draw the frame. However, an editable text frame drawn in this manner maintains the same "fixed" look in any appearance and cannot adapt to theme switches.

If the Appearance Manager is present, however, `MyEditTextFrameUserItemProc` calls `MyAppearanceSavvyEditTextFrameUserItemProc`. `MyAppearanceSavvyEditTextFrameUserItemProc` then passes the appropriate Appearance Manager constant for the drawing state (`kThemeStateActive` or `kThemeStateInactive`) to the function `DrawThemeEditTextFrame` (page 97). `DrawThemeEditTextFrame` draws the frame appropriately for the activity state and the current theme. And, when a theme switch occurs, the frame automatically takes on a look consistent with the current theme.

Listing 3-4 Drawing a dialog user item that is theme-compliant

```
static pascal void MyClassicEditTextFrameUserItemProc (
                                                WindowPtr window,
                                                DialogItemIndex itemIndex)
{
    short   iType;
    Handle  iHandle;
    Rect    iRect;

    GetDialogItem (window,itemIndex,&iType,&iHandle,&iRect);
    InsetRect (&iRect,-1,-1);
    // We're pre-Appearance Mgr here, so always draw in black...
    PenNormal ( );
    // unless the editable text field is disabled (inactive); if so, draw in gray
    if (iType & kItemDisableBit) PenPat (&(qd.gray));
    FrameRect (&iRect);
}

static pascal void MyAppearanceSavvyEditTextFrameUserItemProc (
                                                WindowPtr window,
                                                DialogItemIndex itemIndex)
```

```
{
    short   iType;
    Handle  iHandle;
    Rect    iRect;

    GetDialogItem (window,itemIndex,&iType,&iHandle,&iRect);
    DrawThemeEditTextFrame (&iRect,
        (iType & kItemDisableBit) ?
            kThemeStateInactive : kThemeStateActive);
}

static pascal void MyEditTextFrameUserItemProc (
                                                WindowPtr window,
                                                DialogItemIndex itemIndex)
{
    OSStatus err = noErr;

    Boolean haveAppearance;

    if (!(err = MyIsAppearancePresent (&haveAppearance)))
    {
        if (haveAppearance)
            MyAppearanceSavvyEditTextFrameUserItemProc (window, itemIndex);
        else
            MyClassicEditTextFrameUserItemProc (window, itemIndex);
    }
}
```

Making a Control User Pane Theme-Compliant

Listing 3-5 shows a sample function, `MyEditTextFrameControlUserPaneDrawProc`, which, depending upon the presence of the Appearance Manager, branches between two functions, each of which draws a frame for an editable text field. In this example, the frame is defined as a control user pane. Before drawing, the `MyEditTextFrameControlUserPaneDrawProc` function calls the `MyIsAppearancePresent` function, described in “Becoming a Client of the Appearance Manager” (page 41), to determine whether the Appearance Manager is present.

`MyEditTextFrameControlUserPaneDrawProc` calls `MyClassicEditTextFrameControlUserPaneDrawProc` if the Appearance Manager is not present. MyClassicEditTextFrameControlUserPaneDrawProc first obtains the control rectangle, then supplies the control rectangle and a color to QuickDraw to draw the frame. However, an editable text frame drawn in this manner maintains a “fixed” look in any appearance and cannot adapt to a theme switch.

If the Appearance Manager is present, MyEditTextFrameControlUserPaneDrawProc calls the MyAppearanceSavvyEditTextFrameControlUserPaneDrawProc function. MyAppearanceSavvyEditTextFrameControlUserPaneDrawProc then passes the appropriate Appearance Manager constant for the drawing state (kThemeStateActive or kThemeStateInactive) to the function DrawThemeEditTextFrame (page 97). DrawThemeEditTextFrame draws the frame appropriately for the activity state and the current theme. And, when a theme switch occurs, the frame automatically takes on a look consistent with the current theme.

Listing 3-5 Drawing a control user pane that is theme-compliant

```
static pascal void MyClassicEditTextFrameControlUserPaneDrawProc (
                                                    ControlHandle control,
                                                    SInt16 /* part */)
{
    Rect contrlRect = (**control).contrlRect;
    InsetRect (&contrlRect,-1,-1);
    // We’re pre-Appearance Mgr here, so always draw in black...
    PenNormal ( );
    // unless the control part code value in contrlHilite indicates
    // that the control is inactive (or disabled); if so, draw in gray
    if ((**control).contrlHilite >= 254)
        PenPat (&(qd.gray));
    FrameRect (&contrlRect);
}

static pascal void MyAppearanceSavvyEditTextFrameControlUserPaneDrawProc (
                                                    ControlHandle control,
                                                    SInt16 /* part */)
{
    Rect contrlRect = (**control).contrlRect;
    DrawThemeEditTextFrame (&contrlRect,
        ((**control).contrlHilite < 254) ?
        kThemeStateActive : kThemeStateInactive);
}

static pascal void MyEditTextFrameControlUserPaneDrawProc (
                                                    ControlHandle control,
                                                    SInt16 part)
{
    OSStatus err = noErr;

    Boolean haveAppearance;

    if (!(err = MyIsAppearancePresent (&haveAppearance)))
    {
        if (haveAppearance)
            MyAppearanceSavvyEditTextFrameControlUserPaneDrawProc (control, part);
        else
            MyClassicEditTextFrameControlUserPaneDrawProc (control, part);
    }
}
```

Making a Custom Definition Function Theme-Compliant

The `MyEditTextFrameControlDefProc` function, shown in Listing 3-6, draws a frame for an editable text field that is defined as a custom control definition function. The `MyEditTextFrameControlDefProc` function calls the `MyIsAppearancePresent` function, described in “Becoming a Client of the Appearance Manager” (page 41), to determine whether the Appearance Manager is present. Depending upon the presence of the Appearance Manager, `MyEditTextFrameControlDefProc` branches between two functions to draw the frame.

If the Appearance Manager is not present, `MyEditTextFrameControlDefProc` calls the non-theme-compliant function, `MyClassicEditTextFrameControlDefProc`. `MyClassicEditTextFrameControlDefProc` first obtains the control rectangle, then supplies the control rectangle and a color to QuickDraw to draw the frame. However, an editable text frame drawn in this manner maintains the same “fixed” look in any appearance, and it cannot adapt to theme switches.

If the Appearance Manager is present, however, `MyEditTextFrameControlDefProc` calls the `MyAppearanceSavvyEditTextFrameControlDefProc` function.

`MyAppearanceSavvyEditTextFrameControlDefProc` then passes the appropriate Appearance Manager constant for the drawing state (`kThemeStateActive` or `kThemeStateInactive`) to the function `DrawThemeEditTextFrame` (page 97). `DrawThemeEditTextFrame` draws the frame appropriately for the activity state and the current theme. And, when a theme switch occurs, the frame automatically takes on a look consistent with the current theme.

Listing 3-6 Drawing a custom definition function that is theme-compliant

```
static pascal SInt32 MyClassicEditTextFrameControlDefProc (
                                                        SInt16 /* varCode */,
                                                        ControlHandle control,
                                                        ControlDefProcMessage message,
                                                        SInt32 /* param */)
{
    Rect contrlRect;

    switch (message)
    {
        case drawCntl :

            contrlRect = (**control).contrlRect;
            InsetRect (&contrlRect,-1,-1);
            // We're pre-Appearance Mgr here, so always draw in black...
            PenNormal ( );
            // unless the control part code value in contrlHilite indicates
            // that the control is inactive (or disabled); if so, draw in gray
            if ((**control).contrlHilite >= 254)
                PenPat (&(qd.gray));
            FrameRect (&contrlRect);
            break;

        default :

            // other cases omitted for simplicity
            break;
    }

    return 0;
}
```

```
static pascal SInt32 MyAppearanceSavvyEditTextFrameControlDefProc (
                                                    SInt16 /* varCode */,
                                                    ControlHandle control,
                                                    ControlDefProcMessage message,
                                                    SInt32 /* param */)
{
    Rect contrlRect;

    switch (message)
    {
        case drawCntl :

            contrlRect = (**control).contrlRect;
            DrawThemeEditTextFrame (&contrlRect,
                ((**control).contrlHilite < 254) ?
                    kThemeStateActive : kThemeStateInactive);
            break;

        default :

            // other cases omitted for simplicity
            break;
    }

    return 0;
}

static pascal SInt32 MyEditTextFrameControlDefProc (
                                                SInt16 varCode,
                                                ControlHandle control,
                                                ControlDefProcMessage message,
                                                SInt32 param)
{
    OSStatus err = noErr;

    Boolean haveAppearance;

    if (!(err = MyIsAppearancePresent (&haveAppearance)))
    {
        if (haveAppearance)
```

```
            err = MyAppearanceSavvyEditTextFrameControlDefProc (
                                                        varCode,
                                                        control,
                                                        message,
                                                        param);
        else
            err = MyClassicEditTextFrameControlDefProc (
                                                        varCode,
                                                        control,
                                                        message,
                                                        param);
    }

    return err;
}
```

Drawing Tracks

The Appearance Manager provides a variety of functions that you can use to make your program's tracks—that is, its scroll bars, sliders, and progress bars—theme-compliant. You can use the Appearance Manager to handle most aspects of drawing, obtaining values for, and checking for mouse-down events on tracks and their related parts (such as the indicator on a scroll bar or slider, a scroll bar's arrows, or the tick marks on a slider).

Your application can use the function `DrawThemeTrack` (page 110) to draw a theme-compliant slider, progress bar, or scroll bar. If you use `DrawThemeTrack` to draw a scroll bar, use the function `DrawThemeScrollBarArrows` (page 104) to draw the scroll bar's arrows; see Listing 3-7 for an example of using these two functions together to draw a complete scroll bar. If you use `DrawThemeTrack` to draw a slider, use `DrawThemeTrackTickMarks` (page 111) if you need to draw tick marks for the slider.

Listing 3-7 Drawing a scroll bar with arrows

```
Rect bounds;
ThemeTrackDrawInfo drawInfo;
OSStatus err;

SetRect (&bounds, 10, 10, 200, 26);
```

```
/* Draw the arrows and pass the actual track rect into the
drawInfo structure for DrawThemeTrack to use */
err = DrawThemeScrollBarArrows (&bounds, kThemeTrackActive, 0, true, &drawInfo.bounds);

if (err == noErr)
{
    drawInfo.kind = kThemeScrollBar;
    /* drawInfo.bounds is set to the modified bounds, with the arrows removed,
    on exit from DrawThemeScrollBarArrows */
    drawInfo.min = 0;
    drawInfo.max = 100;
    drawInfo.value = 65;
    drawInfo.attributes = kThemeTrackHorizontal | kThemeTrackShowThumb;
    drawInfo.enableState = kThemeTrackActive;
    drawInfo.trackInfo.scrollbar.viewsize = 0;
    drawInfo.trackInfo.scrollbar.pressState = 0;

    err = DrawThemeTrack (&drawInfo, NULL, NULL, 0);
}
```

Creating Custom Themes

There are two common cases where you may wish to create a custom theme. The first is to set up a custom theme environment for your program, to be used only when your program is active. This is useful for some programs, such as games, that need to control the entire user environment while they are active. The second is to create a theme environment that you want to be user-selectable and to have systemwide effect.

In the first case, if you set up a custom theme environment for your application, you don't need the theme you create to show up in the Appearance control panel (that is, to be user-selectable). The steps to create a custom theme that is not user-selectable are as follows:

1. Create a collection, using the Collection Manager function `NewCollection`.
2. Using theme collection tags and Collection Manager functions, add collection items describing attributes of the theme to the collection. Note that you do

not need to add collection items for every tag defined by the Appearance Manager; you need to include collection items only for those attributes that you want to change. See "Theme Collection Tags" (page 190) for descriptions of available tag values.

3. Call the function `GetTheme` (page 70) to get a collection containing a copy of the data for the theme that is currently running.
4. Call `SetTheme` to set your collection as the current theme.
5. After you have finished using the custom theme environment for your program, restore the previous theme by calling `SetTheme`. Pass `SetTheme` a reference to the collection for the previous theme, which you obtained with your call to `GetTheme` in Step 3.
6. Dispose of the collection and its contents when you are done.

In the second case, if you want the theme to be user-selectable and to have systemwide effect, you need the theme to show up in the Appearance control panel and must therefore follow these steps:

1. Create a collection, using the Collection Manager function `NewCollection`.
2. Using theme collection tags and Collection Manager functions, add collection items describing attributes of the theme to the collection. Note that you do not need to add collection items for every tag defined by the Appearance Manager; you need to include collection items only for those attributes that you want to change. See "Theme Collection Tags" (page 190) for descriptions of available tag values.
3. Flatten the collection to a handle, using the Collection Manager function `FlattenCollectionToHdl`.
4. Use the Resource Manager to save the flattened collection as a resource of type `'scen'`. Save the `'scen'` resource into a custom theme file (that is, in a file of type `'scen'`) in the Theme Files folder; see "Appearance Manager File Type Constants" (page 179) for a description of the `'scen'` file type. See *More Macintosh Toolbox* for a discussion of the Resource Manager and how resources are added to files and released.
5. Dispose of the collection and its contents when you are done.

Appearance Manager Reference

Contents

This chapter describes the Appearance Manager application programming interface (API) through Appearance Manager 1.1, as follows:

Gestalt Constants

Before calling any functions dependent upon the Appearance Manager's presence, your application should pass the selector `gestaltAppearanceAttr` to the `Gestalt` function to determine whether the Appearance Manager is present. To determine which version of the Appearance Manager is installed, your application should check for the presence of the `Gestalt` selector `gestaltAppearanceVersion`.

```
enum {
    gestaltAppearanceAttr          = 'appr',
    gestaltAppearanceVersion       = 'apvr',
    gestaltAppearanceExists        = 0,
    gestaltAppearanceCompatMode    = 1
};
```

Constant descriptions

`gestaltAppearanceAttr`
: The `Gestalt` selector passed to determine whether the Appearance Manager is present. Produces a 32-bit value whose bits you should test to determine which Appearance Manager features are available.

`gestaltAppearanceVersion`
: The `Gestalt` selector passed to determine which version of the Appearance Manager is installed. If this selector exists,

Appearance Manager 1.0.1 (or later) is installed. The version number of the currently installed Appearance Manager is returned in the low-order word of the result in binary code decimal format (for example, version 1.0.1 would be 0x0101). If this selector does not exist but `gestaltAppearanceAttr` does, Appearance Manager 1.0 is installed.

`gestaltAppearanceExists`
: If this bit is set, Appearance Manager functions are available. To determine which version of the Appearance Manager is installed, check for the presence of the `Gestalt` selector `gestaltAppearanceVersion`. If this bit is not set, Appearance Manager functions are not available.

`gestaltAppearanceCompatMode`
: If this bit is set, systemwide platinum appearance is off. When systemwide platinum appearance is off, the Appearance Manager does not auto-map standard System 7 definition functions to their Mac OS 8 equivalents (for those applications that have not called `RegisterAppearanceClient`). If this bit is not set, systemwide platinum appearance is on, and the Appearance Manager auto-maps standard System 7 definition functions to their Mac OS 8 equivalents for all applications.

Functions

The Appearance Manager provides functions in the following areas:

- "Registering With the Appearance Manager" (page 67)
- "Accessing Theme Information" (page 69)
- "Using Theme-Compliant Colors and Patterns" (page 75)
- "Playing Theme Sounds" (page 87)
- "Specifying Theme-Compliant Cursors" (page 90)
- "Drawing Theme-Compliant Controls" (page 92)
- "Drawing Theme-Compliant Windows" (page 128)

- "Drawing Theme-Compliant Menus" (page 144)

Registering With the Appearance Manager

The Appearance Manager provides the following registration functions:

- `RegisterAppearanceClient` (page 68) registers your program with the Appearance Manager.
- `UnregisterAppearanceClient` (page 69) informs the Appearance Manager that your program is no longer its client.
- `IsAppearanceClient` (page 67) returns whether a given process is currently registered as a client of the Appearance Manager.

IsAppearanceClient

Returns whether a given process is currently registered as a client of the Appearance Manager.

```
pascal Boolean IsAppearanceClient (
                        const ProcessSerialNumber *process);
```

`process`	A pointer to a value of type `ProcessSerialNumber`. Pass the serial number of the process to examine.
function result	A value of type `Boolean`. If `true`, the specified process is currently registered as a client of the Appearance Manager; otherwise, `false`.

DISCUSSION

Applications typically do not need to call the `IsAppearanceClient` function. A plug-in could call `IsAppearanceClient` to determine whether the process in which it is running is a registered Appearance Manager client. To register with the Appearance Manager, call the function `RegisterAppearanceClient` (page 68).

VERSION NOTES

Available with Appearance Manager 1.1 and later.

RegisterAppearanceClient

Registers your program with the Appearance Manager.

```
pascal OSStatus RegisterAppearanceClient (void);
```

function result A result code. The result code `appearanceProcessRegisteredErr` indicates that your program was already registered when you called the `RegisterAppearanceClient` function. For other possible result codes, see "Result Codes" (page 238).

DISCUSSION

The `RegisterAppearanceClient` function must be called at the beginning of your program, prior to initializing or drawing any onscreen elements or invoking any definition functions, such as the menu bar.

You should call `RegisterAppearanceClient` in order to receive Appearance Manager Apple events. Under Appearance Manager 1.1 and later, when the user changes the current appearance (that is, when a theme switch occurs), the Appearance Manager sends Apple events to all running applications that are registered as clients of the Appearance Manager and which are high-level event aware. Because typical results of a theme switch might include a change in menu bar height or window structure dimensions, as well as changes to the system fonts, colors, and patterns currently in use, you should listen for and respond to the Appearance Manager Apple events under most circumstances. See "Appearance Manager Apple Event Constants" (page 177) for more details.

When your program calls `RegisterAppearanceClient`, the Appearance Manager also automatically maps standard pre–Appearance Manager definition functions to their theme-compliant equivalents for your program, whether or not systemwide appearance is active; see "Definition Function Mapping and Program Registration" (page 32) for more details on this process.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

SEE ALSO

The function `UnregisterAppearanceClient` (page 69).

UnregisterAppearanceClient

Informs the Appearance Manager that your program is no longer its client.

```
pascal OSStatus UnregisterAppearanceClient (void);
```

function result A result code. The result code `appearanceProcessNotRegisteredErr` indicates that your program was not registered when you called the `UnregisterAppearanceClient` function. For other possible result codes, see "Result Codes" (page 238).

DISCUSSION

The `UnregisterAppearanceClient` function is automatically called for you when your program terminates. While you do not typically need to call this function, you might want to call `UnregisterAppearanceClient` if you are running a plug-in architecture, and you know that a given plug-in is not theme-compliant. In this case you would bracket your use of the plug-in with calls to `UnregisterAppearanceClient` (before the plug-in is used) and `RegisterAppearanceClient` (after the plug-in is used), so that the Appearance Manager is turned off for the duration of the plug-in's usage.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

SEE ALSO

The function `RegisterAppearanceClient` (page 68).

Accessing Theme Information

The Appearance Manager provides the following functions for working with theme and appearance files:

- `IsValidAppearanceFileType` (page 72) returns whether the system can interpret files of a given file type as appearance files.
- `GetTheme` (page 70) obtains a collection containing data describing the current theme.

- `SetTheme` (page 73) sets a specified collection as the current theme.
- `IterateThemes` (page 73) iterates over all themes installed on a system.

The Appearance Manager provides the following functions for working with theme fonts:

- `GetThemeFont` (page 71) obtains information about a system font in the current theme.
- `UseThemeFont` (page 74) sets the font of the current graphics port to one of the current theme's system fonts.

GetTheme

Obtains a collection containing data describing the current theme.

```
pascal OSStatus GetTheme (
                    Collection ioCollection);
```

`ioCollection`	A value of type `Collection`. Pass a reference to a collection object, such as that created by calling the Collection Manager function `NewCollection`. On return, the collection contains data describing attributes of the current theme. See *Inside Macintosh: QuickDraw GX Environment and Utilities* for a description of `NewCollection`.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

The `GetTheme` function obtains a collection containing a copy of the data for the current theme. The theme data is in the form of collection items, each corresponding to an attribute of the theme. For a given theme, the actual number of collection items may vary, depending upon how fully the theme's attributes are specified. See "Theme Collection Tags" (page 190) for descriptions of the possible theme collection items.

Your application can use theme collection tags, along with various Collection Manager functions, to access the data in the collection.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

SEE ALSO

The function `SetTheme` (page 73).

GetThemeFont

Obtains information about a system font in the current theme.

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

`inFontID`	A value of type `ThemeFontID`. Pass a constant specifying the kind of font (that is, the current large, small, or small emphasized system fonts or the views font) for which you wish to retrieve the current font name, size, and style in use. See “Theme Font ID Constants” (page 200) for descriptions of possible values.
`inScript`	A value of type `ScriptCode`. Pass a script code identifying the script system for which you wish obtain font information. You may pass the metascript code `smSystemScript` to specify the system script.
`outFontName`	A value of type `StringPtr`. Pass a pointer to a Pascal string. On return, the string contains the name of the font in use. Pass `NULL` if you do not wish to obtain this information.
`outFontSize`	A pointer to a signed 16-bit integer. On return, the integer value specifies the size of the font in use. Pass `NULL` if you do not wish to obtain this information.
`outStyle`	A pointer to a value of type `Style`. On return, the value specifies the style of the font in use. Pass `NULL` if you do not wish to obtain this information.
function result	A result code; see “Result Codes” (page 238).

DISCUSSION

Your application can call the `GetThemeFont` function to obtain the precise font settings (font name, size, and style) used by a system font under the current theme.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

SEE ALSO

The function `UseThemeFont` (page 74).

IsValidAppearanceFileType

Returns whether the system can interpret files of a given file type as appearance files.

```
pascal Boolean IsValidAppearanceFileType (
                     OSType fileType);
```

`fileType`	A four-character code. Pass the file type to be examined.
function result	A value of type `Boolean`. If `true`, files of the specified file type can be used as appearance files; otherwise, `false`.

DISCUSSION

Under Appearance Manager 1.1, only the `'thme'` and `'pltn'` file types, described in "Appearance Manager File Type Constants" (page 179), are valid appearance file types. Your application typically does not need to call this function.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

IterateThemes

Iterates over all themes installed on a system.

```
pascal OSStatus IterateThemes (
                    ThemeIteratorUPP inProc,
                    void *inUserData);
```

`inProc`	A value of type `ThemeIteratorUPP` (page 171). Pass a universal procedure pointer to an application-defined function such as that described in `MyThemeIteratorProc` (page 160). `IterateThemes` calls the specified function for each theme found installed in the system.
`inUserData`	A pointer to data of any type. Provide any data to be passed in to the `inUserData` parameter of the callback function specified in the `inProc` parameter. Pass `NULL`, if you do not wish to provide any data.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

The `IterateThemes` function continues to iterate until the function specified in the `inProc` parameter returns `false` or until there are no more themes.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

SetTheme

Sets a specified collection as the current theme.

```
pascal OSStatus SetTheme (
                    Collection ioCollection);
```

`ioCollection`	A value of type `Collection`. Pass a reference to a collection object, such as that created by calling the Collection Manager function `NewCollection`. Before calling `SetTheme`, set the

collection to contain theme data that you wish to use for the current theme. The theme data is in the form of collection items, each corresponding to an attribute of the theme. For a given theme, the actual number of collection items may vary, depending upon how fully the theme's attributes are specified. Your application can use theme collection tags, along with various Collection Manager functions, to access the data in the collection. See "Theme Collection Tags" (page 190) for descriptions of the possible theme collection items. See *Inside Macintosh: QuickDraw GX Environment and Utilities* for a discussion of collections.

function result A result code; see "Result Codes" (page 238).

DISCUSSION

The `SetTheme` function sets the attributes of the current theme. You may use `SetTheme` to set up a custom theme environment for your application, to be used only when your application is active. You may also use `SetTheme` to create a theme environment that you want to be user-selectable and to have systemwide effect. See "Creating Custom Themes" (page 56) for more details on these processes.

Your application can use the `GetTheme` (page 70) function to obtain a collection containing a copy of the data for the current theme.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

UseThemeFont

Sets the font of the current graphics port to one of the current theme's system fonts.

```
pascal OSStatus UseThemeFont (
                        ThemeFontID inFontID,
                        ScriptCode inScript);
```

`inFontID`	A value of type `ThemeFontID`. Pass a constant specifying the kind of font (that is, the current large, small, or small emphasized system fonts or the views font) to be applied to the current port. See "Theme Font ID Constants" (page 200) for descriptions of possible values.
`inScript`	A value of type `ScriptCode`. Pass a script code specifying the script system for which you wish to set the current font; you may pass the metascript code `smSystemScript` to specify the system script.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

Your application can call the `UseThemeFont` function to draw text in one of the current theme's system fonts.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

SEE ALSO

The function `GetThemeFont` (page 71).

Using Theme-Compliant Colors and Patterns

The Appearance Manager provides the following functions for working with the drawing state of the current graphics port:

- `GetThemeDrawingState` (page 80) obtains the drawing state of the current graphics port.
- `SetThemeDrawingState` (page 84) sets the drawing state of the current graphics port.
- `DisposeThemeDrawingState` (page 78) releases the memory associated with a reference to a graphics port's drawing state.
- `NormalizeThemeDrawingState` (page 83) sets the current graphics port to a default drawing state.

The Appearance Manager provides the following functions for setting the foreground or background of the current graphics port:

- `ApplyThemeBackground` (page 76) sets the background color or pattern of the current port to be consistent with that of an embedding object.
- `SetThemeBackground` (page 83) applies a theme-compliant color or pattern to the background of the current port.
- `SetThemePen` (page 85) applies a theme-compliant color or pattern to the foreground of the current port.

The Appearance Manager provides the following functions for working with text colors:

- `GetThemeTextColor` (page 81) obtains the text color used for a specified element under the current theme.
- `SetThemeTextColor` (page 86) sets the current text color to be consistent with that of a specified element.

The Appearance Manager provides the following functions for obtaining theme color information:

- `GetThemeAccentColors` (page 78) obtains a copy of a theme's accent colors.
- `GetThemeBrushAsColor` (page 79) obtains the color that corresponds to a given theme brush type under the current theme.
- `IsThemeInColor` (page 82) returns whether the current theme would draw in color in the given environment.

ApplyThemeBackground

Sets the background color or pattern of the current port to be consistent with that of an embedding object.

```
pascal OSStatus ApplyThemeBackground (
                    ThemeBackgroundKind inKind,
                    const Rect *bounds,
                    ThemeDrawState inState,
                    SInt16 inDepth,
                    Boolean inColorDev);
```

`inKind`	A value of type `ThemeBackgroundKind`. Pass a constant specifying the type of embedding object. See "Theme Background Kind Constants" (page 179) for descriptions of possible values.
`bounds`	A pointer to a structure of type `Rect`. Before calling `ApplyThemeBackground`, set the rectangle to a size and position that contains the embedding object, in local coordinates.
`inState`	A value of type `ThemeDrawState`. Pass a constant specifying the current state of the embedding object. See "Theme Draw State Constants" (page 199) for descriptions of possible values.
`inDepth`	A signed 16-bit integer. Pass a value specifying the bit depth (in bits per pixel) of the current graphics port.
`inColorDev`	A value of type `Boolean`. Pass `true` to indicate that you are drawing on a color device, or `false` for a monochrome device.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

The `ApplyThemeBackground` function sets the background color or pattern of the current port to match the background of an embedding object, such as a placard or tab control. Your application should call `ApplyThemeBackground` before erasing the background of your application's content to ensure that the content background matches that of the object in which it is visually embedded.

`ApplyThemeBackground` aligns patterns based on the rectangle passed in the `bounds` parameter. This is in contrast to the function `SetThemeBackground` (page 83), which aligns patterns based on the origin of the current port.

You do not need to call `ApplyThemeBackground` if your content is an embedded part within a control hierarchy and is logically as well as visually embedded in its container; in this case, the Control Manager automatically requests the embedding control to set up the background before drawing the embedded control.

If you have a custom control definition function that erases its background before drawing, you should use the Control Manager function `SetUpControlBackground` before erasing. `SetUpControlBackground` calls `ApplyThemeBackground` if necessary.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

DisposeThemeDrawingState

Releases the memory associated with a reference to a graphics port's drawing state.

```
pascal OSStatus DisposeThemeDrawingState (
                        ThemeDrawingState inState);
```

`inState`	A value of type `ThemeDrawingState` (page 165). Pass a value specifying the previous drawing state for the current graphics port. You may obtain this value from the `outState` parameter of `GetThemeDrawingState` (page 80).
function result	A result code; see "Result Codes" (page 238).

VERSION NOTES

Available with Appearance Manager 1.1 and later.

SEE ALSO

The function `SetThemeDrawingState` (page 84).

GetThemeAccentColors

Obtains a copy of a theme's accent colors.

```
pascal OSStatus GetThemeAccentColors (
                        CTabHandle *outColors);
```

`outColors`	A pointer to a value of type `CTabHandle`. On return, the handle refers to a `ColorTable` structure containing the current accent colors.

function result	A result code; see "Result Codes" (page 238). `GetThemeAccentColors` returns the result `appearanceThemeHasNoAccents` if the current theme has no accent colors.

SPECIAL CONSIDERATIONS

Note that the Appearance Manager does not currently define semantics for any indexes into the color table produced by the `GetThemeAccentColors` function.

VERSION NOTES

Available with Appearance Manager 1.0.1 and later.

GetThemeBrushAsColor

Obtains the color that corresponds to a given theme brush type under the current theme.

```
pascal OSStatus GetThemeBrushAsColor (
                     ThemeBrush inBrush,
                     SInt16 inDepth,
                     Boolean inColorDev,
                     RGBColor *outColor);
```

`inBrush`	A value of type `ThemeBrush`. Pass a constant specifying the theme brush type for which you wish to obtain a color; see "Theme Brush Constants" (page 180) for descriptions of possible values.
`inDepth`	A signed 16-bit integer. Pass a value specifying the bit depth (in bits per pixel) of the current graphics port.
`inColorDev`	A value of type `Boolean`. Pass `true` to indicate that you are drawing on a color device. Pass `false` for a monochrome device.
`outColor`	A pointer to a structure of type `RGBColor`. On return, the structure contains a color corresponding to the color or pattern used by the specified theme brush under the current theme.

function result A result code; see “Result Codes” (page 238).

DISCUSSION

The `GetThemeBrushAsColor` function obtains a color that corresponds to that which is in use for a specified theme brush. If, in the current theme, the specified brush draws with a pattern instead of a color, a theme-specified approximate color is obtained. Your application should call `GetThemeBrushAsColor` only when you must use an `RGBColor` value for a specific operation; typically, your application should call the functions `SetThemeBackground` (page 83) and `SetThemePen` (page 85) for greatest fidelity with the current theme.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

GetThemeDrawingState

Obtains the drawing state of the current graphics port.

```
pascal OSStatus GetThemeDrawingState (
                        ThemeDrawingState *outState);
```

`outState` A pointer to a value of type `ThemeDrawingState` (page 165). On return, `GetThemeDrawingState` sets the `outState` parameter to point to a copy of the drawing state for the current graphics port.

function result A result code; see “Result Codes” (page 238).

DISCUSSION

Your application may call the `GetThemeDrawingState` function before performing an operation that modifies the drawing state of a graphics port. To return the graphics port to its previous drawing state and release the memory allocated for the drawing state reference, your application should call `SetThemeDrawingState` (page 84), providing the reference obtained in the `outState` parameter of

`GetThemeDrawingState`. You can also call `DisposeThemeDrawingState` (page 78) to release the allocated memory.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

GetThemeTextColor

Obtains the text color used for a specified element under the current theme.

```
pascal OSStatus GetThemeTextColor (
                    ThemeTextColor inColor,
                    SInt16 inDepth,
                    Boolean inColorDev,
                    RGBColor *outColor);
```

`inColor`	A value of type `ThemeTextColor`. Pass a constant specifying the element for which you wish to obtain the current text color; see “Theme Text Color Constants” (page 225) for descriptions of possible values.
`inDepth`	A signed 16-bit integer. Pass a value specifying the bit depth (in bits per pixel) of the current graphics port.
`inColorDev`	A value of type `Boolean`. Pass `true` to indicate that you are drawing on a color device. Pass `false` for a monochrome device.
`outColor`	A pointer to a structure of type `RGBColor`. On return, the structure contains the text color used for the specified element under the current theme.
function result	A result code; see “Result Codes” (page 238).

VERSION NOTES

Available with Appearance Manager 1.1 and later.

SEE ALSO

The function `SetThemeTextColor` (page 86).

IsThemeInColor

Returns whether the current theme would draw in color in the given environment.

```
pascal Boolean IsThemeInColor (
                        SInt16 inDepth,
                        Boolean inIsColorDevice);
```

`inDepth`	A signed 16-bit integer. Pass a value specifying the bit depth (in bits per pixel) of the current graphics port.
`inIsColorDevice`	A value of type `Boolean`. Pass `true` to indicate that you are drawing on a color device, or `false` for a monochrome device.
function result	A value of type `Boolean`. `IsThemeInColor` returns `true` if, given the depth and color device information, the theme would draw in color; otherwise, `false`.

DISCUSSION

To be consistent with the current theme, your application can call the `IsThemeInColor` function to determine whether or not the Appearance Manager is drawing the theme in color or black and white. If the function returns `true`, you should draw in color; if it returns `false`, you should draw in black and white. Note that the Appearance Manager may draw a theme in black and white not only because of the current bit depth or device type, but also because the theme may have defined black-and-white elements, such as the "Black & White" accent color in the platinum appearance.

VERSION NOTES

Available with Appearance Manager 1.0.1 and later.

NormalizeThemeDrawingState

Sets the current graphics port to a default drawing state.

```
pascal OSStatus NormalizeThemeDrawingState (void);
```

function result A result code; see "Result Codes" (page 238).

DISCUSSION

The `NormalizeThemeDrawingState` function sets the background of a graphics port to white; the pen of the port to a size of 1 pixel by 1 pixel, a pattern mode of `patCopy`, and a pattern of black; and the text mode of the port to `srcOr`. `NormalizeThemeDrawingState` also flushes from memory any color foreground or background patterns saved in the port's `GrafPort.pnPat` or `GrafPort.bkPat` fields, respectively.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

SetThemeBackground

Applies a theme-compliant color or pattern to the background of the current port.

```
pascal OSStatus SetThemeBackground (
                    ThemeBrush inBrush,
                    SInt16 inDepth,
                    Boolean inIsColorDevice);
```

`inBrush` A value of type `ThemeBrush`. Pass a constant specifying the theme brush to which to set the background; see "Theme Brush Constants" (page 180) for descriptions of possible values.

`inDepth` A signed 16-bit integer. Pass a value specifying the bit depth (in bits per pixel) of the current graphics port.

`inIsColorDevice`
A value of type `Boolean`. Pass `true` to indicate that you are drawing on a color device. Pass `false` for a monochrome device.

function result A result code. The result code `appearanceBadBrushIndexErr` indicates that the brush constant passed was not valid. For other possible result codes, see "Result Codes" (page 238).

DISCUSSION

Your application should call the `SetThemeBackground` function each time you wish to draw in a specified brush type. Note that the `SetThemeBackground` function aligns patterns with 0,0 in the current port. To align a pattern independently of the port origin, use the function `ApplyThemeBackground` (page 76).

Because the constant in the `inBrush` parameter can specify a color or pattern, depending on the current theme, your application must save and restore the current drawing state of the graphics port around calls to `SetThemeBackground`. Under Appearance Manager 1.1 and later, you can use the functions `GetThemeDrawingState` (page 80) and `SetThemeDrawingState` (page 84) to do this.

Prior to Appearance Manager 1.1, you must save and restore the `pnPixPat` and `bkPixPat` fields of your graphics port when saving the text and background colors. Because patterns in the `bkPixPat` field override the background color of the window, call the QuickDraw function `BackPat` to set your background pattern to a normal white pattern. This ensures that you can use `RGBBackColor` to set your background color to white, call the QuickDraw function `EraseRect`, and get the expected results.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

SetThemeDrawingState

Sets the drawing state of the current graphics port.

```
pascal OSStatus SetThemeDrawingState (
                    ThemeDrawingState inState,
                    Boolean inDisposeNow);
```

`inState` A value of type `ThemeDrawingState` (page 165). Pass a `ThemeDrawingState` value such as that produced in the `outState` parameter of `GetThemeDrawingState` (page 80).

`inDisposeNow`	A value of type `Boolean`. Pass a value of `true` to release the memory allocated for the drawing state reference. Pass `false` if you wish to continue using the drawing state and do not want to dispose of the memory at this time; you must call `DisposeThemeDrawingState` (page 78) to dispose of the memory any time before your application terminates.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

Your application can save the port state by calling the function `GetThemeDrawingState` (page 80) and restore the port state by calling the function `SetThemeDrawingState`, supplying the value obtained in the `outState` parameter of `GetThemeDrawingState`, after you have completed all of your drawing.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

SetThemePen

Applies a theme-compliant color or pattern to the foreground of the current port.

```
pascal OSStatus SetThemePen (
                    ThemeBrush inBrush,
                    SInt16 inDepth,
                    Boolean inIsColorDevice);
```

`inBrush`	A value of type `ThemeBrush`. Pass a constant specifying the theme brush type to which to set the pen; see "Theme Brush Constants" (page 180) for descriptions of possible values.
`inDepth`	A signed 16-bit integer. Pass a value specifying the bit depth (in bits per pixel) of the current graphics port.
`inIsColorDevice`	A value of type `Boolean`. Pass `true` to indicate that you are drawing on a color device. Pass `false` for a monochrome device.

function result	A result code. The result code `appearanceBadBrushIndexErr` indicates that the brush constant passed in was not valid. For other possible result codes, see "Result Codes" (page 238).

DISCUSSION

Your application should call the `SetThemePen` function each time you wish to draw an element in a specified brush constant.

Because the constant in the `inBrush` parameter can represent a color or pattern, depending on the current theme, your application must save and restore the current drawing state of the graphics port around calls to `SetThemePen`. Under Appearance Manager 1.1 and later, you can use the functions `GetThemeDrawingState` (page 80) and `SetThemeDrawingState` (page 84) to do this. Prior to Appearance Manager 1.1, you must save and restore the `pnPixPat` and `bkPixPat` fields of your graphics port when saving the text and background colors. Because patterns in the `pnPixPat` field override the foreground color of the window, call the QuickDraw function `PenPat` to set your foreground pattern to a normal white pattern. This ensures that you can use `RGBForeColor` to set your foreground color to white, call the QuickDraw function `PaintRect`, and get the expected results.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

SetThemeTextColor

Sets the current text color to be consistent with that of a specified element.

```
pascal OSStatus SetThemeTextColor (
                    ThemeTextColor inColor,
                    SInt16 inDepth,
                    Boolean inIsColorDevice);
```

`inColor`	A value of type `ThemeTextColor`. Pass a constant specifying an interface element. `SetThemeTextColor` sets the current text color to be the same as the color of that element's text. See "Theme Text Color Constants" (page 225) for descriptions of possible values.

`inDepth`	A signed 16-bit integer. Pass a value specifying the bit depth (in bits per pixel) of the current graphics port.
`inIsColorDevice`	A value of type `Boolean`. Pass `true` to indicate that you are drawing on a color device. Pass `false` for a monochrome device.
function result	A result code. The result code `appearanceBadTextColorIndexErr` indicates that the text color index passed was not valid. For other possible result codes, see "Result Codes" (page 238).

DISCUSSION

Your application typically uses the `SetThemeTextColor` function inside a `DeviceLoop` drawing procedure to set the foreground color to a theme-compliant value.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

SEE ALSO

The function `GetThemeTextColor` (page 81).

Playing Theme Sounds

The Appearance Manager provides the following functions for playing theme sounds:

- `PlayThemeSound` (page 89) plays an asynchronous sound associated with the specified state change.
- `BeginThemeDragSound` (page 88) continuously plays a theme-specific sound associated with the user's movement of a given interface object.
- `EndThemeDragSound` (page 88) terminates the playing of a sound associated with the user's movement of a given interface object.

BeginThemeDragSound

Continuously plays a theme-specific sound associated with the user's movement of a given interface object.

```
pascal OSStatus BeginThemeDragSound (
                        ThemeDragSoundKind kind);
```

`kind`	A value of type `ThemeDragSoundKind`. Pass a constant specifying the sound to play; see "Theme Drag Sound Constants" (page 197) for descriptions of possible values.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

The Appearance Manager automatically plays drag sounds for standard user interface elements and for Drag Manager drag actions. Your application may call `BeginThemeDragSound`, typically upon detecting a drag initiation, to play a drag sound for a custom element. `BeginThemeDragSound` plays the specified sound in a continuous loop until your application calls the function `EndThemeDragSound` (page 88), typically upon receiving a mouse-up event.

Note that the `BeginThemeDragSound` function automatically tracks the current mouse position and handles any panning or variations in pitch for the sound.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

EndThemeDragSound

Terminates the playing of a sound associated with the user's movement of a given interface object.

```
pascal OSStatus EndThemeDragSound (void);
```

function result	A result code; see "Result Codes" (page 238).

DISCUSSION

The Appearance Manager automatically starts and stops drag sounds for standard user interface elements and for Drag Manager drag actions. Your application may call `BeginThemeDragSound` (page 88), typically upon detecting a drag initiation, to play a drag sound for a custom element. Call the `EndThemeDragSound` function to turn off a drag sound when the drag is completed, typically upon receipt of a mouse-up event.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

PlayThemeSound

Plays an asynchronous sound associated with the specified state change.

```
pascal OSStatus PlayThemeSound (
                        ThemeSoundKind kind);
```

`kind`	A value of type `ThemeSoundKind`. Pass a constant specifying the sound to play; see "Theme Sound Constants" (page 208) for descriptions of possible values.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

The Appearance Manager automatically plays theme sounds for standard user interface elements. Your application can call the `PlayThemeSound` function to play a theme sound for a custom element. The sound plays asynchronously until complete, stopping automatically.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

Specifying Theme-Compliant Cursors

The Appearance Manager provides the following functions for specifying theme-compliant cursors:

- `SetThemeCursor` (page 91) sets the cursor to a version of the specified cursor type that is consistent with the current theme.
- `SetAnimatedThemeCursor` (page 90) animates a version of the specified cursor type that is consistent with the current theme.

SetAnimatedThemeCursor

Animates a version of the specified cursor type that is consistent with the current theme.

```
pascal OSStatus SetAnimatedThemeCursor (
                    ThemeCursor inCursor,
                    UInt32 inAnimationStep);
```

`inCursor` A value of type `ThemeCursor`. Pass a constant specifying the type of cursor to set; see "Theme Cursor Constants" (page 194) for a description of the possible values. Note that only cursors designated as able to be animated should be used for this function. If you specify an unanimatable cursor type, `SetAnimatedThemeCursor` returns the error `themeBadCursorIndexErr` (–30565).

`inAnimationStep`
An unsigned 32-bit value. Pass a value specifying the current animation step of the cursor. To animate the cursor, increment the value by 1 with each call to `SetAnimatedThemeCursor`.

function result A result code; see "Result Codes" (page 238).

DISCUSSION

Appearance Manager 1.1 introduces cursors that can change appearance with a theme change. In order to be theme-compliant, your program should use these theme-specific cursors whenever possible, instead of the classic black-and-white cursors.

Your application should call the `SetAnimatedThemeCursor` function to ensure that its animated cursors are theme-compliant, rather than using any QuickDraw cursor utilities functions such as `SetCursor`, `SetCCursor`, `SpinCursor`, or `RotateCursor`. If you wish a non-animated cursor to be theme-compliant, call the function `SetThemeCursor` (page 91).

Because these are color cursors, they currently cannot be set from interrupt time. Therefore, if you support animated cursors that are changed at interrupt time you should continue to use your own cursors for now.

SPECIAL CONSIDERATIONS

Do not call `SetAnimatedThemeCursor` at interrupt time.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

SEE ALSO

The function `SetThemeCursor` (page 91).

SetThemeCursor

Sets the cursor to a version of the specified cursor type that is consistent with the current theme.

```
pascal OSStatus SetThemeCursor (
                        ThemeCursor inCursor);
```

`inCursor`	A value of type `ThemeCursor`. Pass a constant specifying the type of cursor to set; see "Theme Cursor Constants" (page 194) for a description of possible values.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

Appearance Manager 1.1 introduces cursors that can change appearance with a theme change. In order to be theme-compliant, your program should use these theme-specific cursors whenever possible, instead of the classic black-and-white cursors. Because these are color cursors, they currently cannot be set from interrupt time.

Your application should call the `SetThemeCursor` function to ensure that its cursors are theme-compliant, rather than the QuickDraw cursor utilities functions `SetCursor` or `SetCCursor`. If you wish an animatable cursor to be theme-compliant, call the function `SetAnimatedThemeCursor` (page 90).

SPECIAL CONSIDERATIONS

Do not call `SetThemeCursor` at interrupt time.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

Drawing Theme-Compliant Controls

The Appearance Manager provides the following functions for drawing theme-compliant buttons:

- `DrawThemeButton` (page 94) draws a button.
- `DrawThemePopupArrow` (page 102) draws a pop-up arrow.
- `GetThemeButtonBackgroundBounds` (page 113) obtains the rectangle that contains a button.
- `GetThemeButtonContentBounds` (page 114) obtains the rectangle where content can be drawn for a button.
- `GetThemeButtonRegion` (page 115) obtains the region occupied by a button.

The Appearance Manager provides the following functions for drawing theme-compliant tabs:

- `DrawThemeTab` (page 107) draws a tab.
- `DrawThemeTabPane` (page 109) draws a tab pane.
- `GetThemeTabRegion` (page 119) obtains the region occupied by a tab.

The Appearance Manager provides the following functions for drawing theme-compliant tracks, such as sliders and scroll bars:

- `DrawThemeTrack` (page 110) draws a track.
- `DrawThemeTrackTickMarks` (page 111) draws tick marks for a track.
- `DrawThemeTickMark` (page 109) draws a tick mark.
- `DrawThemeScrollBarArrows` (page 104) draws scroll bar arrows consistent with the current system preferences.
- `GetThemeTrackBounds` (page 120) obtains the bounding rectangle of a track.
- `GetThemeTrackDragRect` (page 121) obtains the area in which the user may drag a track's indicator.
- `GetThemeTrackLiveValue` (page 122) obtains the current value of a track's indicator, given its relative position.
- `GetThemeTrackThumbPositionFromOffset` (page 123) obtains the relative position of a track's indicator, given an offset from its prior position.
- `GetThemeTrackThumbPositionFromRegion` (page 124) obtains the relative position of a track's indicator, given its current position.
- `GetThemeTrackThumbRgn` (page 124) obtains the region containing a track's indicator.
- `GetThemeScrollBarTrackRect` (page 118) obtains the area containing the track portion of a scroll bar.
- `HitTestThemeTrack` (page 127) returns whether the user clicked upon the specified track.
- `HitTestThemeScrollBarArrows` (page 125) returns whether the user clicked upon the specified scroll bar's arrows.

The Appearance Manager provides the following functions for determining the current system preferences for control style settings:

- `GetThemeCheckBoxStyle` (page 116) obtains the system preference for the type of mark to use in a checkbox.
- `GetThemeScrollBarArrowStyle` (page 116) obtains the system preference for the type of scroll bar arrows to be used.
- `GetThemeScrollBarThumbStyle` (page 117) obtains the system preference for the type of scroll box to be used.

The Appearance Manager provides the following other functions for drawing theme-compliant custom controls:

- `DrawThemeChasingArrows` (page 96) draws an asynchronous arrows indicator.
- `DrawThemeEditTextFrame` (page 97) draws an editable text frame.
- `DrawThemeFocusRect` (page 98) draws or erases a focus ring around a specified rectangle.
- `DrawThemeFocusRegion` (page 99) draws or erases a focus ring around a specified region.
- `DrawThemeGenericWell` (page 100) draws an image well frame.
- `DrawThemeListBoxFrame` (page 100) draws a list box frame.
- `DrawThemePlacard` (page 101) draws a placard.
- `DrawThemePrimaryGroup` (page 103) draws a primary group box frame.
- `DrawThemeSecondaryGroup` (page 105) draws a secondary group box frame.
- `DrawThemeSeparator` (page 106) draws a separator line.

DrawThemeButton

Draws a button.

```
pascal OSStatus DrawThemeButton (
                    const Rect *inBounds,
                    ThemeButtonKind inKind,
                    const ThemeButtonDrawInfo *inNewInfo,
                    const ThemeButtonDrawInfo *inPrevInfo,
                    ThemeEraseUPP inEraseProc,
                    ThemeButtonDrawUPP inLabelProc,
                    UInt32 inUserData);
```

`inBounds`	A pointer to a structure of type `Rect`. Pass a rectangle specifying the boundary of the button, in local coordinates.
`inKind`	A value of type `ThemeButtonKind`. Pass a constant specifying the type of button to draw. See "Theme Button Kind Constants" (page 187) for descriptions of possible values.
`inNewInfo`	A pointer to a structure of type `ThemeButtonDrawInfo` (page 165). Before calling `DrawThemeButton`, set the structure to contain the new state, value, and adornment for the button. `DrawThemeButton` uses the information passed in the `inNewInfo` and `inPrevInfo`

	parameters to apply transitional animation or sound effects as the button state changes, if such are specified under the current theme.
`inPrevInfo`	A pointer to a structure of type `ThemeButtonDrawInfo` (page 165). If the button state is changing, set the structure to contain the previous state, value, and adornment for the button, to allow `DrawThemeButton` to apply any transitional effects. If the button state is not changing, you can pass `NULL`.
`inEraseProc`	A value of type `ThemeEraseUPP` (page 171). If you have a custom background, pass a universal procedure pointer to an application-defined function such as that described in `MyThemeEraseProc` (page 159). `DrawThemeButton` calls that function to erase the background before drawing the button. If you pass `NULL`, no erasing occurs.
`inLabelProc`	A value of type `ThemeButtonDrawUPP` (page 173). If you pass a universal procedure pointer to an application-defined function such as that described in `MyThemeButtonDrawProc` (page 157), `DrawThemeButton` calls that function to draw the label of the button. If you pass `NULL`, no label is drawn.
`inUserData`	An unsigned 32-bit integer. Provide any data to be passed in to the callback functions specified in the `inLabelProc` and `inEraseProc` parameters. Pass `NULL` if you do not wish to provide any data.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

The `DrawThemeButton` function draws a theme-compliant button. If a `ThemeEraseProcPtr` is specified in the `inEraseProc` parameter, `DrawThemeButton` uses that function to erase the background of the button before drawing the button. After the button is drawn, if a `ThemeButtonDrawProcPtr` is specified in the `inLabelProc` parameter, `DrawThemeButton` calls that function to draw the button's label.

Note that `DrawThemeButton` also draws any appearance adornments for the button and that these can extend beyond the button's basic bounding rectangle, as specified in the `inBounds` parameter, and may be of variable shape. You may therefore wish to call the function `GetThemeButtonBackgroundBounds` (page 113) to obtain the actual rectangle containing the pixels belonging to a button under the current theme.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

DrawThemeChasingArrows

Draws an asynchronous arrows indicator.

```
pascal OSStatus DrawThemeChasingArrows (
                    const Rect *bounds,
                    UInt32 index,
                    ThemeDrawState state,
                    ThemeEraseUPP eraseProc,
                    UInt32 eraseData);
```

`bounds`	A pointer to a structure of type `Rect`. Before calling `DrawThemeChasingArrows`, set the rectangle to contain the asynchronous arrows, in local coordinates.
`index`	An unsigned 32-bit value. Pass a value specifying the current animation step of the arrows. To animate the arrows, increment the initial value by 1 with each call to `DrawThemeChasingArrows`.
`state`	A value of type `ThemeDrawState`. Pass a constant specifying the state in which to draw the asynchronous arrows indicator; see “Theme Draw State Constants” (page 199). The asynchronous arrows indicator can be drawn as active or inactive; passing `kThemeStatePressed` produces an error.
`eraseProc`	A value of type `ThemeEraseUPP` (page 171). If you have a custom background, pass a universal procedure pointer to an application-defined function such as that described in `MyThemeEraseProc` (page 159). `DrawThemeChasingArrows` calls that function to erase the background before drawing the asynchronous arrows. If you pass `NULL`, no erasing occurs.
`eraseData`	An unsigned 32-bit integer. Provide any data to be passed in to the `eraseData` parameter of the callback function specified in the `eraseProc` parameter.

function result A result code; see "Result Codes" (page 238).

DISCUSSION

The `DrawThemeChasingArrows` function draws a theme-compliant asynchronous arrows (also known as "chasing arrows") indicator.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

DrawThemeEditTextFrame

Draws an editable text frame.

```
pascal OSStatus DrawThemeEditTextFrame (
                    const Rect *inRect,
                    ThemeDrawState inState);
```

`inRect`	A pointer to a structure of type `Rect`. Before calling `DrawThemeEditTextFrame`, set the rectangle to the position around which to draw the editable text frame, in local coordinates.
`inState`	A value of type `ThemeDrawState`. Pass a constant specifying the state in which to draw the editable text frame; see "Theme Draw State Constants" (page 199). The frame can be drawn as active or inactive; passing `kThemeStatePressed` produces an error.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

The `DrawThemeEditTextFrame` function draws a theme-compliant frame for an editable text field. The frame is a maximum of 2 pixels thick and is drawn outside the specified rectangle. You should not use this function to draw frames for items other than editable text fields.

To ensure that you get an appropriate focus ring for your editable text field, you should pass the same rectangle that you use with `DrawThemeEditTextFrame` function to the function `DrawThemeFocusRect` (page 98).

VERSION NOTES

Available with Appearance Manager 1.0 and later.

DrawThemeFocusRect

Draws or erases a focus ring around a specified rectangle.

```
pascal OSStatus DrawThemeFocusRect (
                    const Rect *inRect,
                    Boolean inHasFocus);
```

`inRect`	A pointer to a structure of type `Rect`. Before calling `DrawThemeFocusRect`, set the rectangle to the position around which to draw the focus ring, in local coordinates. The focus ring is drawn outside the rectangle that is passed in, and it can be outset a maximum of 3 pixels.
`inHasFocus`	A value of type `Boolean`. Pass `true` to draw the focus ring. Pass `false` to erase the focus ring.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

Your application can use the `DrawThemeFocusRect` function to draw a theme-compliant focus ring. The presence of a focus ring indicates whether an item has keyboard focus.

SPECIAL CONSIDERATIONS

If you are drawing a focus ring around an element for which you have drawn a frame using `DrawThemeEditTextFrame` (page 97) or `DrawThemeListBoxFrame` (page 100), you must coordinate your drawing sequence to achieve the correct look. When drawing the element, your application 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 editable text frame or list box frame, you must redraw the editable text frame or list box frame after calling `DrawThemeFocusRect`, because there is typically an overlap.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

DrawThemeFocusRegion

Draws or erases a focus ring around a specified region.

```
pascal OSStatus DrawThemeFocusRegion (
                     RgnHandle inRegion,
                     Boolean inHasFocus);
```

`inRegion`	A value of type `RgnHandle`. Before calling `DrawThemeFocusRegion`, set the region to the position around which to draw the focus ring, in local coordinates. The focus ring is drawn outside the region that is passed in, and it can be outset a maximum of 3 pixels.
`inHasFocus`	A value of type `Boolean`. Pass `true` to draw the focus region. Pass `false` to erase the focus region.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

Your application can use the `DrawThemeFocusRegion` function to draw a theme-compliant focus ring. The presence of a focus ring indicates whether an item has keyboard focus.

VERSION NOTES

Available with Appearance Manager 1.0.1 and later.

DrawThemeGenericWell

Draws an image well frame.

```
pascal OSStatus DrawThemeGenericWell (
                    const Rect *inRect,
                    ThemeDrawState inState,
                    Boolean inFillCenter);
```

`inRect`	A pointer to a structure of type `Rect`. Before calling `DrawThemeGenericWell`, set the rectangle to the position around which to draw the image well frame, in local coordinates.
`inState`	A value of type `ThemeDrawState`. Pass a constant specifying the state in which to draw the image well frame; see "Theme Draw State Constants" (page 199). The well can be drawn as active or inactive; passing `kThemeStatePressed` produces an error.
`inFillCenter`	A value of type `Boolean`. Set to `true` to fill the image well frame with white; otherwise, `false`.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

The `DrawThemeGenericWell` function draws a theme-compliant image well frame. You can specify that the center of the well be filled in with white.

VERSION NOTES

Available with Appearance Manager 1.0.1 and later.

DrawThemeListBoxFrame

Draws a list box frame.

```
pascal OSStatus DrawThemeListBoxFrame (
                    const Rect *inRect,
                    ThemeDrawState inState);
```

`inRect`	A pointer to a structure of type `Rect`. Before calling `DrawThemeListBoxFrame`, set the rectangle to the position around which to draw the list box frame, in local coordinates.
`inState`	A value of type `ThemeDrawState`. Pass a constant specifying the state in which to draw the list box frame; see "Theme Draw State Constants" (page 199). The frame can be drawn as active or inactive; passing `kThemeStatePressed` produces an error.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

The `DrawThemeListBoxFrame` function draws a theme-compliant list box frame. The frame is a maximum of 2 pixels thick and is drawn outside the specified rectangle. To ensure that you get an appropriate focus ring for your list box, you should pass the same rectangle that you use with the `DrawThemeListBoxFrame` function to the function `DrawThemeFocusRect` (page 98).

VERSION NOTES

Available with Appearance Manager 1.0 and later.

DrawThemePlacard

Draws a placard.

```
pascal OSStatus DrawThemePlacard (
                        const Rect *inRect,
                        ThemeDrawState inState);
```

`inRect`	A pointer to a structure of type `Rect`. Before calling `DrawThemePlacard`, set the rectangle to a size and position that contains the placard, in local coordinates.
`inState`	A value of type `ThemeDrawState`. Pass a constant specifying the state in which to draw the placard; see "Theme Draw State Constants" (page 199). The placard can be drawn as active, inactive, or pressed.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

The `DrawThemePlacard` function draws a theme-compliant placard inside the specified rectangle.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

DrawThemePopupArrow

Draws a pop-up arrow.

```
pascal OSStatus DrawThemePopupArrow (
                    const Rect *bounds,
                    ThemeArrowOrientation orientation,
                    ThemePopupArrowSize size,
                    ThemeDrawState state,
                    ThemeEraseUPP eraseProc,
                    UInt32 eraseData);
```

`bounds`	A pointer to a structure of type `Rect`. Before calling `DrawThemePopupArrow`, set the rectangle to contain the arrow, in local coordinates. `DrawThemePopupArrow` positions the arrow relative to the top left corner of the rectangle.
`orientation`	A value of type `ThemeArrowOrientation`. Pass a constant specifying the direction in which the pop-up arrow points. See "Theme Pop-Up Arrow Orientation Constants" (page 204) for descriptions of possible values.
`size`	A value of type `ThemePopupArrowSize`. Pass a constant specifying the size of the pop-up arrow to draw. See "Theme Pop-Up Arrow Size Constants" (page 205) for descriptions of possible values.
`state`	A value of type `ThemeDrawState`. Pass a constant specifying the current state of the button containing the pop-up arrow. See "Theme Draw State Constants" (page 199) for descriptions of possible values.

`eraseProc`	A value of type `ThemeEraseUPP` (page 171). If you have a custom background, pass a universal procedure pointer to an application-defined function such as that described in `MyThemeEraseProc` (page 159). `DrawThemePopupArrow` calls that function to erase the background before drawing the pop-up arrow. If you pass `NULL`, no erasing occurs.
`eraseData`	An unsigned 32-bit integer. Provide any data to be passed in to the `eraseData` parameter of the callback function specified in the `eraseProc` parameter.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

The `DrawThemePopupArrow` function draws a theme-compliant pop-up arrow. A pop-up arrow is an image drawn onto another control to indicate that when the control is clicked, you get a pop-up menu. A pop-up arrow is not a separate button itself. Typically, a pop-up arrow is used in conjunction with a button, such as a push button or bevel button. Bevel button controls automatically draw a pop-up arrow if a menu is associated with the control.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

DrawThemePrimaryGroup

Draws a primary group box frame.

```
pascal OSStatus DrawThemePrimaryGroup (
                    const Rect *inRect,
                    ThemeDrawState inState);
```

`inRect`	A pointer to a structure of type `Rect`. Before calling `DrawThemePrimaryGroup`, set the rectangle to the bounds of the primary group box frame, in local coordinates.

`inState`	A value of type `ThemeDrawState`. Pass a constant specifying the state in which to draw the primary group box frame; see "Theme Draw State Constants" (page 199). The frame can be drawn as active or inactive; passing `kThemeStatePressed` produces an error.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

The `DrawThemePrimaryGroup` function draws a theme-compliant primary group box frame. The primary group box frame is drawn inside the specified rectangle and is a maximum of 2 pixels thick.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

DrawThemeScrollBarArrows

Draws scroll bar arrows consistent with the current system preferences.

```
pascal OSStatus DrawThemeScrollBarArrows (
                    const Rect *bounds,
                    ThemeTrackEnableState enableState,
                    ThemeTrackPressState pressState,
                    Boolean isHoriz,
                    Rect *trackBounds);
```

`bounds`	A pointer to a structure of type `Rect`. Before calling `DrawThemeScrollBarArrows`, set the rectangle to contain the scroll bar for which to draw arrows, in local coordinates. Typically, the rectangle you specify is the entire base control rectangle—that is, the value contained in the `contrlRect` field of the scroll bar's `ControlRecord` structure.
`enableState`	A value of type `ThemeTrackEnableState`. Pass a constant specifying the current state of the scroll bar; see "Theme Track Enable State Constants" (page 231) for descriptions of possible values.

`pressState`	A value of type `ThemeTrackPressState`. Pass a constant specifying what is pressed in an active scroll bar or 0 if nothing is pressed. The press state is ignored if the scroll bar is not active. See "Theme Track Press State Constants" (page 233) for descriptions of possible values.
`isHoriz`	A value of type `Boolean`. Pass `true` if the scroll bar is horizontal; pass `false` if it is vertical.
`trackBounds`	A pointer to a structure of type `Rect`. On return, the rectangle is set to the bounds of the track portion of the scroll bar; this rectangle excludes the area containing the scroll bar arrows. Pass `NULL` if you do not wish to obtain this information.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

The `DrawThemeScrollBarArrows` function draws a set of theme-compliant scroll bar arrows for the scroll bar whose position and dimensions are specified in the `bounds` parameter. Depending upon the current system preferences, `DrawThemeScrollBarArrows` draws the arrows in one of the following configurations:

- one arrow at either end of the scroll bar
- two arrows at the same end of the scroll bar

As shown in Listing 3-7 in "Drawing Tracks" (page 55), your application can use `DrawThemeScrollBarArrows` in conjunction with the function `DrawThemeTrack` (page 110) to draw a complete scroll bar.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

DrawThemeSecondaryGroup

Draws a secondary group box frame.

```
pascal OSStatus DrawThemeSecondaryGroup (
                    const Rect *inRect,
                    ThemeDrawState inState);
```

`inRect`	A pointer to a structure of type `Rect`. Before calling `DrawThemeSecondaryGroup`, set the rectangle to the bounds of the secondary group box frame to be drawn, in local coordinates.
`inState`	A value of type `ThemeDrawState`. Pass a constant specifying the state in which to draw the secondary group box frame; see "Theme Draw State Constants" (page 199). The frame can be drawn as active or inactive; passing `kThemeStatePressed` produces an error.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

The `DrawThemeSecondaryGroup` function draws a theme-compliant secondary group box frame. The secondary group box frame is drawn inside the specified rectangle and is a maximum of 2 pixels thick. Note that a secondary group box frame is typically nested within a primary group box frame.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

DrawThemeSeparator

Draws a separator line.

```
pascal OSStatus DrawThemeSeparator (
                    const Rect *inRect,
                    ThemeDrawState inState);
```

`inRect`	A pointer to a structure of type `Rect`. Before calling `DrawThemeSeparator`, set the rectangle to contain the separator line, in local coordinates. 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.

`inState`	A value of type `ThemeDrawState`. Pass a constant specifying the state in which to draw the separator line; see "Theme Draw State Constants" (page 199). The separator line can be drawn as active or inactive; passing `kThemeStatePressed` produces an error.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

The `DrawThemeSeparator` function draws a theme-compliant separator line. The separator line is a maximum of 2 pixels thick and is drawn inside the specified rectangle.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

DrawThemeTab

Draws a tab.

```
pascal OSStatus DrawThemeTab (
                    const Rect *inRect,
                    ThemeTabStyle inStyle,
                    ThemeTabDirection inDirection,
                    ThemeTabTitleDrawUPP labelProc,
                    UInt32 userData);
```

`inRect`	A pointer to a structure of type `Rect`. Before calling `DrawThemeTab`, set the rectangle to the bounds of the tab, in local coordinates. There are two standard sizes (or heights) for tabs that should be used in your calculation of the tab rectangle—these are measured by the distance the tabs protrude from the pane. Small tabs have a height of 16 pixels; large tabs have a height of 21 pixels. (The widths of tabs are variable.) Additionally, the distance that the tab overlaps the pane must be included in the tab rectangle; this overlap distance is always 3

	pixels, although the 3-pixel overlap is only drawn for the front tab. The tab rectangle should reflect the orientation of the tab that is specified in the `inDirection` parameter.
`inStyle`	A value of type `ThemeTabStyle`. Pass a constant specifying the relative position (front or non-front) and state of the tab. See "Theme Tab Style Constants" (page 224) for descriptions of possible values.
`inDirection`	A value of type `ThemeTabDirection`. Pass a constant specifying the direction in which to orient the tab. See "Theme Tab Direction Constants" (page 223) for descriptions of possible values.
`labelProc`	A value of type `ThemeTabTitleDrawUPP` (page 172). Pass a universal procedure pointer to an application-defined function such as that described in `MyThemeTabTitleDrawProc` (page 161). `DrawThemeTab` calls your function to draw the title of the tab. If you pass `NULL`, no drawing occurs.
`userData`	An unsigned 32-bit integer. Provide any data to be passed in to the `userData` parameter of the callback function specified in the `labelProc` parameter.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

The `DrawThemeTab` function draws a theme-compliant tab. A tab control consists of two basic components: multiple tabs that label the various content pages that can be displayed and a single pane upon which the content for each tab is drawn. Use the function `DrawThemeTabPane` (page 109) to draw the tab pane. The Appearance Manager coordinates the appearance of the pane and frontmost tab automatically.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

DrawThemeTabPane

Draws a tab pane.

```
pascal OSStatus DrawThemeTabPane (
                        const Rect *inRect,
                        ThemeDrawState inState);
```

`inRect`	A pointer to a structure of type `Rect`. Before calling `DrawThemeTabPane`, set the rectangle to the bounds of the tab pane, in local coordinates.
`inState`	A value of type `ThemeDrawState`. Pass a constant specifying the state in which to draw the tab pane; see "Theme Draw State Constants" (page 199). The tab pane can be drawn as active or inactive; passing `kThemeStatePressed` produces an error.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

The `DrawThemeTabPane` function draws a theme-compliant tab pane. A tab control consists of two basic components: multiple tabs that label the various content pages that can be displayed and a single pane upon which the content for each tab is drawn. Use the function `DrawThemeTab` (page 107) to draw the tab. The Appearance Manager coordinates the appearance of the pane and frontmost tab automatically.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

DrawThemeTickMark

Draws a tick mark.

```
pascal OSStatus DrawThemeTickMark (
                        const Rect *bounds,
                        ThemeDrawState state);
```

`bounds`	A pointer to a structure of type `Rect`. Before calling `DrawThemeTickMark`, set the rectangle to the position that contains the tick mark, in local coordinates. Note that tick marks are of a fixed—3 pixel by 8 pixel—size.
`state`	A value of type `ThemeDrawState`. Pass a constant specifying the state in which to draw the tick mark; see "Theme Draw State Constants" (page 199). The tick mark can be drawn as active or inactive; passing `kThemeStatePressed` produces an error.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

The `DrawThemeTickMark` function draws a single theme-compliant tick mark. To draw a complete set of tick marks for a track, call the function `DrawThemeTrackTickMarks` (page 111).

VERSION NOTES

Available with Appearance Manager 1.1 and later.

DrawThemeTrack

Draws a track.

```
pascal OSStatus DrawThemeTrack (
                    const ThemeTrackDrawInfo *drawInfo,
                    RgnHandle rgnGhost,
                    ThemeEraseUPP eraseProc,
                    UInt32 eraseData);
```

`drawInfo`	A pointer to a structure of type `ThemeTrackDrawInfo` (page 166). Before calling `DrawThemeTrack`, set the structure to contain the current visual characteristics of the track.
`rgnGhost`	A value of type `RgnHandle`. If the track is of a type that contains an indicator, such as a scroll bar or slider, you may pass a handle to the region where `DrawThemeTrack` is to draw a ghost image of

	the track indicator. Your application should only use a ghost image with the indicator when a track does not support live feedback. Pass `NULL` if you do not want to draw a ghost image.
`eraseProc`	A value of type `ThemeEraseUPP` (page 171). If you have a custom background, pass a universal procedure pointer to an application-defined function such as that described in `MyThemeEraseProc` (page 159). `DrawThemeTrack` calls that function to erase the background before drawing the track. If you pass `NULL`, no erasing occurs.
`eraseData`	An unsigned 32-bit integer. Provide any data to be passed in to the callback function specified in the `eraseProc` parameter.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

Your application may use the `DrawThemeTrack` function to draw a theme-compliant slider, progress bar, or scroll bar. If you use `DrawThemeTrack` to draw a scroll bar, use the function `DrawThemeScrollBarArrows` (page 104) to draw the scroll bar's arrows; see Listing 3-7 in "Drawing Tracks" (page 55) for an example of using these two functions together to draw a complete scroll bar. If you use `DrawThemeTrack` to draw a slider, use `DrawThemeTrackTickMarks` (page 111) to draw any tick marks for the slider.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

DrawThemeTrackTickMarks

Draws tick marks for a track.

```
pascal OSStatus DrawThemeTrackTickMarks (
                     const ThemeTrackDrawInfo *drawInfo,
                     ItemCount numTicks,
                     ThemeEraseUPP eraseProc,
                     UInt32 eraseData);
```

`drawInfo`	A pointer to a structure of type `ThemeTrackDrawInfo` (page 166). Before calling `DrawThemeTrackTickMarks`, set the structure to describe the current visual characteristics of the track. Because, under Appearance Manager 1.1, sliders are the only track type to support tick marks, you should set the `kind` field of the `ThemeTrackDrawInfo` structure to `kThemeSlider` and fill out the remainder of the structure appropriately for a slider track. You should set the `bounds` field of the `ThemeTrackDrawInfo` structure to the boundary of the track itself, not including the area that contains the tick marks; you can obtain the actual bounding rectangle of the track by calling the function `GetThemeTrackBounds` (page 120). `DrawThemeTrackTickMarks` draws the tick marks outside the track's bounding rectangle, above or below the track depending on the thumb direction indicated by the `drawInfo.trackInfo.slider.thumbDir` field.
`numTicks`	A value of type `ItemCount`. Pass an unsigned 32-bit value specifying the number of tick marks to be drawn.
`eraseProc`	A value of type `ThemeEraseUPP` (page 171). If you have a custom background, pass a universal procedure pointer to an application-defined function such as that described in `MyThemeEraseProc` (page 159). `DrawThemeTrackTickMarks` calls that function to erase the background before drawing tick marks. If you pass `NULL`, no erasing occurs.
`eraseData`	An unsigned 32-bit integer. Provide any data to be passed in to the callback function specified in the `eraseProc` parameter.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

Your application can call the `DrawThemeTrackTickMarks` function to draw theme-compliant tick marks for a slider control. (Under Appearance Manager 1.1, sliders are the only track type that supports tick marks.) To draw a track control, call the function `DrawThemeTrack` (page 110). To draw a single tick mark, call the function `DrawThemeTickMark` (page 109).

VERSION NOTES

Available with Appearance Manager 1.1 and later.

GetThemeButtonBackgroundBounds

Obtains the rectangle that contains a button.

```
pascal OSStatus GetThemeButtonBackgroundBounds (
                    const Rect *inBounds,
                    ThemeButtonKind inKind,
                    const ThemeButtonDrawInfo *inDrawInfo,
                    Rect *outBounds);
```

`inBounds`	A pointer to a structure of type `Rect`. Before calling `GetThemeButtonBackgroundBounds`, set the rectangle to the boundary of the button without any adornments, in local coordinates.
`inKind`	A value of type `ThemeButtonKind`. Pass a constant specifying the type of button being examined. See "Theme Button Kind Constants" (page 187) for descriptions of possible values.
`inDrawInfo`	A pointer to a structure of type `ThemeButtonDrawInfo` (page 165). Before calling `GetThemeButtonBackgroundBounds`, set the structure to contain the state, value, and adornment for the button.
`outBounds`	A pointer to a structure of type `Rect`. On return, the rectangle contains the actual boundary of the button, including any adornments, in local coordinates.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

Appearance adornments can extend beyond the basic bounding rectangle of a button and may be of variable shape. Your application may call the `GetThemeButtonBackgroundBounds` function to obtain the actual rectangle containing the pixels belonging to a button under the current theme.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

GetThemeButtonContentBounds

Obtains the rectangle where content can be drawn for a button.

```
pascal OSStatus GetThemeButtonContentBounds (
                    const Rect *inBounds,
                    ThemeButtonKind inKind,
                    const ThemeButtonDrawInfo *inDrawInfo,
                    Rect *outBounds);
```

`inBounds`	A pointer to a structure of type `Rect`. Before calling `GetThemeButtonContentBounds`, set the rectangle to contain the boundary of the button, in local coordinates.
`inKind`	A value of type `ThemeButtonKind`. Pass a constant specifying the type of button being examined. See “Theme Button Kind Constants” (page 187) for descriptions of possible values.
`inDrawInfo`	A pointer to a structure of type `ThemeButtonDrawInfo` (page 165). Before calling `GetThemeButtonContentBounds`, set the structure to contain the state, value, and adornment for the button.
`outBounds`	A pointer to a structure of type `Rect`. On return, the rectangle contains the actual boundary, in local coordinates, of the area of the button’s face in which content can be drawn.
function result	A result code; see “Result Codes” (page 238).

DISCUSSION

The `GetThemeButtonContentBounds` function obtains the rectangle where content can be drawn for a button under the current theme.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

GetThemeButtonRegion

Obtains the region occupied by a button.

```
pascal OSStatus GetThemeButtonRegion (
                    const Rect *inBounds,
                    ThemeButtonKind inKind,
                    const ThemeButtonDrawInfo *inNewInfo,
                    RgnHandle outRegion);
```

`inBounds`	A pointer to a structure of type `Rect`. Before calling `GetThemeButtonRegion`, set the rectangle to the boundary of the button without any adornments, in local coordinates.
`inKind`	A value of type `ThemeButtonKind`. Pass a constant specifying the type of button being examined. See "Theme Button Kind Constants" (page 187) for descriptions of possible values.
`inNewInfo`	A pointer to a structure of type `ThemeButtonDrawInfo` (page 165). Before calling `GetThemeButtonRegion`, set the structure to contain the state, value, and adornment for the button.
`outRegion`	A value of type `RgnHandle`. On return, the region contains the actual dimensions and position of the button and any adornments, in local coordinates.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

Appearance adornments can extend beyond the basic bounding rectangle of a button and may be of variable shape. Your application may call the `GetThemeButtonRegion` function to obtain the exact area covered by pixels belonging to a specific button under the current theme.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

GetThemeCheckBoxStyle

Obtains the system preference for the type of mark to use in a checkbox.

```
pascal OSStatus GetThemeCheckBoxStyle (
                        ThemeCheckBoxStyle *outStyle);
```

`outStyle`	A pointer to a value of type `ThemeCheckBoxStyle`. On return, the value specifies the type of mark being used. See "Theme Checkbox Style Constants" (page 190) for descriptions of possible values.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

Because international systems may specify the use of one type of mark to use in checkboxes over another, your application should call `GetThemeCheckBoxStyle` to obtain the correct type of mark to use on the current system.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

GetThemeScrollBarArrowStyle

Obtains the system preference for the type of scroll bar arrows to be used.

```
pascal OSStatus GetThemeScrollBarArrowStyle (
                        ThemeScrollBarArrowStyle *outStyle);
```

`outStyle`	A pointer to a value of type `ThemeScrollBarArrowStyle`. On return, the value specifies the type of scroll bar arrows being used. See "Theme Scroll Bar Arrow Style Constants" (page 205) for descriptions of possible values.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

Because the user can specify varying types of scroll bar arrows on a theme-specific basis, your application should call `GetThemeScrollBarArrowStyle` to obtain the preferred style under the current theme.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

GetThemeScrollBarThumbStyle

Obtains the system preference for the type of scroll box to be used.

```
pascal OSStatus GetThemeScrollBarThumbStyle (
                        ThemeScrollBarThumbStyle *outStyle);
```

`outStyle`	A pointer to a value of type `ThemeScrollBarThumbStyle`. On return, the value specifies the type of scroll box being used. See "Theme Scroll Box Style Constants" (page 206) for descriptions of possible values.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

Because the user can specify either proportional or fixed-size scroll boxes (also known as "scroll indicators" or "thumbs") on a theme-specific basis, your application should call `GetThemeScrollBarThumbStyle` to obtain the preferred style under the current theme.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

GetThemeScrollBarTrackRect

Obtains the area containing the track portion of a scroll bar.

```
pascal OSStatus GetThemeScrollBarTrackRect (
                    const Rect *bounds,
                    ThemeTrackEnableState enableState,
                    ThemeTrackPressState pressState,
                    Boolean isHoriz,
                    Rect *trackBounds);
```

`bounds`	A pointer to a structure of type `Rect`. Before calling `GetThemeScrollBarTrackRect`, set the rectangle to the boundary of the scroll bar, in local coordinates. Typically, the rectangle you specify is the entire base control rectangle—that is, the value contained in the `contrlRect` field of the track's `ControlRecord` structure.
`enableState`	A value of type `ThemeTrackEnableState`. Pass a constant specifying the current state of the scroll bar; see "Theme Track Enable State Constants" (page 231) for descriptions of possible values.
`pressState`	A value of type `ThemeTrackPressState`. Pass a constant specifying what is pressed in an active scroll bar or 0 if nothing is pressed; the press state is ignored if the scroll bar is not active. See "Theme Track Press State Constants" (page 233) for descriptions of possible values.
`isHoriz`	A value of type `Boolean`. Pass `true` if the scroll bar is horizontal; pass `false` if it is vertical.
`trackBounds`	A pointer to a structure of type `Rect`. On return, the structure contains the rectangle that bounds the track portion of the scroll bar. Note that the rectangle produced does not include in its bounds any tick marks that a track (such as a slider) might have; tick marks are drawn outside the track rectangle. Similarly, for a scroll bar, the rectangle produced does not contain the scroll bar arrows, just the track itself.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

Your application may call the `GetThemeScrollBarTrackRect` function to obtain the actual rectangle containing the track portion of a scroll bar under the current theme.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

GetThemeTabRegion

Obtains the region occupied by a tab.

```
pascal OSStatus GetThemeTabRegion (
                    const Rect *inRect,
                    ThemeTabStyle inStyle,
                    ThemeTabDirection inDirection,
                    RgnHandle ioRgn);
```

`inRect`	A pointer to a structure of type `Rect`. Before calling `GetThemeTabRegion`, set the rectangle to the boundary of the tab, in local coordinates.
`inStyle`	A value of type `ThemeTabStyle`. Pass a constant specifying the relative position (front or non-front) and state of the tab to be examined. See "Theme Tab Style Constants" (page 224) for descriptions of possible values.
`inDirection`	A value of type `ThemeTabDirection`. Pass a constant specifying the direction in which the tab is oriented. See "Theme Tab Direction Constants" (page 223) for descriptions of possible values.
`ioRgn`	A value of type `RgnHandle`. On return, the region contains the actual dimensions and position of the tab, in local coordinates.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

Because a tab can have a non-rectangular shape, your application should call `GetThemeTabRegion` to get the actual region containing the tab under the current theme, in order to perform accurate hit testing.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

GetThemeTrackBounds

Obtains the bounding rectangle of a track.

```
pascal OSStatus GetThemeTrackBounds (
                         const ThemeTrackDrawInfo *drawInfo,
                         Rect *bounds);
```

`drawInfo`	A pointer to a structure of type `ThemeTrackDrawInfo` (page 166). Before calling `GetThemeTrackBounds`, set the structure to describe the current visual characteristics of the track. Typically, the rectangle you specify in `ThemeTrackDrawInfo.bounds` is the proposed bounding rectangle for the track. `GetThemeTrackBounds` examines this rectangle to determine the actual bounds that the track would occupy. Depending on the track type, the actual bounding rectangle for a track might contain an absolute or fixed value (as for the height of a progress bar, which is always 14 pixels). Or, the track bounds might scale (as for a scroll bar) to fit the proposed bounds.
`bounds`	A pointer to a structure of type `Rect`. On return, the rectangle contains the actual boundary of the track, in local coordinates. Note that the rectangle produced does not include in its bounds any tick marks that a track (such as a slider) might have; tick marks are drawn outside the track rectangle. Similarly, for a scroll bar, the rectangle produced does not contain the scroll bar arrows, just the track itself.
function result	A result code; see “Result Codes” (page 238).

DISCUSSION

Your application may call the `GetThemeTrackBounds` function to obtain the actual rectangle containing a track under the current theme.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

GetThemeTrackDragRect

Obtains the area in which the user may drag a track's indicator.

```
pascal OSStatus GetThemeTrackDragRect (
                     const ThemeTrackDrawInfo *drawInfo,
                     Rect *dragRect);
```

`drawInfo`	A pointer to a structure of type `ThemeTrackDrawInfo` (page 166). Before calling `GetThemeTrackDragRect`, set the structure to contain the current visual characteristics of the track.
`dragRect`	A pointer to a structure of type `Rect`. On return, the rectangle contains the actual boundary of the indicator's drag rectangle, in local coordinates.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

Because of varying indicator geometries and theme designs, the draggable area for an indicator is not typically exactly the same as the track rectangle. Your application should call `GetThemeTrackDragRect` to obtain the actual area within a track where an indicator can be dragged under the current theme.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

GetThemeTrackLiveValue

Obtains the current value of a track's indicator, given its relative position.

```
pascal OSStatus GetThemeTrackLiveValue (
                    const ThemeTrackDrawInfo *drawInfo,
                    SInt32 relativePosition,
                    SInt32 *value);
```

`drawInfo` A pointer to a structure of type `ThemeTrackDrawInfo` (page 166). Before calling `GetThemeTrackLiveValue`, set the structure to contain the current visual characteristics of the track.

`relativePosition`
A signed 32-bit value. Pass the distance, in pixels, between the minimum end of the track and the near side of the indicator. You may obtain this value by calling either of the functions `GetThemeTrackThumbPositionFromOffset` (page 123) or `GetThemeTrackThumbPositionFromRegion` (page 124).

`value` A pointer to a signed 32-bit value. On return, this value contains the new value of the indicator.

function result A result code; see "Result Codes" (page 238).

DISCUSSION

Your application can use the `GetThemeTrackLiveValue` function to respond to the `posCntl` and `kControlMsgCalcValueFromPos` control definition message.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

GetThemeTrackThumbPositionFromOffset

Obtains the relative position of a track's indicator, given an offset from its prior position.

```
pascal OSStatus GetThemeTrackThumbPositionFromOffset (
                    const ThemeTrackDrawInfo *drawInfo,
                    Point thumbOffset,
                    SInt32 *relativePosition);
```

`drawInfo`	A pointer to a structure of type `ThemeTrackDrawInfo` (page 166). Before calling `GetThemeTrackThumbPositionFromOffset`, set the structure to contain the current visual characteristics of the track.
`thumbOffset`	A structure of type `Point`. Pass the point (in coordinates local to the control's window) that specifies the vertical and horizontal offset, in pixels, by which the indicator has moved from its current position. Typically, this is the offset between the locations where the cursor was when the user pressed and released the mouse button while dragging the indicator.
`relativePosition`	A pointer to a signed 32-bit value. On return, this value contains the new distance, in pixels, between the minimum end of the track and the near side of the indicator.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

Your application can use the `GetThemeTrackThumbPositionFromOffset` function to respond to the `posCntl` control definition message.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

GetThemeTrackThumbPositionFromRegion

Obtains the relative position of a track's indicator, given its current position.

```
pascal OSStatus GetThemTrackThumbPositionFromRegion (
                        const ThemeTrackDrawInfo *drawInfo,
                        RgnHandle thumbRgn,
                        SInt32 *relativePosition);
```

`drawInfo` A pointer to a structure of type `ThemeTrackDrawInfo` (page 166). Before calling `GetThemeTrackThumbPositionFromRegion`, set the structure to contain the current visual characteristics of the track.

`thumbRgn` A value of type `RgnHandle`. Before calling `GetThemeTrackThumbPositionFromRegion` set the region to contain the actual dimensions and position of the indicator, in local coordinates.

`relativePosition`
A pointer to a signed 32-bit value. On return, this value contains the new distance, in pixels, between the minimum end of the track and the near side of the indicator.

function result A result code; see "Result Codes" (page 238).

DISCUSSION

Your application can use the `GetThemeTrackThumbPositionFromRegion` function to respond to the `kControlMsgCalcValueFromPos` control definition message.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

GetThemeTrackThumbRgn

Obtains the region containing a track's indicator.

```
pascal OSStatus GetThemeTrackThumbRgn (
                        const ThemeTrackDrawInfo *drawInfo,
                        RgnHandle thumbRgn);
```

`drawInfo`	A pointer to a structure of type `ThemeTrackDrawInfo` (page 166). Before calling `GetThemeTrackThumbRgn`, set the structure to contain the current visual characteristics of the track.
`thumbRgn`	A value of type `RgnHandle`. On return, the region contains the actual dimensions and position of the indicator, in local coordinates.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

Your application can use the `GetThemeTrackThumbRgn` function to obtain the indicator region for tracks that have indicators, such as sliders and scroll bars.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

HitTestThemeScrollBarArrows

Returns whether the user clicked upon the specified scroll bar's arrows.

```
pascal Boolean HitTestThemeScrollBarArrows (
                    const Rect *scrollBarBounds,
                    ThemeTrackEnableState enableState,
                    ThemeTrackPressState pressState,
                    Boolean isHoriz,
                    Point ptHit,
                    Rect *trackBounds,
                    ControlPartCode *partcode);
```

`scrollBarBounds`

A pointer to a structure of type `Rect`. Before calling `HitTestThemeScrollBarArrows`, set the rectangle to the boundary of the scroll bar, in local coordinates. Typically, the rectangle you specify is the entire base control rectangle—that is, the value contained in the `contrlRect` field of the scroll bar's `ControlRecord` structure.

`enableState`	A value of type `ThemeTrackEnableState`. Pass a constant specifying the current state of the scroll bar; see "Theme Track Enable State Constants" (page 231) for descriptions of possible values.
`pressState`	A value of type `ThemeTrackPressState`. Pass a constant specifying what is pressed in an active scroll bar or 0 if nothing is pressed; the press state is ignored if the scroll bar is not active. See "Theme Track Press State Constants" (page 233) for descriptions of possible values.
`isHoriz`	A value of type `Boolean`. Pass `true` if the scroll bar is horizontal; pass `false` if it is vertical.
`ptHit`	A structure of type `Point`. Pass the point, specified in local coordinates, where the mouse-down event occurred. Your application may retrieve this value from the `where` field of the `event` structure.
`trackBounds`	A pointer to a structure of type `Rect`. On return, the rectangle contains the bounds of the track portion of the scroll bar; this rectangle excludes the area containing the scroll bar arrows. Pass `NULL` if you do not wish to obtain this information.
`partcode`	A pointer to a value of type `ControlPartCode`. On return, this value specifies the arrow in which the mouse-down event occurred.
function result	A value of type `Boolean`. If `true`, the mouse-down event occurred inside the scroll bar arrows; otherwise, `false`.

DISCUSSION

Your application may use the `HitTestThemeScrollBarArrows` function to test whether a given mouse-down event occurred on a scroll bar's arrows. If not, you may then use the rectangle produced in the `trackBounds` parameter of `HitTestThemeScrollBarArrows` as the bounds of the track for the function `HitTestThemeTrack` (page 127), in order to determine whether the mouse-down event occurred in the track part of the scroll bar.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

HitTestThemeTrack

Returns whether the user clicked upon the specified track.

```
pascal Boolean HitTestThemeTrack (
                        const ThemeTrackDrawInfo *drawInfo,
                        Point mousePoint,
                        ControlPartCode *partHit);
```

`drawInfo`	A pointer to a structure of type `ThemeTrackDrawInfo` (page 166). Before calling `HitTestThemeTrack`, set the structure to contain the current visual characteristics of the track.
`mousePoint`	A structure of type `Point`. Pass the point, specified in local coordinates, where the mouse-down event occurred. Your application may retrieve this value from the `where` field of the `event` structure.
`partHit`	A pointer to a value of type `ControlPartCode`. On return, this value specifies the part of the track in which the mouse-down event occurred.
function result	A value of type `Boolean`. If `true`, the mouse-down event occurred inside the track; otherwise, `false`.

DISCUSSION

The `HitTestThemeTrack` function checks to see whether a given track contains the specified point at which a mouse-down event occurred.

For a scroll bar–type track, your application should also check to see whether the mouse-down event occurred in the scroll bar's arrows, which are not considered part of the track and are not tested by this function. To do this, your application should first use the function `HitTestThemeScrollBarArrows` (page 125) to test whether a given mouse-down event occurred on a scroll bar's arrows. If not, you may then use the rectangle produced in the `rTrack` parameter of `HitTestThemeScrollBarArrows` as the bounds of the track for `HitTestThemeTrack`, in order to determine whether the mouse-down event occurred in the track part of the scroll bar.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

Drawing Theme-Compliant Windows

The Appearance Manager provides the following functions for applying theme-compliant colors and patterns to windows:

- `SetThemeWindowBackground` (page 143) associates a theme-compliant color or pattern with the background of a window.
- `SetThemeTextColorForWindow` (page 142) sets a window's foreground color to a theme-compliant color.

The Appearance Manager provides the following functions for drawing theme-compliant windows:

- `DrawThemeModelessDialogFrame` (page 129) draws a beveled outline inside the content area of a modeless dialog box.
- `DrawThemeScrollBarDelimiters` (page 130) outlines a window's scroll bars.
- `DrawThemeStandaloneGrowBox` (page 131) draws a size box.
- `DrawThemeStandaloneNoGrowBox` (page 132) draws a fill image for use in the corner space between scroll bars.
- `DrawThemeTitleBarWidget` (page 133) draws a close box, zoom box, or collapse box.
- `DrawThemeWindowFrame` (page 135) draws a window frame.
- `DrawThemeWindowHeader` (page 136) draws a window header.
- `DrawThemeWindowListViewHeader` (page 137) draws a window list view header.

The Appearance Manager provides the following functions for obtaining window region information:

- `GetThemeStandaloneGrowBoxBounds` (page 138) obtains the bounds of a size box.
- `GetThemeWindowRegion` (page 139) obtains the specified window region.
- `GetThemeWindowRegionHit` (page 140) obtains the part of the window that the user clicked upon.

DrawThemeModelessDialogFrame

Draws a beveled outline inside the content area of a modeless dialog box.

```
pascal OSStatus DrawThemeModelessDialogFrame (
                    const Rect *inRect,
                    ThemeDrawState inState);
```

`inRect`	A pointer to a structure of type `Rect`. Before calling `DrawThemeModelessDialogFrame`, set the rectangle to the boundary of the window's content area (that is, its port rectangle), inset by 1 pixel on each side, in local coordinates.
`inState`	A value of type `ThemeDrawState`. Pass a constant specifying the state in which to draw the modeless dialog box frame; see "Theme Draw State Constants" (page 199) for descriptions of possible values. The frame can be drawn as active or inactive; passing `kThemeStatePressed` produces an error.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

The `DrawThemeModelessDialogFrame` function draws a beveled frame, no more than 2 pixels wide, that bounds the window's content area. You can use this function to make a custom modeless dialog box theme-compliant; the Dialog Manager automatically draws the interior frame for standard dialog boxes.

SPECIAL CONSIDERATIONS

If you use `DrawThemeModelessDialogFrame` to draw a frame for a modeless dialog box, your application must explicitly invalidate and redraw the frame area if the dialog box is resized.

VERSION NOTES

Available with Appearance Manager 1.0.1 and later.

DrawThemeScrollBarDelimiters

Outlines a window's scroll bars.

```
pascal OSStatus DrawThemeScrollBarDelimiters (
                    ThemeWindowType flavor,
                    const Rect *inContRect,
                    ThemeDrawState state,
                    ThemeWindowAttributes attributes);
```

`flavor`	A value of type `ThemeWindowType`. Pass a constant specifying the type of window for which to draw scroll bar delimiters. See "Theme Window Type Constants" (page 236) for descriptions of possible values.
`inContRect`	A pointer to a structure of type `Rect`. Before calling `DrawThemeScrollBarDelimiters`, set the rectangle to the boundary of the content rectangle of the window, in local coordinates.
`state`	A value of type `ThemeDrawState`. Pass a constant—either `kThemeStateActive` or `kThemeStateInactive`—appropriate to the current state of the window. The scroll bar delimiters can be drawn as active or inactive; passing `kThemeStatePressed` produces an error. See "Theme Draw State Constants" (page 199) for descriptions of these values.
`attributes`	A value of type `ThemeWindowAttributes`. Pass one or more constants corresponding to the window's current visual attributes. See "Theme Window Attribute Constants" (page 235) for descriptions of possible values. Pass 0 if the window has none of the enumerated attributes.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

The `DrawThemeScrollBarDelimiters` function draws theme-compliant outlines for both the horizontal and vertical scroll bars in a given window. The scroll bars are each assumed to cover the full length of their respective sides of the window's content region; if the scroll bars for which you wish delimiters to be drawn are not full length, or if only one scroll bar exists for a given window, `DrawThemeScrollBarDelimiters` should not be used.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

DrawThemeStandaloneGrowBox

Draws a size box.

```
pascal OSStatus DrawThemeStandaloneGrowBox (
                    Point origin,
                    ThemeGrowDirection growDirection,
                    Boolean isSmall,
                    ThemeDrawState state);
```

`origin`	A structure of type `Point`. Pass the origin point of the size box rectangle. For example, the origin point of the size box for an object that can grow downward and to the right is the size box's upper-left corner. Typically, you use the coordinates of the corner of whatever object owns the size box for the `origin` value. For example, if you are drawing a scrolling list that can grow downward and to the right, the `origin` value would be the coordinates of the bottom-right corner of the list.
`growDirection`	A value of type `ThemeGrowDirection`. Pass a constant specifying the direction(s) in which the resizeable object can grow. See "Theme Size Box Direction Constants" (page 206) for descriptions of possible values. The Appearance Manager uses the `growDirection` parameter to establish which corner of the size box is the origin.
`isSmall`	A value of type `Boolean`. Pass a value of `true` to specify a small size box (typically for use with small scroll bars) or `false` to specify a standard size box.
`state`	A value of type `ThemeDrawState`. Pass a constant—either `kThemeStateActive` or `kThemeStateInactive`—appropriate to the current state of the size box; the size box cannot be drawn as pressed. See "Theme Draw State Constants" (page 199) for descriptions of these values.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

The `DrawThemeStandaloneGrowBox` function draws a theme-compliant size box that is suitable for use inside the content area of a window. The image is designed to fit between scroll bars and does not have to be abutted with the window frame.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

SEE ALSO

The function `DrawThemeStandaloneNoGrowBox` (page 132).

DrawThemeStandaloneNoGrowBox

Draws a fill image for use in the corner space between scroll bars.

```
pascal OSStatus DrawThemeStandaloneNoGrowBox (
                        Point origin,
                        ThemeGrowDirection growDirection,
                        Boolean isSmall,
                        ThemeDrawState state);
```

`origin`	A structure of type `Point`. Pass the origin point of the rectangle in which to draw the image. Typically, you use the coordinates of the corner of whatever object owns the image for the `origin` value. For example, if you are drawing the image in the bottom-right corner of a window between the scroll bars of a non-resizeable scrolling list, the `origin` value would be the coordinates of the bottom-right corner of the list.
`growDirection`	A value of type `ThemeGrowDirection`. See "Theme Size Box Direction Constants" (page 206) for descriptions of possible values. The Appearance Manager uses the `growDirection` parameter to establish which corner of the rectangle that contains the image is the origin.

`isSmall`	A value of type `Boolean`. Pass a value of `true` to specify a small image (for use with small scroll bars) or `false` to specify a large image (for use with standard scroll bars).
`state`	A value of type `ThemeDrawState`. Pass a constant—either `kThemeStateActive` or `kThemeStateInactive`—appropriate to the current state of the window containing the fill image; the image cannot be drawn as pressed. See "Theme Draw State Constants" (page 199) for descriptions of these values.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

The `DrawThemeStandaloneNoGrowBox` function draws a theme-compliant image for use as filler in the corner space between scroll bars that

- abut the frame of a window that is not resizeable and which therefore lacks a size box to fill the intervening space
- do not abut the window frame

VERSION NOTES

Available with Appearance Manager 1.1 and later.

SEE ALSO

The function `DrawThemeStandaloneGrowBox` (page 131).

DrawThemeTitleBarWidget

Draws a close box, zoom box, or collapse box.

```
pascal OSStatus DrawThemeTitleBarWidget (
                    ThemeWindowType flavor,
                    const Rect *contRect,
                    ThemeDrawState state,
                    const ThemeWindowMetrics *metrics,
                    ThemeWindowAttributes attributes,
                    ThemeTitleBarWidget widget);
```

`flavor`	A value of type `ThemeWindowType`. Pass a constant specifying the type of window for which to draw a title bar item. See "Theme Window Type Constants" (page 236) for descriptions of possible values.
`contRect`	A pointer to a structure of type `Rect`. Before calling `DrawThemeTitleBarWidget`, specify the rectangle for which you wish to draw a title bar item, in coordinates local to the current port. This rectangle is typically the content rectangle of a window.
`state`	A value of type `ThemeDrawState`. Pass a constant—`kThemeStateActive`, `kThemeStateInactive`, or `kThemeStatePressed`—appropriate to the current state of the title bar item. See "Theme Draw State Constants" (page 199) for descriptions of these values.
`metrics`	A pointer to a structure of type `ThemeWindowMetrics` (page 169). Before calling `DrawThemeTitleBarWidget`, set the structure to contain information describing the window for which you wish to draw a title bar item.
`attributes`	A value of type `ThemeWindowAttributes`. Pass one or more constants corresponding to the window's current visual attributes. See "Theme Window Attribute Constants" (page 235) for descriptions of possible values. Pass 0 if the window has none of the enumerated attributes.
`widget`	A value of type `ThemeTitleBarWidget`. Pass a constant—`kThemeWidgetCloseBox`, `kThemeWidgetZoomBox`, or `kThemeWidgetCollapseBox`—appropriate to the type of title bar item you wish to draw. See "Theme Title Bar Item Constants" (page 230) for descriptions of these values.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

The `DrawThemeTitleBarWidget` function draws theme-compliant title bar items. Your application should not typically need to call this function; `DrawThemeTitleBarWidget` is typically of use only for applications that need to draw title bar items of simulated windows. Note that while the `DrawThemeWindowFrame` function automatically draws all title bar items, your application must call the `DrawThemeTitleBarWidget` function during tracking, to ensure that the title bar items' states are drawn correctly.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

DrawThemeWindowFrame

Draws a window frame.

```
pascal OSStatus DrawThemeWindowFrame (
                    ThemeWindowType flavor,
                    const Rect *contRect,
                    ThemeDrawState state,
                    const ThemeWindowMetrics *metrics,
                    ThemeWindowAttributes attributes,
                    WindowTitleDrawingUPP titleProc,
                    UInt32 titleData);
```

`flavor`	A value of type `ThemeWindowType`. Pass a constant specifying the type of window for which to draw a frame. See "Theme Window Type Constants" (page 236) for descriptions of possible values.
`contRect`	A pointer to a structure of type `Rect`. Before calling `DrawThemeWindowFrame`, specify the rectangle for which you wish to draw a window frame, in coordinates local to the current port. This rectangle is typically the content rectangle of a window.
`state`	A value of type `ThemeDrawState`. Pass a constant—either `kThemeStateActive` or `kThemeStateInactive`—appropriate to the current state of the window. See "Theme Draw State Constants" (page 199) for descriptions of these values.
`metrics`	A pointer to a structure of type `ThemeWindowMetrics` (page 169). Before calling `DrawThemeWindowFrame`, set the structure to describe the window for which to draw a frame.

`attributes`	A value of type `ThemeWindowAttributes`. Pass one or more constants corresponding to the window's current visual attributes. See "Theme Window Attribute Constants" (page 235) for descriptions of possible values. Pass 0 if the window has none of the enumerated attributes.
`titleProc`	A value of type `WindowTitleDrawingUPP` (page 174). If you pass the value `kThemeWindowHasTitleText` in the `attributes` parameter, you should pass a universal procedure pointer to an application-defined function such as that described in `MyWindowTitleDrawingProc` (page 162) in the `titleProc` parameter. `DrawThemeWindowFrame` calls that function to draw the window's title. Pass `NULL` if there is no title to be drawn.
`titleData`	An unsigned 32-bit integer. Provide any data to be passed in to the `userData` parameter of the callback function specified in the `titleProc` parameter.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

The `DrawThemeWindowFrame` function draws a window frame appropriate to the specified window type. You may use `DrawThemeWindowFrame` to make a custom window theme-compliant.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

DrawThemeWindowHeader

Draws a window header.

```
pascal OSStatus DrawThemeWindowHeader (
                        const Rect *inRect,
                        ThemeDrawState inState);
```

`inRect`	A pointer to a structure of type `Rect`. Before calling `DrawThemeWindowHeader`, specify the rectangle containing the window header, in local coordinates.

`inState`	A value of type `ThemeDrawState`. Pass a constant specifying the state in which to draw the window header; see "Theme Draw State Constants" (page 199). The header can be drawn as active or inactive; passing `kThemeStatePressed` produces an error.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

The `DrawThemeWindowHeader` function draws a theme-compliant window header, such as that used by the Finder.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

DrawThemeWindowListViewHeader

Draws a window list view header.

```
pascal OSStatus DrawThemeWindowListViewHeader (
                    const Rect *inRect,
                    ThemeDrawState inState);
```

`inRect`	A pointer to a structure of type `Rect`. Before calling `DrawThemeWindowListViewHeader`, specify the rectangle in which to draw the window list view header, in local coordinates.
`inState`	A value of type `ThemeDrawState`. Pass a constant specifying the state in which to draw the window list view header; see "Theme Draw State Constants" (page 199). The header can be drawn as active or inactive; passing `kThemeStatePressed` produces an error.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

The `DrawThemeWindowListViewHeader` function draws a theme-compliant window list view header, such as that used by the Finder. 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.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

GetThemeStandaloneGrowBoxBounds

Obtains the bounds of a size box.

```
pascal OSStatus GetThemeStandaloneGrowBoxBounds (
                    Point origin,
                    ThemeGrowDirection growDirection,
                    Boolean isSmall,
                    Rect *bounds);
```

`origin`	A structure of type `Point`. Pass the origin point of the size box rectangle. For example, the origin point of the size box for an object that can grow downward and to the right is the size box's upper-left corner. Typically, you use the coordinates of the corner of whatever object owns the size box for the `origin` value; for instance, if you are drawing a scrolling list that can grow downward and to the right, the `origin` value would be the coordinates of the bottom-right corner of the list.
`growDirection`	A value of type `ThemeGrowDirection`. For a size box, pass a constant specifying the direction(s) in which the window can grow. See "Theme Size Box Direction Constants" (page 206) for descriptions of possible values. The Appearance Manager uses the `growDirection` parameter to establish which corner of the size box is the origin.
`isSmall`	A value of type `Boolean`. Pass a value of `true` to specify a small size box or fill image. Pass `false` to specify a large size box or fill image.
`bounds`	A pointer to a structure of type `Rect`. On return, the rectangle contains the boundary of the size box or fill image.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

The `GetThemeStandaloneGrowBoxBounds` function obtains the bounds of a size box under the current theme. Note that you can also use `GetThemeStandaloneGrowBoxBounds` to obtain the bounds of the fill image drawn by the function `DrawThemeStandaloneNoGrowBox` (page 132).

VERSION NOTES

Available with Appearance Manager 1.1 and later.

GetThemeWindowRegion

Obtains the specified window region.

```
pascal OSStatus GetThemeWindowRegion (
                    ThemeWindowType flavor,
                    const Rect *contRect,
                    ThemeDrawState state,
                    const ThemeWindowMetrics *metrics,
                    ThemeWindowAttributes attributes,
                    WindowRegionCode winRegion,
                    RgnHandle rgn);
```

`flavor`	A value of type `ThemeWindowType`. Pass a constant specifying the type of window to be examined. See "Theme Window Type Constants" (page 236) for descriptions of possible values.
`contRect`	A pointer to a structure of type `Rect`. Before calling `GetThemeWindowRegion`, set the rectangle to the content area of the window, specified in coordinates local to the current port.
`state`	A value of type `ThemeDrawState`. Pass a constant—either `kThemeStateActive` or `kThemeStateInactive`—appropriate to the current state of the window. See "Theme Draw State Constants" (page 199) for descriptions of these values.
`metrics`	A pointer to a structure of type `ThemeWindowMetrics` (page 169). Before calling `GetThemeWindowRegion`, set the structure to contain information describing the window.

`attributes`	A value of type `ThemeWindowAttributes`. Pass one or more constants corresponding to the window's current visual attributes. See "Theme Window Attribute Constants" (page 235) for descriptions of possible values. Pass 0 if the window has none of the enumerated attributes.
`winRegion`	A value of type `WindowRegionCode`. Pass a constant specifying the region of the window whose dimensions you wish to obtain.
`rgn`	A value of type `RgnHandle`. Pass a handle to a valid region. On return, the region represents the actual region requested.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

The `GetThemeWindowRegion` function obtains the dimensions of the specified window region under the current theme.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

GetThemeWindowRegionHit

Obtains the part of the window that the user clicked upon.

```
pascal Boolean GetThemeWindowRegionHit (
                     ThemeWindowType flavor,
                     const Rect *inContRect,
                     ThemeDrawState state,
                     const ThemeWindowMetrics *metrics,
                     ThemeWindowAttributes inAttributes,
                     Point inPoint,
                     WindowRegionCode *outRegionHit);
```

`flavor`	A value of type `ThemeWindowType`. Pass a constant specifying the type of window to be examined. See "Theme Window Type Constants" (page 236) for descriptions of possible values.

`inContRect`	A pointer to a structure of type `Rect`. Before calling `GetThemeWindowRegionHit`, set rectangle to the content area of the window, specified in coordinates local to the current port.
`state`	A value of type `ThemeDrawState`. Pass a constant—either `kThemeStateActive` or `kThemeStateInactive`—appropriate to the current state of the window. See "Theme Draw State Constants" (page 199) for descriptions of these values.
`metrics`	A pointer to a structure of type `ThemeWindowMetrics` (page 169). Before calling `GetThemeWindowRegionHit`, set the structure to contain information describing the window.
`inAttributes`	A value of type `ThemeWindowAttributes`. Pass one or more constants corresponding to the window's current visual attributes. See "Theme Window Attribute Constants" (page 235) for descriptions of possible values. Pass 0 if the window has none of the enumerated attributes.
`inPoint`	A structure of type `Point`. Pass the point, specified in specified in coordinates local to the current port, where the mouse-down event occurred. Your application may retrieve this value from the `where` field of the `event` structure.
`outRegionHit`	A pointer to a value of type `WindowRegionCode`. On return, the value is set to the region code of the window part in which the point passed in the `inPoint` parameter is located.
function result	A value of type `Boolean`. If `true`, the mouse-down event occurred inside the window; otherwise, `false`.

DISCUSSION

Your window definition function should call the `GetThemeWindowRegionHit` function to determine where a specified mouse-down event occurred.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

SetThemeTextColorForWindow

Sets a window's foreground color to a theme-compliant color.

```
pascal OSStatus SetThemeTextColorForWindow (
                        WindowPtr window,
                        Boolean isActive,
                        SInt16 depth,
                        Boolean isColorDev);
```

`window`	A value of type `WindowPtr`. Pass a pointer to the window for which to set the text color.
`isActive`	A value of type `Boolean`. Pass `true` if the window is currently active; otherwise, `false`.
`depth`	A signed 16-bit integer. Pass the bit depth (in bits per pixel) of the current graphics port.
`isColorDev`	A value of type `Boolean`. Set to `true` to indicate that you are drawing on a color device. Set to `false` for a monochrome device.
function result	A result code; see "Result Codes" (page 238). If the specified window does not currently have a theme brush associated with its background or if the associated theme brush is not translatable into a text color, `SetThemeTextColorForWindow` returns an error.

DISCUSSION

The `SetThemeTextColorForWindow` function determines whether the specified window has a theme brush associated with its background and, if so, applies the correct foreground color given the theme brush, the window's activity state, and the current drawing environment. This foreground color is applied to any text drawn after the call to `SetThemeTextColorForWindow`. Note that `SetThemeTextColorForWindow` only applies a custom foreground color when the window has a theme brush associated with it or if the window is a Dialog Manager dialog box. If your window is not managed by the Dialog Manager, you must use the function `SetThemeWindowBackground` (page 143) to associate a theme brush with the window before calling `SetThemeTextColorForWindow`.

For example, you can call `SetThemeWindowBackground` to associate the active alert background theme brush with a window:

```
SetThemeWindowBackground (myWindow, kThemeBrushAlertBackgroundActive,
true)
// ...
SetThemeTextColorForWindow (myWindow, true, 16, true)
```

Then, when you call `SetThemeTextColorForWindow`, the result is that the window’s foreground color is automatically set to the color appropriate for an active alert in the specified drawing environment. In this case, the foreground color that the Appearance Manager would use is that corresponding to the theme text color constant `kThemeTextColorAlertActive`.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

SetThemeWindowBackground

Associates a theme-compliant color or pattern with the background of a window.

```
pascal OSStatus SetThemeWindowBackground (
                    WindowPtr inWindow,
                    ThemeBrush inBrush,
                    Boolean inUpdate);
```

`inWindow`	A value of type `WindowPtr`. Pass a pointer to the window for which to set the background.
`inBrush`	A value of type `ThemeBrush`. Pass a constant representing the pattern or color to which to set the window background; see “Theme Brush Constants” (page 180) for descriptions of possible values.
`inUpdate`	A value of type `Boolean`. Pass `true` to invalidate the content region of the window and erase the window. If you pass `false`, the window background is set but no drawing occurs on screen.
function result	A result code. The result code `appearanceBadBrushIndexErr` indicates that the brush constant passed was not valid. For a list of other result codes, see “Result Codes” (page 238).

DISCUSSION

The `SetThemeWindowBackground` function sets the color or pattern to which the Window Manager erases the window background.

Because the constant in the `inBrush` parameter can represent a color or pattern, depending on the current theme, your application must save and restore the current drawing state of the graphics port around calls to `SetThemeWindowBackground`. Under Appearance Manager 1.1 and later, you can use the functions `GetThemeDrawingState` (page 80) and `SetThemeDrawingState` (page 84) to do this. Prior to Appearance Manager 1.1, you must save and restore the `pnPixPat` and `bkPixPat` fields of your graphics port when saving the text and background colors. Because patterns in the `bkPixPat` field override the background color of the window, call the QuickDraw function `BackPat` to set your background pattern to a normal white pattern. This ensures that you can use `RGBBackColor` to set your background color to white, call the QuickDraw function `EraseRect`, and get the expected results.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

Drawing Theme-Compliant Menus

The Appearance Manager provides the following functions for drawing theme-compliant menus:

- `DrawThemeMenuBackground` (page 145) draws a menu background.
- `GetThemeMenuBackgroundRegion` (page 150) obtains the background region for a menu.

The Appearance Manager provides the following functions for drawing theme-compliant menu titles:

- `DrawThemeMenuTitle` (page 149) draws a menu title.
- `GetThemeMenuTitleExtra` (page 154) obtains a measurement of the space to either side of a menu title.

The Appearance Manager provides the following functions for drawing theme-compliant menu items:

- `DrawThemeMenuItem` (page 146) draws a menu item.

- `DrawThemeMenuSeparator` (page 148) draws a menu item separator line.
- `GetThemeMenuItemExtra` (page 152) obtains a measurement of the space surrounding a menu item.
- `GetThemeMenuSeparatorHeight` (page 153) obtains the height of a menu separator line.

The Appearance Manager provides the following functions for drawing theme-compliant menu bars:

- `DrawThemeMenuBarBackground` (page 146) draws a menu bar background.
- `GetThemeMenuBarHeight` (page 151) obtains the optimal height of a menu bar.

DrawThemeMenuBackground

Draws a menu background.

```
pascal OSStatus DrawThemeMenuBackground (
                        const Rect *inMenuRect,
                        ThemeMenuType inMenuType);
```

`inMenuRect`	A pointer to a structure of type `Rect`. Before calling `DrawThemeMenuBackground`, set the rectangle to contain the entire menu, in global coordinates.
`inMenuType`	A value of type `ThemeMenuType`. Pass a constant specifying the type of menu for which to draw a background; see "Theme Menu Type Constants" (page 203) for descriptions of possible values.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

The `DrawThemeMenuBackground` function draws a theme-compliant menu background in the specified rectangle.

VERSION NOTES

Available with Appearance Manager 1.0.1 and later.

DrawThemeMenuBarBackground

Draws a menu bar background.

```
pascal OSStatus DrawThemeMenuBarBackground (
                    const Rect *inBounds,
                    ThemeMenuBarState inState,
                    UInt32 inAttributes);
```

`inBounds`	A pointer to a structure of type `Rect`. Before calling `DrawThemeMenuBarBackground`, set the rectangle to specify the menu bar's initial size and location, in global coordinates.
`inState`	A value of type `ThemeMenuBarState`. Pass a constant specifying the state (active or selected) in which to draw the menu bar; see "Theme Menu Bar State Constants" (page 200).
`inAttributes`	Reserved. Pass 0.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

The `DrawThemeMenuBarBackground` function draws a theme-compliant menu bar background in the specified rectangle.

VERSION NOTES

Available with Appearance Manager 1.0.1 and later.

DrawThemeMenuItem

Draws a menu item.

```
pascal OSStatus DrawThemeMenuItem (
                    const Rect *inMenuRect,
                    const Rect *inItemRect,
                    SInt16 inVirtualMenuTop,
                    SInt16 inVirtualMenuBottom,
                    ThemeMenuState inState,
                    ThemeMenuItemType inItemType,
                    MenuItemDrawingUPP inDrawProc,
                    UInt32 inUserData);
```

`inMenuRect` A pointer to a structure of type `Rect`. Before calling `DrawThemeMenuItem`, set the rectangle to contain the entire menu, in global coordinates. This is the actual menu rectangle as used in your menu definition function.

`inItemRect` A pointer to a structure of type `Rect`. Before calling `DrawThemeMenuItem`, set the rectangle to contain the menu item, in global coordinates. The menu item's background is drawn in the rectangle passed in the `inItemRect` parameter. You should calculate the size of the menu item's content and then call `GetThemeMenuItemExtra` (page 152) to get the amount of padding surrounding menu items in the current theme; the width and height of the menu item rectangle are determined by adding these values together.

`inVirtualMenuTop`
A signed 16-bit integer. Pass a value representing the actual top of the menu. Normally this value is the top coordinate of the rectangle supplied in the `inMenuRect` parameter. This value could be different, however, if a menu is scrolled or bigger than can be displayed in the menu rectangle. You typically pass the value of the global variable `TopMenuItem` into this parameter if you are writing a custom menu definition function.

`inVirtualMenuBottom`
A signed 16-bit integer. Pass a value representing the actual bottom of the menu. Typically this value is the bottom coordinate of the rectangle supplied in the `inMenuRect` parameter. This value could be different, however, if a menu is scrolled or bigger than can be displayed in the menu rectangle. You typically pass the value of the global variable `AtMenuBottom` into this parameter if you are writing a custom menu definition function.

`inState` A value of type `ThemeMenuState`. Pass a constant specifying the state (active, selected, or disabled) in which to draw the menu item; see "Theme Menu State Constants" (page 203).

`inItemType`	A value of type `ThemeMenuItemType`. If you pass `kThemeMenuItemScrollUpArrow` or `kThemeMenuItemScrollDownArrow`, then you should pass `NULL` for the `inDrawProc` parameter, since there is no content to be drawn. If you pass `kThemeMenuItemHierarchical`, the hierarchical arrow is drawn for you. See "Theme Menu Item Type Constants" (page 201) for descriptions of possible values.
`inDrawProc`	A value of type `MenuItemDrawingUPP` (page 175). Pass a universal procedure pointer to a menu item drawing function such as `MyMenuItemDrawingProc` (page 155). The value of the `inDrawProc` parameter can be a valid universal procedure pointer or `NULL`.
`inUserData`	An unsigned 32-bit integer. Provide any data to be passed in to the `inUserData` parameter of `MyMenuItemDrawingProc` (page 155).
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

The `DrawThemeMenuItem` function draws a theme-compliant menu item.

VERSION NOTES

Available with Appearance Manager 1.0.1 and later.

DrawThemeMenuSeparator

Draws a menu item separator line.

```
pascal OSStatus DrawThemeMenuSeparator (const Rect *inItemRect);
```

`inItemRect`	A pointer to a structure of type `Rect`. Before calling `DrawThemeMenuSeparator`, set the rectangle to contain the menu item separator to be drawn, in global coordinates. The rectangle should be the same height as the height returned by the function `GetThemeMenuSeparatorHeight` (page 153).
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

The `DrawThemeMenuSeparator` function draws a theme-compliant menu item separator line.

VERSION NOTES

Available with Appearance Manager 1.0.1 and later.

DrawThemeMenuTitle

Draws a menu title.

```
pascal OSStatus DrawThemeMenuTitle (
                    const Rect *inMenuBarRect,
                    const Rect *inTitleRect,
                    ThemeMenuState inState,
                    UInt32 inAttributes,
                    MenuTitleDrawingUPP inTitleProc,
                    UInt32 inTitleData);
```

`inMenuBarRect`	A pointer to a structure of type `Rect`. Before calling `DrawThemeMenuTitle`, set the rectangle to contain the entire menu bar in which the title is to be drawn, in global coordinates. The menu bar background is drawn in the rectangle passed in the `inMenuBarRect` parameter. Your application can call `GetThemeMenuBarHeight` (page 151) to get the height of the menu bar.
`inTitleRect`	A pointer to a structure of type `Rect`. Before calling `DrawThemeMenuTitle`, set the rectangle to contain the menu title, in global coordinates. The title background is drawn in the rectangle passed in the `inTitleRect` parameter. The width of this rectangle is determined by calculating the width of the menu title's content and then calling `GetThemeMenuTitleExtra` (page 154) to get the amount of padding between menu titles in the current theme; these two values are added together and added to the left edge of where the title should be drawn. The top and bottom coordinates of this rectangle should be the same as those of the `inMenuBarRect` parameter.

`inState`	A value of type `ThemeMenuState`. Pass a constant specifying the state (active, selected, or disabled) in which to draw the menu title; see "Theme Menu State Constants" (page 203).
`inAttributes`	Reserved. Pass 0.
`inTitleProc`	A value of type `MenuTitleDrawingUPP` (page 175). Pass a universal procedure pointer to a menu title drawing function such as `MyMenuTitleDrawingProc` (page 156), defining how to draw the contents of the menu title. The value of the `inTitleProc` parameter can be a valid universal procedure pointer or `NULL`.
`inTitleData`	An unsigned 32-bit integer. Provide any data to be passed in to the `inUserData` parameter of `MyMenuTitleDrawingProc` (page 156).
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

The `DrawThemeMenuTitle` function draws a theme-compliant menu title.

VERSION NOTES

Available with Appearance Manager 1.0.1 and later.

GetThemeMenuBackgroundRegion

Obtains the background region for a menu.

```
pascal OSStatus GetThemeMenuBackgroundRegion (
                    const Rect *inMenuRect,
                    ThemeMenuType menuType,
                    RgnHandle Region);
```

`inMenuRect`	A pointer to a structure of type `Rect`. Before calling `GetThemeMenuBackgroundRegion`, set the rectangle to contain the entire menu, in global coordinates.

`menuType`	A value of type `ThemeMenuType`. Pass a constant specifying the type of menu (pull-down, pop-up, or hierarchical) whose background you wish to obtain; see "Theme Menu Type Constants" (page 203) for descriptions of possible values.
`Region`	A value of type `RgnHandle`. Pass a region handle created by your application. On return, the region is set to that of the rectangle specified in the `inMenuRect` parameter, that is, the menu's background region.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

The `GetThemeMenuBackgroundRegion` function obtains the background region that a menu occupies under the current theme.

VERSION NOTES

Available with Appearance Manager 1.0.1 and later.

GetThemeMenuBarHeight

Obtains the height of a menu bar.

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

`outHeight`	A pointer to a signed 16-bit integer. On return, the integer value represents the height (in pixels) of the menu bar.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

The `GetThemeMenuBarHeight` function obtains the specified height of a menu bar in the current theme. This is in contrast to the Menu Manager function `GetMBarHeight`, which obtains the actual space that the menu bar is currently occupying on the screen. In most instances, the values produced by these two functions are the same. But, when the menu bar is hidden, `GetMBarHeight` produces a value of 0, and `GetThemeMenuBarHeight` still provides the "ideal" menu bar height.

SPECIAL CONSIDERATIONS

Because menu bar heights may vary among appearances by one or more pixels, you should check the current menu bar height after a theme switch. Specifically, your application should respond to the theme-switch Apple event, `kAEAppearanceChanged`, by checking the current menu bar height. See "Appearance Manager Apple Event Constants" (page 177) for more details on `kAEAppearanceChanged`.

It is important to check the menu bar height before positioning any windows. Failure to do so may result in the menu bar overlapping your application's windows.

VERSION NOTES

Available with Appearance Manager 1.0.1 and later.

GetThemeMenuItemExtra

Obtains a measurement of the space surrounding a menu item.

```
pascal OSStatus GetThemeMenuItemExtra (
                    ThemeMenuItemType inItemType,
                    SInt16 *outHeight,
                    SInt16 *outWidth);
```

`inItemType`	A value of type `ThemeMenuItemType`. Pass a constant identifying the type of menu item for which you are interested in getting a measurement. See "Theme Menu Item Type Constants" (page 201).
`outHeight`	A pointer to a signed 16-bit integer. On return, the integer value represents the total amount of padding between the content of the menu item and the top and bottom of its frame (in pixels). Your content's height plus the measurement provided by the `outHeight` parameter equals the total item height.
`outWidth`	A pointer to a signed 16-bit integer. On return, the integer value represents the total amount of padding between the content of the menu item and the left and right limits of the menu (in pixels). Your content's width plus the measurement provided by the `outWidth` parameter equals the total item width.

function result A result code; see "Result Codes" (page 238).

DISCUSSION

Your application should call the `GetThemeMenuItemExtra` function when you are writing your own menu definition function and wish to be theme-compliant. Once you have determined the height and width of the content of a menu item, call `GetThemeMenuItemExtra` to get a measurement in pixels of the space surrounding a menu item, including any necessary inter-item spacing, in the current theme. By combining the values for your menu item's content and the extra padding needed by the theme, you can derive the size of the rectangle needed to encompass both the content and the theme element together.

VERSION NOTES

Available with Appearance Manager 1.0.1 and later.

GetThemeMenuSeparatorHeight

Obtains the height of a menu separator line.

```
pascal OSStatus GetThemeMenuSeparatorHeight (SInt16 *outHeight);
```

`outHeight`	A pointer to a signed 16-bit integer. On return, the integer value represents the height (in pixels) of the menu separator line.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

The `GetThemeMenuSeparatorHeight` function obtains the height of a menu separator line under the current theme. Your application should call the `GetThemeMenuSeparatorHeight` function when you are writing your own menu definition function and wish to calculate a menu rectangle for a separator to match the current theme.

VERSION NOTES

Available with Appearance Manager 1.0.1 and later.

GetThemeMenuTitleExtra

Obtains a measurement of the space to either side of a menu title.

```
pascal OSStatus GetThemeMenuTitleExtra (
                    SInt16 *outWidth,
                    Boolean inIsSquished);
```

`outWidth`	A pointer to a signed 16-bit integer. On return, the integer value represents the horizontal distance (in pixels) between the menu title and the bounds of its containing rectangle.
`inIsSquished`	A value of type `Boolean`. If all the titles do not fit in the menu bar and you wish to condense the menu title's spacing to fit, pass `true`. If you pass `false`, the menu title is not condensed.
function result	A result code; see "Result Codes" (page 238).

DISCUSSION

Once you have determined the height and width of the content of a menu title, call `GetThemeMenuTitleExtra` to get the space surrounding the menu title in the current theme.

VERSION NOTES

Available with Appearance Manager 1.0.1 and later.

Application-Defined Functions

The Appearance Manager supports the following application-defined functions:

- `MyThemeIteratorProc` (page 160) performs a custom response to an iteration over themes installed on a system.
- `MyThemeButtonDrawProc` (page 157) draws a button label.
- `MyThemeEraseProc` (page 159) draws a background.
- `MyThemeTabTitleDrawProc` (page 161) draws a tab title.
- `MyWindowTitleDrawingProc` (page 162) draws a window title.

- `MyMenuItemDrawingProc` (page 155) draws a menu item.
- `MyMenuTitleDrawingProc` (page 156) draws a menu title.

MyMenuItemDrawingProc

Draws a menu item.

Here's how to declare a menu item drawing function, if you were to name the function `MyMenuItemDrawingProc`:

```
pascal void (MyMenuItemDrawingProc)
                    (const Rect *inBounds,
                    SInt16 inDepth,
                    Boolean inIsColorDevice,
                    SInt32 inUserData);
```

`inBounds`	A pointer to a structure of type `Rect`. You are passed a rectangle specifying the dimensions and position in which you should draw your menu item content. Your menu item drawing function is called clipped to the rectangle in which you are allowed to draw your content; do not draw outside this region.
`inDepth`	A signed 16-bit integer. You are passed the bit depth (in bits per pixel) of the current graphics port.
`inIsColorDevice`	A value of type `Boolean`. You are passed `true` to indicate that you are drawing on a color device; `inIsColorDevice` is `false` for a monochrome device.
`inUserData`	You are passed data specifying how to draw the menu item content from the `inUserData` parameter of `DrawThemeMenuItem` (page 146).

DISCUSSION

At the time your menu item drawing function is called, the foreground text color and mode is already set to draw in the correct state (enabled, selected, disabled) and correct color for the theme. You do not need to set the color unless you have special drawing needs. If you do have special drawing needs, you should supply the `inDepth` value and the value of the `inIsColorDevice` parameter to the function `IsThemeInColor` to determine whether or not you should draw the menu item content in color.

Note that the Appearance Manager calls your `MyMenuItemDrawingProc` function for every device that the `inBounds` rectangle intersects.

SPECIAL CONSIDERATIONS

The Appearance Manager draws the background of the menu item prior to calling your menu item drawing function, so you should not erase the item's background from this function.

VERSION NOTES

Available with Appearance Manager 1.0.1 and later.

MyMenuTitleDrawingProc

Draws a menu title.

Here's how to declare a menu title drawing function, if you were to name the function `MyMenuTitleDrawingProc`:

```
pascal void (MyMenuTitleDrawingProc)
                    (const Rect *inBounds,
                    SInt16 inDepth,
                    Boolean inIsColorDevice,
                    SInt32 inUserData);
```

`inBounds` A pointer to a structure of type `Rect`. You are passed a rectangle specifying the dimensions and position in which you should draw your menu title content. Your menu title drawing function is called clipped to the rectangle in which you are allowed to draw your content; do not draw outside this region.

`inDepth` A signed 16-bit integer. You are passed the bit depth (in bits per pixel) of the current graphics port.

`inIsColorDevice`
A value of type `Boolean`. You are passed `true` to indicate that you are drawing on a color device; `inIsColorDevice` is `false` for a monochrome device.

`inUserData` You are passed data specifying how to draw the menu title content from the `inTitleData` parameter of `DrawThemeMenuTitle` (page 149).

DISCUSSION

At the time your menu title drawing function is called, the foreground text color and mode is already set to draw in the correct state (enabled, selected, disabled) and correct color for the theme. You do not need to set the color unless you have special drawing needs. If you do have special drawing needs, you should supply the `inDepth` value and the value of the `inIsColorDevice` parameter to the function `IsThemeInColor` to determine whether or not you should draw the menu title content in color.

Note that the Appearance Manager calls your `MyMenuTitleDrawingProc` function for every device that the `inBounds` rectangle intersects.

SPECIAL CONSIDERATIONS

The Appearance Manager draws the background of the menu title prior to calling your menu title drawing function, so you should not erase the title's background from this function.

VERSION NOTES

Available with Appearance Manager 1.0.1 and later.

MyThemeButtonDrawProc

Draws a button label.

Here's how to declare a button label drawing function, if you were to name the function `MyThemeButtonDrawProc`:

```
pascal void MyThemeButtonDrawProc (
                    const Rect *bounds,
                    ThemeButtonKind kind,
                    ThemeButtonDrawInfo *info,
                    UInt32 userData,
                    SInt16 depth,
                    Boolean isColorDev);
```

`bounds`	A pointer to a structure of type `Rect`. The rectangle you are passed is set to the area in which you should draw your content. Your button label drawing function is called clipped to the rectangle in which you are allowed to draw your content; do not draw outside this region. Note that if a right-to-left adornment is specified in the `ThemeButtonDrawInfo` structure passed into the `info` parameter, you may need to accommodate this orientation when placing your content.
`kind`	A value of type `ThemeButtonKind`. You are passed a constant specifying the button type. See "Theme Button Kind Constants" (page 187) for descriptions of possible values.
`info`	A pointer to a structure of type `ThemeButtonDrawInfo` (page 165). The structure is set to contain the current state, value, and adornment for the button.
`userData`	An unsigned 32-bit value. You are passed data specifying how to draw the content, from the `inUserData` parameter of `DrawThemeButton` (page 94).
`depth`	A signed 16-bit value. You are passed the bit depth (in bits per pixel) of the current graphics port.
`isColorDev`	A value of type `Boolean`. If `true`, indicates that you are drawing on a color device; a value of `false` indicates a monochrome device.

DISCUSSION

At the time your button label drawing function is called, the foreground text color and mode is already set to draw in the correct state (active or inactive) and correct color for the theme. You do not need to set the color unless you have special drawing needs. If you do have special drawing needs, you should supply the `depth` value and the value of the `isColorDevice` parameter to the function `IsThemeInColor` to determine whether or not you should draw your content in color. Note that the Appearance Manager calls your `MyThemeButtonDrawProc` function for every device that the `bounds` rectangle intersects.

SPECIAL CONSIDERATIONS

The Appearance Manager draws the button background prior to calling your button label drawing function, so you should not erase the button background from your label drawing function.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

MyThemeEraseProc

Draws a background.

Here's how to declare a background drawing function, if you were to name the function `MyThemeEraseProc`:

```
pascal void MyThemeEraseProc (
                    const Rect *bounds,
                    UInt32 eraseData,
                    SInt16 depth,
                    Boolean isColorDev);
```

`bounds`	A pointer to a structure of type `Rect`. The rectangle you are passed is set to the area in which you should draw. Your drawing function is called clipped to the rectangle in which you are allowed to draw; do not draw outside this region.
`eraseData`	An unsigned 32-bit value. You are passed data specifying how to draw, from the `eraseData` parameter of `DrawThemeChasingArrows` (page 96), `DrawThemePopupArrow` (page 102), `DrawThemeTrack` (page 110), or `DrawThemeTrackTickMarks` (page 111) or from the `inUserData` parameter of `DrawThemeButton` (page 94).
`depth`	A signed 16-bit value. You are passed the bit depth (in bits per pixel) of the current graphics port.
`isColorDev`	A value of type `Boolean`. If `true`, indicates that you are drawing on a color device; a value of `false` indicates a monochrome device.

DISCUSSION

At the time your drawing function is called, the foreground text color and mode is already set to draw in the correct state (active or inactive) and correct color for the theme. You do not need to set the color unless you have special drawing needs. If you do have special drawing needs, you should supply the `depth` value and the value of the `isColorDevice` parameter to the function `IsThemeInColor` to determine whether or not you should draw in color. Note that the Appearance Manager calls your `MyThemeEraseProc` function for every device that the `bounds` rectangle intersects, so your application does not need to call the `DeviceLoop` function itself.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

MyThemeIteratorProc

Performs a custom response to an iteration over themes installed on a system.

Here's how to declare a theme iteration function, if you were to name the function `MyThemeIteratorProc`:

```
pascal Boolean MyThemeIteratorProc (
                        ConstStr255Param inFileName,
                        SInt16 resID,
                        Collection inThemeSettings,
                        void *inUserData);
```

`inFileName`	A value of type `ConstStr255Param`. You are passed the name of the file containing the theme being iterated upon.
`resID`	A signed 16-bit integer. You are passed the resource ID of the theme.
`inThemeSettings`	A value of type `Collection`. You are passed a reference to a collection that contains data describing attributes of the theme. Note that the Appearance Manager owns this collection, and that your application should not dispose of it.

`inUserData`	A pointer to data of any type. You are passed the value specified in the `inUserData` parameter of the function `IterateThemes` (page 73).
function result	A value of type `Boolean`. If you return `true`, `IterateThemes` continues iterating. Set to `false` to terminate the iteration.

SPECIAL CONSIDERATIONS

Your application should not open and close theme files during this call.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

MyThemeTabTitleDrawProc

Draws a tab title.

Here's how to declare a tab title drawing function, if you were to name the function `MyThemeTabTitleDrawProc`:

```
pascal void MyThemeTabTitleDrawProc (
                     const Rect *bounds,
                     ThemeTabStyle style,
                     ThemeTabDirection direction,
                     SInt16 depth,
                     Boolean isColorDev,
                     UInt32 userData);
```

`bounds`	A pointer to a structure of type `Rect`. The rectangle you are passed is set to the area in which you should draw your tab title content. Your tab title drawing function is called clipped to the rectangle in which you are allowed to draw your content; do not draw outside this region.
`style`	A value of type `ThemeTabStyle`. You are passed a constant specifying the relative position (front or non-front) and state of the tab. See "Theme Tab Style Constants" (page 224) for descriptions of possible values.

`direction`	A value of type `ThemeTabDirection`. You are passed a constant specifying the direction in which the tab is oriented. See "Theme Tab Direction Constants" (page 223) for descriptions of possible values.
`depth`	A signed 16-bit value. You are passed the bit depth (in bits per pixel) of the current graphics port.
`isColorDev`	A value of type `Boolean`. If `true`, indicates that you are drawing on a color device; a value of `false` indicates a monochrome device.
`userData`	An unsigned 32-bit value. You are passed data specifying how to draw the tab title content, from the `userData` parameter of `DrawThemeTab` (page 107).

DISCUSSION

At the time your tab title drawing function is called, the foreground text color and mode is already set to draw in the correct state (active or inactive) and correct color for the theme. You do not need to set the color unless you have special drawing needs. If you do have special drawing needs, you should supply the `depth` value and the value of the `isColorDevice` parameter to the function `IsThemeInColor` to determine whether or not you should draw the tab title content in color. Note that the Appearance Manager calls your `MyThemeTabTitleDrawProc` function for every device that the `bounds` rectangle intersects.

SPECIAL CONSIDERATIONS

The Appearance Manager draws the tab background prior to calling your tab title drawing function, so you should not erase the tab background from your title drawing function.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

MyWindowTitleDrawingProc

Draws a window title.

Here's how to declare a window title drawing function, if you were to name the function `MyWindowTitleDrawingProc`:

```
pascal void MyWindowTitleDrawingProc (
                     const Rect *bounds,
                     SInt16 depth,
                     Boolean colorDevice,
                     SInt32 userData);
```

`bounds`	A pointer to a structure of type `Rect`. The rectangle you are passed is set to the area in which you should draw your window title content. Your window title drawing function is called clipped to the rectangle in which you are allowed to draw your content; do not draw outside this region.
`depth`	A signed 16-bit value. You are passed the bit depth (in bits per pixel) of the current graphics port.
`colorDevice`	A value of type `Boolean`. If `true`, indicates that you are drawing on a color device; a value of `false` indicates a monochrome device.
`userData`	A signed 32-bit value. You are passed data specifying how to draw the window title content, from the `titleData` parameter of `DrawThemeWindowFrame` (page 135).

DISCUSSION

At the time your window title drawing function is called, the foreground text color and mode is already set to draw in the correct window state (active or inactive) and correct color for the theme. You do not need to set the color unless you have special drawing needs. If you do have special drawing needs, you should supply the `depth` value and the value of the `colorDevice` parameter to the function `IsThemeInColor` to determine whether or not you should draw the window title content in color. Note that the Appearance Manager calls your `MyWindowTitleDrawingProc` function for every device that the `bounds` rectangle intersects.

SPECIAL CONSIDERATIONS

The Appearance Manager draws the background of the window title prior to calling your window title drawing function, so you should not erase the background from this function.

VERSION NOTES

Available with Appearance Manager 1.1 and later.

Data Types

The Appearance Manager supplies the following data types:

- `ThemeDrawingState` (page 165)
- `ThemeButtonDrawInfo` (page 165)
- `ThemeTrackDrawInfo` (page 166)
- `ScrollBarTrackInfo` (page 167)
- `SliderTrackInfo` (page 168)
- `ProgressTrackInfo` (page 169)
- `ThemeWindowMetrics` (page 169)
- `ThemeIteratorUPP` (page 171)
- `ThemeEraseUPP` (page 171)
- `ThemeTabTitleDrawUPP` (page 172)
- `ThemeButtonDrawUPP` (page 173)
- `WindowTitleDrawingUPP` (page 174)
- `MenuTitleDrawingUPP` (page 175)
- `MenuItemDrawingUPP` (page 175)

ThemeDrawingState

The ThemeDrawingState type is a reference to a private structure containing information about the current state of a graphics port. You can use the ThemeDrawingState type with the function GetThemeDrawingState (page 80) to obtain the current graphics port's drawing state and with the function SetThemeDrawingState (page 84) to restore a port's drawing state. You should dispose of the memory allocated to contain a ThemeDrawingState reference by calling DisposeThemeDrawingState (page 78) or passing a value of true in the inDisposeNow parameter of SetThemeDrawingState. The ThemeDrawingState type is available with Appearance Manager 1.1 and later.

```
typedef struct OpaqueThemeDrawingState* ThemeDrawingState;
```

ThemeButtonDrawInfo

A ThemeButtonDrawInfo structure describes the changeable visual characteristics of a button. Your application can use a ThemeButtonDrawInfo structure, together with a constant of type ThemeButtonKind, to fully describe the visual characteristics of a given button type at a given point in time. See "Theme Button Kind Constants" (page 187) for a description of ThemeButtonKind values.

Your application uses the ThemeButtonDrawInfo structure in the function DrawThemeButton (page 94) to draw a theme-compliant button and in the functions GetThemeButtonRegion (page 115) and GetThemeButtonContentBounds (page 114) to obtain information about a specific button type. The ThemeButtonDrawInfo structure is available with Appearance Manager 1.1 and later.

```
struct ThemeButtonDrawInfo {
    ThemeDrawState          state;
    ThemeButtonValue        value;
    ThemeButtonAdornment    adornment;
};
typedef struct ThemeButtonDrawInfo ThemeButtonDrawInfo;
typedef ThemeButtonDrawInfo *ThemeButtonDrawInfoPtr;
```

Field descriptions

state	A value of type ThemeDrawState, specifying the state of the button, such as whether it is active, inactive, or pressed. See

	"Theme Draw State Constants" (page 199) for descriptions of possible values.
`value`	A value of type `ThemeButtonValue`, specifying the value of the button, such as, in the case of checkbox, whether it is drawn as on, off, or mixed. See "Theme Button Value Constants" (page 189) for descriptions of possible values.
`adornment`	A value of type `ThemeButtonAdornment`, specifying any supplementary characteristics of the button, such as whether it is drawn with a focus ring. See "Theme Button Adornment Constants" (page 186) for descriptions of possible values.

ThemeTrackDrawInfo

Your application fills out the applicable fields of a `ThemeTrackDrawInfo` structure to fully describe any given track control. The `ThemeTrackDrawInfo` structure is available with Appearance Manager 1.1 and later.

```
struct ThemeTrackDrawInfo {
    ThemeTrackKind          kind;
    Rect                    bounds;
    SInt32                  min;
    SInt32                  max;
    SInt32                  value;
    UInt32                  reserved;

    ThemeTrackAttributes    attributes;
    ThemeTrackEnableState   enableState;
    UInt8                   filler1; // padding

    union {
        ScrollBarTrackInfo  scrollbar;
        SliderTrackInfo     slider;
        ProgressTrackInfo   progress;
    } trackInfo;
};
typedef struct ThemeTrackDrawInfo ThemeTrackDrawInfo;
```

Field descriptions

`kind`	A value of type `ThemeTrackKind`, specifying the type of track to be drawn. See "Theme Track Kind Constants" (page 232) for descriptions of possible values.
`bounds`	A structure of type `Rect` specifying the dimensions and position of the track, in local coordinates.
`min`	A signed 32-bit integer specifying the minimum value for the track.
`max`	A signed 32-bit integer specifying the maximum value for the track.
`value`	A signed 32-bit integer specifying the current value for the track.
`reserved`	Reserved.
`attributes`	A value of type `ThemeTrackAttributes` specifying additional attributes of the track, such as whether the track has an indicator. See "Theme Track Attributes Constants" (page 231) for descriptions of possible values.
`enableState`	A value of type `ThemeTrackEnableState` specifying the current state of the track control; see "Theme Track Enable State Constants" (page 231) for descriptions of possible values.
`trackInfo`	A union of the `ScrollBarTrackInfo`, `SliderTrackInfo`, and `ProgressTrackInfo` structures. Your application fills in the structure that is appropriate for the kind of track with which you are working. See `ScrollBarTrackInfo` (page 167), `SliderTrackInfo` (page 168), and `ProgressTrackInfo` (page 169) for details on these structures.

ScrollBarTrackInfo

Your application uses the `ScrollBarTrackInfo` structure in the `ThemeTrackDrawInfo` (page 166) structure to describe the scroll bar–specific features of a given track control. The `ScrollBarTrackInfo` structure is available with Appearance Manager 1.1 and later.

```
struct ScrollBarTrackInfo {
    SInt32                  viewSize;
    ThemeTrackPressState    pressState;
};
typedef struct ScrollBarTrackInfo ScrollBarTrackInfo;
```

Field descriptions

`viewSize`	A signed 32-bit integer, specifying the size of the content being displayed. This value should be expressed in terms of the same units of measurement as are used for the minimum, maximum, and current settings of the scroll bar.
`pressState`	A value of type `ThemeTrackPressState`, specifying what in the scroll bar is currently pressed. See "Theme Track Press State Constants" (page 233) for descriptions of possible values. Pass 0 if nothing is currently pressed.

SliderTrackInfo

Your application supplies a `SliderTrackInfo` structure to the `ThemeTrackDrawInfo` (page 166) structure to describe the slider-specific features of a given track control. The `SliderTrackInfo` structure is available with Appearance Manager 1.1 and later.

```
struct SliderTrackInfo {
    ThemeThumbDirection     thumbDir;
    ThemeTrackPressState    pressState;
};
typedef struct SliderTrackInfo SliderTrackInfo;
```

Field descriptions

`thumbDir`	A value of type `ThemeThumbDirection`, specifying the direction in which the slider indicator points. See "Theme Slider Indicator Direction Constants" (page 207) for descriptions of possible values.
`pressState`	A value of type `ThemeTrackPressState`, specifying the part of the slider that is currently pressed. See "Theme Track Press State Constants" (page 233) for descriptions of possible values. Pass 0 if nothing is currently pressed.

ProgressTrackInfo

Your application supplies a `ProgressTrackInfo` structure to the `ThemeTrackDrawInfo` (page 166) structure to describe the progress bar–specific features of a given track control. The `ProgressTrackInfo` structure is available with Appearance Manager 1.1 and later.

```
struct ProgressTrackInfo {
    UInt8   phase;
};
typedef struct ProgressTrackInfo ProgressTrackInfo;
```

Field description

`phase`	A value ranging from 1 to 4, specifying the current animation phase for an indeterminate progress bar; increment this value to animate the progress bar. Set this field to 0 for a determinate progress bar.

ThemeWindowMetrics

Your application uses the `ThemeWindowMetrics` structure to inform the Appearance Manager of the dimensions of specific parts of your window. See the functions discussed in "Drawing Theme-Compliant Windows" (page 128) for specific uses of the `ThemeWindowMetrics` type. The `ThemeWindowMetrics` structure is available with Appearance Manager 1.1 and later.

```
struct ThemeWindowMetrics {
    UInt16      metricSize;
    SInt16      titleHeight;
    SInt16      titleWidth;
    SInt16      popupTabOffset;
    SInt16      popupTabWidth;
    UInt16      popupTabPosition;
};
typedef struct ThemeWindowMetrics ThemeWindowMetrics;
typedef ThemeWindowMetrics *ThemeWindowMetricsPtr;
```

Field descriptions

`metricSize`	A value specifying the size of the `ThemeWindowMetrics` structure.

`titleHeight`	A measurement in pixels of the height of the title text in the current system font, including any icon that may be present in the title region. Set this field to 0 if the window does not contain a title.
`titleWidth`	A measurement in pixels of the width of the title text in the current system font, including any icon that may be present in the title region. Set this field to 0 if the window does not contain a title.
`popupTabOffset`	A measurement in pixels of the distance that the left edge of a pop-up window's tab is offset from the left edge of the window. This value is used in conjunction with the value passed in the `popupTabPosition` field to determine the actual position of the tab. Set this field to 0 if the window is not a pop-up window.
`popupTabWidth`	A measurement in pixels of the width of a pop-up window's tab. Set this field to 0 if the window is not a pop-up window.
`popupTabPosition`	A value specifying the rule to apply when positioning a pop-up window's tab. Set this field to 0 if the window is not a pop-up window. A value of `kThemePopupTabNormalPosition` specifies that the left edge of the tab is to be drawn at the position indicated by the `popupTabOffset` field. A value of `kThemePopupTabCenterOnWindow` specifies that the tab is to be drawn centered on the window; the `popupTabOffset` field is ignored. A value of `kThemePopupTabCenterOnOffset` specifies that the tab is to be drawn centered at the position indicated by the `popupTabOffset` field.

ThemeIteratorUPP

The Appearance Manager defines the type for an application-defined theme iteration function as follows:

```
typedef pascal (Boolean, ThemeIteratorProcPtr) (
                     ConstStr255Param inFileName,
                     SInt16 resID,
                     Collection inThemeSettings,
                     void *inUserData);
```

The Appearance Manager defines the data type `ThemeIteratorUPP` to identify the universal procedure pointer for an application-defined theme iteration function:

```
typedef ThemeIteratorProcPtr ThemeIteratorUPP;
```

You typically use the `NewThemeIteratorProc` macro like this:

```
ThemeIteratorUPP myThemeIteratorUPP;
myThemeIteratorUPP = NewThemeIteratorProc(MyThemeIteratorProc);
```

You typically use the `CallThemeIteratorProc` macro like this:

```
CallThemeIteratorProc(myThemeIteratorUPP, inFileName, resID,
inThemeSettings, inUserData);
```

To implement your own theme iteration function, see `MyThemeIteratorProc` (page 160). The `ThemeIteratorProcPtr` type is available with Appearance Manager 1.1 and later.

ThemeEraseUPP

The Appearance Manager defines the type for an application-defined background drawing function as follows:

```
typedef pascal (void, ThemeEraseProcPtr) (
                     const Rect *bounds,
                     UInt32 eraseData,
                     SInt16 depth,
                     Boolean isColorDev);
```

The Appearance Manager defines the data type `ThemeEraseUPP` to identify the universal procedure pointer for an application-defined background drawing function:

```
typedef ThemeEraseProcPtr ThemeEraseUPP;
```

You typically use the `NewThemeEraseProc` macro like this:

```
ThemeEraseUPP myThemeEraseUPP;
myThemeEraseUPP = NewThemeEraseProc(MyThemeEraseProc);
```

You typically use the `CallThemeEraseProc` macro like this:

```
CallThemeEraseProc(myThemeEraseUPP, bounds, eraseData, depth,
isColorDev);
```

To implement your own background drawing function, see `MyThemeEraseProc` (page 159). The `ThemeEraseProcPtr` type is available with Appearance Manager 1.1 and later.

ThemeTabTitleDrawUPP

The Appearance Manager defines the type for an application-defined tab title drawing function as follows:

```
typedef pascal (void, ThemeTabTitleDrawProcPtr) (
                     const Rect *bounds,
                     ThemeTabStyle style,
                     ThemeTabDirection direction,
                     SInt16 depth,
                     Boolean isColorDev,
                     UInt32 userData);
```

The Appearance Manager defines the data type `ThemeTabTitleDrawUPP` to identify the universal procedure pointer for an application-defined tab title drawing function:

```
typedef ThemeTabTitleDrawProcPtr ThemeTabTitleDrawUPP;
```

You typically use the `NewThemeTabTitleDrawProc` macro like this:

```
ThemeTabTitleDrawUPP myThemeTabTitleDrawUPP;
myThemeTabTitleDrawUPP =
NewThemeTabTitleDrawProc(MyThemeTabTitleDrawProc);
```

You typically use the `CallThemeTabTitleDrawProc` macro like this:

```
CallThemeTabTitleDrawProc(myThemeTabTitleDrawUPP, bounds, style,
direction, depth, isColorDev, userData);
```

To implement your own tab title drawing function, see `MyThemeTabTitleDrawProc` (page 161). The `ThemeTabTitleDrawProcPtr` type is available with Appearance Manager 1.1 and later.

ThemeButtonDrawUPP

The Appearance Manager defines the type for an application-defined button label drawing function as follows:

```
typedef pascal (void, ThemeButtonDrawProcPtr) (
                     const Rect *bounds,
                     ThemeButtonKind kind,
                     ThemeButtonDrawInfo *info,
                     UInt32 userData,
                     SInt16 depth,
                     Boolean isColorDev);
```

The Appearance Manager defines the data type `ThemeButtonDrawUPP` to identify the universal procedure pointer for an application-defined button label drawing function:

```
typedef ThemeButtonDrawProcPtr ThemeButtonDrawUPP;
```

You typically use the `NewThemeButtonDrawProc` macro like this:

```
ThemeButtonDrawUPP myThemeButtonDrawUPP;
myThemeButtonDrawUPP = NewThemeButtonDrawProc(MyThemeButtonDrawProc);
```

You typically use the `CallThemeButtonDrawProc` macro like this:

```
CallThemeButtonDrawProc(myThemeButtonDrawUPP, bounds, kind, info,
userData, depth, isColorDev);
```

To implement your own button label drawing function, see `MyThemeButtonDrawProc` (page 157). The `ThemeButtonDrawProcPtr` type is available with Appearance Manager 1.1 and later.

WindowTitleDrawingUPP

The Appearance Manager defines the type for an application-defined window title drawing function as follows:

```
typedef pascal (void, WindowTitleDrawingProcPtr) (
                     const Rect *bounds,
                     SInt16 depth,
                     Boolean colorDevice,
                     SInt32 userData);
```

The Appearance Manager defines the data type `WindowTitleDrawingUPP` to identify the universal procedure pointer for an application-defined window title drawing function:

```
typedef WindowTitleDrawingProcPtr WindowTitleDrawingUPP;
```

You typically use the `NewWindowTitleDrawingProc` macro like this:

```
WindowTitleDrawingUPP myWindowTitleDrawingUPP;
myWindowTitleDrawingUPP =
NewWindowTitleDrawingProc(MyWindowTitleDrawingProc);
```

You typically use the `CallWindowTitleDrawingProc` macro like this:

```
CallWindowTitleDrawingProc(myWindowTitleDrawingUPP, bounds, depth,
colorDevice, userData);
```

To implement your own window title drawing function, see `MyWindowTitleDrawingProc` (page 162). The `WindowTitleDrawingProcPtr` type is available with Appearance Manager 1.1 and later.

MenuTitleDrawingUPP

The Appearance Manager declares the type for an application-defined menu title drawing function as follows:

```
typedef pascal (void, MenuTitleDrawingProcPtr)(
                     const Rect *inBounds,
                     SInt16 inDepth,
                     Boolean inIsColorDevice,
                     SInt32 inUserData);
```

The Appearance Manager defines the data type `MenuTitleDrawingUPP` to identify the universal procedure pointer for an application-defined menu title drawing function:

```
typedef UniversalProcPtr MenuTitleDrawingUPP;
```

You typically use the `NewMenuTitleDrawingProc` macro like this:

```
MenuTitleDrawingUPP myMenuTitleDrawingUPP;
myMenuTitleDrawingUPP = NewMenuTitleDrawingProc(MyMenuTitleDrawingProc);
```

You typically use the `CallMenuTitleDrawingProc` macro like this:

```
CallMenuTitleDrawingProc(myMenuTitleDrawingUPP, inBounds, inDepth,
inIsColorDevice, inUserData);
```

See `MyMenuTitleDrawingProc` (page 156) for a discussion of how to declare a menu title drawing function.

MenuItemDrawingUPP

The Appearance Manager declares the type for an application-defined menu item drawing function as follows:

```
typedef pascal (void, MenuItemDrawingProcPtr)(
                     const Rect *inBounds,
                     SInt16 inDepth,
                     Boolean inIsColorDevice,
                     SInt32 inUserData);
```

The Appearance Manager defines the data type `MenuItemDrawingUPP` to identify the universal procedure pointer for an application-defined menu item drawing function:

```
typedef UniversalProcPtr MenuItemDrawingUPP;
```

You typically use the `NewMenuItemDrawingProc` macro like this:

```
MenuItemDrawingUPP myMenuItemDrawingUPP;
myMenuItemDrawingUPP = NewMenuItemDrawingProc(MyMenuItemDrawingProc);
```

You typically use the `CallMenuItemDrawingProc` macro like this:

```
CallMenuItemDrawingProc(myMenuItemDrawingUPP, inBounds, inDepth,
inIsColorDevice, inUserData);
```

See `MyMenuItemDrawingProc` (page 155) for a discussion of how to declare a menu item drawing function.

Constants

The following constants are available with the Appearance Manager:

- "Appearance Manager Apple Event Constants" (page 177)
- "Appearance Manager File Type Constants" (page 179)
- "Theme Background Kind Constants" (page 179)
- "Theme Brush Constants" (page 180)
- "Theme Button Adornment Constants" (page 186)
- "Theme Button Kind Constants" (page 187)
- "Theme Button Value Constants" (page 189)
- "Theme Checkbox Style Constants" (page 190)
- "Theme Collection Tags" (page 190)
- "Theme Cursor Constants" (page 194)
- "Theme Drag Sound Constants" (page 197)

- "Theme Draw State Constants" (page 199)
- "Theme Font ID Constants" (page 200)
- "Theme Menu Bar State Constants" (page 200)
- "Theme Menu Item Type Constants" (page 201)
- "Theme Menu State Constants" (page 203)
- "Theme Menu Type Constants" (page 203)
- "Theme Pop-Up Arrow Orientation Constants" (page 204)
- "Theme Pop-Up Arrow Size Constants" (page 205)
- "Theme Scroll Bar Arrow Style Constants" (page 205)
- "Theme Scroll Box Style Constants" (page 206)
- "Theme Size Box Direction Constants" (page 206)
- "Theme Slider Indicator Direction Constants" (page 207)
- "Theme Sound Constants" (page 208)
- "Theme Sound Mask Constants" (page 223)
- "Theme Tab Direction Constants" (page 223)
- "Theme Tab Style Constants" (page 224)
- "Theme Text Color Constants" (page 225)
- "Theme Title Bar Item Constants" (page 230)
- "Theme Track Attributes Constants" (page 231)
- "Theme Track Enable State Constants" (page 231)
- "Theme Track Kind Constants" (page 232)
- "Theme Track Press State Constants" (page 233)
- "Theme Window Attribute Constants" (page 235)
- "Theme Window Type Constants" (page 236)

Appearance Manager Apple Event Constants

Under Appearance Manager 1.1 and later, when the user changes the current appearance (that is, when a theme switch occurs), the Appearance Manager may send any of the following Apple events to all running applications that are high-level event aware and which are registered as clients of the Appearance Manager. Your application registers itself with the Appearance Manager by calling the function `RegisterAppearanceClient` (page 68).

Because typical results of a theme switch might include a change in menu bar height or window structure dimensions, as well as changes to the system fonts, colors, and patterns that are currently in use, applications should listen for and respond to the Appearance Manager Apple events under most circumstances. Note that none of the Appearance Manager Apple events have parameters and that the return value for each is ignored.

```
enum {
    kAppearanceEventClass       = 'appr',
    kAEAppearanceChanged        = 'thme',
    kAESystemFontChanged        = 'sysf',
    kAESmallSystemFontChanged   = 'ssfn',
    kAEViewsFontChanged         = 'vfnt'
};
```

Constant descriptions

`kAppearanceEventClass`
: The event class of Appearance Manager Apple events.

`kAEAppearanceChanged`
: The ID of the event indicating the current appearance has changed.

`kAESystemFontChanged`
: The ID of the event indicating the current system font has changed.

`kAESmallSystemFontChanged`
: The ID of the event indicating the current small system font has changed.

`kAEViewsFontChanged`
: The ID of the event indicating the current views font has changed.

Appearance Manager File Type Constants

Under Appearance Manager 1.1 and later, the following constants are used to identify the various Appearance Manager file types.

```
enum {
    kThemeDataFileType          = 'thme',
    kThemePlatinumFileType      = 'pltn',
    kThemeCustomThemesFileType  = 'scen',
    kThemeSoundTrackFileType    = 'tsnd'
};
```

Constant descriptions

`kThemeDataFileType`
The file type of appearances other than the platinum appearance.

`kThemePlatinumFileType`
The file type of the platinum appearance.

`kThemeCustomThemesFileType`
The file type of a file that contains user-defined themes. See `SetTheme` (page 73) for a discussion of defining your own theme.

`kThemePlatinumFileType`
The file type of a theme soundtrack.

Theme Background Kind Constants

You can pass a constant of type `ThemeBackgroundKind` to the function `ApplyThemeBackground` (page 76) to specify that an embedded object have a background consistent with the current theme and an object in which it is visually embedded. The `ThemeBackgroundKind` constants are available with Appearance Manager 1.1 and later.

```
enum {
    kThemeBackgroundTabPane                = 1,
    kThemeBackgroundPlacard                = 2,
    kThemeBackgroundWindowHeader           = 3,
    kThemeBackgroundListViewWindowHeader   = 4
};
typedef UInt32 ThemeBackgroundKind;
```

Constant descriptions

`kThemeBackgroundTabPane`
: The background for a tab pane.

`kThemeBackgroundPlacard`
: The background for a placard.

`kThemeBackgroundWindowHeader`
: The background for a window header.

`kThemeBackgroundListViewWindowHeader`
: The background for a window list view header.

Theme Brush Constants

The Appearance Manager provides the underlying support for `RGB` color data and overrides System 7 color tables such as `'cctb'` and `'mctb'` with an abstract mechanism that allows colors and patterns to be coordinated with the current theme. You can pass constants of type `ThemeBrush` in the `inBrush` parameter of `SetThemeBackground` (page 83), `SetThemePen` (page 85), and `SetThemeWindowBackground` (page 143) to specify that the Appearance Manager substitute whatever the appropriate color or pattern is for a given human interface element in the current theme.

```
enum {
    kThemeBrushDialogBackgroundActive           = 1,
    kThemeBrushDialogBackgroundInactive         = 2,
    kThemeBrushAlertBackgroundActive            = 3,
    kThemeBrushAlertBackgroundInactive          = 4,
    kThemeBrushModelessDialogBackgroundActive   = 5,
    kThemeBrushModelessDialogBackgroundInactive = 6,
    kThemeBrushUtilityWindowBackgroundActive    = 7,
    kThemeBrushUtilityWindowBackgroundInactive  = 8,
    kThemeBrushListViewSortColumnBackground     = 9,
    kThemeBrushListViewBackground               = 10,
    kThemeBrushIconLabelBackground              = 11,
    kThemeBrushListViewSeparator                = 12,
    kThemeBrushChasingArrows                    = 13,
    kThemeBrushDragHilite                       = 14,
    kThemeBrushDocumentWindowBackground         = 15,
    kThemeBrushFinderWindowBackground           = 16,
    /* Brushes available in Appearance Manager 1.1 or later */
    kThemeBrushScrollBarDelimiterActive         = 17,
    kThemeBrushScrollBarDelimiterInactive       = 18,
    kThemeBrushFocusHighlight                   = 19,
    kThemeBrushPopupArrowActive                 = 20,
    kThemeBrushPopupArrowPressed                = 21,
    kThemeBrushPopupArrowInactive               = 22,
    kThemeBrushAppleGuideCoachmark              = 23,
    kThemeBrushIconLabelBackgroundSelected      = 24,
    kThemeBrushStaticAreaFill                   = 25,
    kThemeBrushActiveAreaFill                   = 26,
    kThemeBrushButtonFrameActive                = 27,
    kThemeBrushButtonFrameInactive              = 28,
    kThemeBrushButtonFaceActive                 = 29,
    kThemeBrushButtonFaceInactive               = 30,
    kThemeBrushButtonFacePressed                = 31,
    kThemeBrushButtonActiveDarkShadow           = 32,
    kThemeBrushButtonActiveDarkHighlight        = 33,
    kThemeBrushButtonActiveLightShadow          = 34,
    kThemeBrushButtonActiveLightHighlight       = 35,
    kThemeBrushButtonInactiveDarkShadow         = 36,
    kThemeBrushButtonInactiveDarkHighlight      = 37,
    kThemeBrushButtonInactiveLightShadow        = 38,
    kThemeBrushButtonInactiveLightHighlight     = 39,
    kThemeBrushButtonPressedDarkShadow          = 40,
    kThemeBrushButtonPressedDarkHighlight       = 41,
    kThemeBrushButtonPressedLightShadow         = 42,
    kThemeBrushButtonPressedLightHighlight      = 43,
    kThemeBrushBevelActiveLight                 = 44,
    kThemeBrushBevelActiveDark                  = 45,
    kThemeBrushBevelInactiveLight               = 46,
    kThemeBrushBevelInactiveDark                = 47,
    kThemeBrushBlack                            = -1,
    kThemeBrushWhite                            = -2
};
typedef SInt16 ThemeBrush;
```

Constant descriptions

`kThemeBrushDialogBackgroundActive`
An active dialog box's background color or pattern.

`kThemeBrushDialogBackgroundInactive`
: An inactive dialog box's background color or pattern.

`kThemeBrushAlertBackgroundActive`
: An active alert box's background color or pattern.

`kThemeBrushAlertBackgroundInactive`
: An inactive alert box's background color or pattern.

`kThemeBrushModelessDialogBackgroundActive`
: An active modeless dialog box's background color or pattern.

`kThemeBrushModelessDialogBackgroundInactive`
: An inactive modeless dialog box's background color or pattern.

`kThemeBrushUtilityWindowBackgroundActive`
: An active utility window's background color or pattern.

`kThemeBrushUtilityWindowBackgroundInactive`
: An inactive utility window's background color or pattern.

`kThemeBrushListViewSortColumnBackground`
: The background color or pattern of the list view column that is being sorted upon.

`kThemeBrushListViewBackground`
: The background color or pattern of a list view column that is not being sorted upon.

`kThemeBrushIconLabelBackground`
: An icon label's color or pattern.

`kThemeBrushListViewSeparator`
: The color or pattern of the horizontal lines that separate rows of items in list view columns.

`kThemeBrushChasingArrows`
: Asynchronous arrows' color or pattern.

`kThemeBrushDragHilite`
: The color or pattern used to indicate that an element is a valid drag-and-drop destination

`kThemeBrushDocumentWindowBackground`
: A document window's background color or pattern.

`kThemeBrushFinderWindowBackground`
: A Finder window's background color or pattern. Generally,

you should not use this constant unless you are trying to create a window that matches a Finder window.

`kThemeBrushScrollBarDelimiterActive`
The color or pattern used to outline the sides of an active scroll bar. Available with Appearance Manager 1.1 and later.

`kThemeBrushScrollBarDelimiterInactive`
The color or pattern used to outline the sides of an inactive scroll bar. Available with Appearance Manager 1.1 and later.

`kThemeBrushFocusHighlight`
The color or pattern of the focus ring around an element that is selected. Available with Appearance Manager 1.1 and later.

`kThemeBrushPopupArrowActive`
The color or pattern of the arrow on an active pop-up menu button. Available with Appearance Manager 1.1 and later.

`kThemeBrushPopupArrowPressed`
The color or pattern of the arrow on a pop-up menu button that is being clicked on by the user. Available with Appearance Manager 1.1 and later.

`kThemeBrushPopupArrowInactive`
The color or pattern of the arrow on an inactive pop-up menu button. Available with Appearance Manager 1.1 and later.

`kThemeBrushAppleGuideCoachmark`
The color or pattern of an Apple Guide coachmark. Available with Appearance Manager 1.1 and later.

`kThemeBrushIconLabelBackgroundSelected`
The color or pattern of the background of an icon's label area, when the icon is selected. Available with Appearance Manager 1.1 and later.

`kThemeBrushStaticAreaFill`
The background color or pattern of an element that does not support user interaction. Available with Appearance Manager 1.1 and later.

`kThemeBrushActiveAreaFill`
The color or pattern of an element that supports user

interaction. Available with Appearance Manager 1.1 and later.

`kThemeBrushButtonFrameActive`
: The color or pattern that outlines an active button. Your application should draw the button outline outside the edge of the button. Available with Appearance Manager 1.1 and later.

`kThemeBrushButtonFrameInactive`
: The color or pattern that outlines an inactive button. Your application should draw the button outline outside the edge of the button. Available with Appearance Manager 1.1 and later.

`kThemeBrushButtonFaceActive`
: The color or pattern of the face of an active button. Available with Appearance Manager 1.1 and later.

`kThemeBrushButtonFaceInactive`
: The color or pattern of the face of an inactive button. Available with Appearance Manager 1.1 and later.

`kThemeBrushButtonFacePressed`
: The color or pattern of the face of a button that is being clicked on by the user. Available with Appearance Manager 1.1 and later.

`kThemeBrushButtonActiveDarkShadow`
: For an active button with a 2-pixel-wide edge, the color or pattern of the bottom and right sides of the outer ring of the edge. Available with Appearance Manager 1.1 and later.

`kThemeBrushButtonActiveDarkHighlight`
: For an active button with a 2-pixel-wide edge, the color or pattern of the top and left sides of the outer ring of the edge. Available with Appearance Manager 1.1 and later.

`kThemeBrushButtonActiveLightShadow`
: For an active button with a 2-pixel-wide edge, the color or pattern of the bottom and right sides of the inner ring of the edge. For an active button with a 1-pixel-wide edge, the color or pattern of the bottom and right sides of the edge. Available with Appearance Manager 1.1 and later.

`kThemeBrushButtonActiveLightHighlight`
: For an active button with a 2-pixel-wide edge, the color or pattern of the top and left sides of the inner ring of the

edge. For an active button with a 1-pixel-wide edge, the color or pattern of the top and left sides of the edge. Available with Appearance Manager 1.1 and later.

`kThemeBrushButtonInactiveDarkShadow`
For an inactive button with a 2-pixel-wide edge, the color or pattern of the bottom and right sides of the outer ring of the edge. Available with Appearance Manager 1.1 and later.

`kThemeBrushButtonInactiveDarkHighlight`
For an inactive button with a 2-pixel-wide edge, the color or pattern of the top and left sides of the outer ring of the edge. Available with Appearance Manager 1.1 and later.

`kThemeBrushButtonInactiveLightShadow`
For an inactive button with a 2-pixel-wide edge, the color or pattern of the bottom and right sides of the inner ring of the edge. For an inactive button with a 1-pixel-wide edge, the color or pattern of the bottom and right sides of the edge. Available with Appearance Manager 1.1 and later.

`kThemeBrushButtonInactiveLightHighlight`
For an inactive button with a 2-pixel-wide edge, the color or pattern of the top and left sides of the inner ring of the edge. For an inactive button with a 1-pixel-wide edge, the color or pattern of the top and left sides of the edge. Available with Appearance Manager 1.1 and later.

`kThemeBrushButtonPressedDarkShadow`
For a button with a 2-pixel-wide edge that is being clicked on by the user, the color or pattern of the bottom and right sides of the outer ring of the edge. Available with Appearance Manager 1.1 and later.

`kThemeBrushButtonPressedDarkHighlight`
For a button with a 2-pixel-wide edge that is being clicked on by the user, the color or pattern of the top and left sides of the outer ring of the edge. Available with Appearance Manager 1.1 and later.

`kThemeBrushButtonPressedLightShadow`
For a button with a 2-pixel-wide edge that is being clicked on by the user, the color or pattern of the bottom and right sides of the inner ring of the edge. For a button with a 1-pixel-wide edge that is being clicked on by the user, the

	color or pattern of the bottom and right sides of the edge. Available with Appearance Manager 1.1 and later.
`kThemeBrushButtonPressedLightHighlight`	For a button with a 2-pixel-wide edge that is being clicked on by the user, the color or pattern of the top and left sides of the inner ring of the edge. For a button with a 1-pixel-wide edge that is being clicked on by the user, the color or pattern of the top and left sides of the edge. Available with Appearance Manager 1.1 and later.
`kThemeBrushBevelActiveLight`	For an active bevel button, the color or pattern of the top and left sides of the bevel. Available with Appearance Manager 1.1 and later.
`kThemeBrushBevelActiveDark`	For an active bevel button, the color or pattern of the bottom and right sides of the bevel. Available with Appearance Manager 1.1 and later.
`kThemeBrushBevelInactiveLight`	For an inactive bevel button, the color or pattern of the top and left sides of the bevel. Available with Appearance Manager 1.1 and later.
`kThemeBrushBevelInactiveDark`	For an inactive bevel button, the color or pattern of the bottom and right sides of the bevel. Available with Appearance Manager 1.1 and later.
`kThemeBrushBlack`	Black; this color does not change from theme to theme. You may use this constant instead of specifying a direct RGB value. Available with Appearance Manager 1.1 and later.
`kThemeBrushWhite`	White; this color does not change from theme to theme. You may use this constant instead of specifying a direct RGB value. Available with Appearance Manager 1.1 and later.

Theme Button Adornment Constants

The `ThemeButtonAdornment` enumeration defines masks your application can use in the `ThemeButtonDrawInfo` (page 165) structure to specify that button controls are drawn with the appropriate human interface characteristics. The

`ThemeButtonAdornment` constants are available with Appearance Manager 1.1 and later.

```
enum {
    kThemeAdornmentNone                 = 0,
    kThemeAdornmentDefault              = (1 << 0),
    kThemeAdornmentFocus                = (1 << 2),
    kThemeAdornmentRightToLeft          = (1 << 4),
    kThemeAdornmentDrawIndicatorOnly    = (1 << 5)
};
typedef UInt16 ThemeButtonAdornment;
```

Constant descriptions

`kThemeAdornmentNone`
: If no bits are set, the button is drawn with no adornment.

`kThemeAdornmentDefault`
: If the bit specified by this mask is set, a default button ring is drawn. This constant applies to push button controls only.

`kThemeAdornmentFocus`
: If the bit specified by this mask is set, a focus ring is drawn.

`kThemeAdornmentRightToLeft`
: If the bit specified by this mask is set, the button is drawn in a right-to-left orientation.

`kThemeAdornmentDrawIndicatorOnly`
: If the bit specified by this mask is set, only the button is drawn, not its label. This characteristic applies to radio buttons, checkboxes, and disclosure triangles.

Theme Button Kind Constants

You can pass constants of type `ThemeButtonKind` to the function `DrawThemeButton` (page 94) to draw a theme-compliant button of a specific type. You can also pass `ThemeButtonKind` constants to the functions `GetThemeButtonRegion` (page 115) and `GetThemeButtonContentBounds` (page 114) to retrieve information about a specific button type. The `ThemeButtonKind` constants are available with Appearance Manager 1.1 and later.

```
enum {
    kThemePushButton        = 0,
    kThemeCheckBox          = 1,
    kThemeRadioButton       = 2,
    kThemeBevelButton       = 3,
    kThemeArrowButton       = 4,
    kThemePopupButton       = 5,
    kThemeDisclosureButton  = 6,
    kThemeIncDecButton      = 7,
    kThemeSmallBevelButton  = 8,
    kThemeMediumBevelButton = 3,
    kThemeLargeBevelButton  = 9
};
typedef UInt16 ThemeButtonKind;
```

Constant descriptions

`kThemePushButton`	Identifies a push button.
`kThemeCheckBox`	Identifies a checkbox.
`kThemeRadioButton`	Identifies a radio button.
`kThemeBevelButton`	Identifies a bevel button with a medium-width bevel; this value is the same as `kThemeMediumBevelButton`.
`kThemeArrowButton`	Identifies an arrow button. This button has the appearance of a single button containing small upward- and downward-pointing triangles drawn back to back; the typical use of this button is with an editable text field to create an editable pop-up menu. This button should not be confused with an increment/decrement button.
`kThemePopupButton`	Identifies a pop-up menu button. This button has the appearance of a single button made of two parts: a menu item text part and an arrow part.
`kThemeDisclosureButton`	Identifies a disclosure triangle.
`kThemeIncDecButton`	Identifies an increment/decrement or “little arrows” button. This button has the appearance of two separate buttons—one containing an upward-pointing triangle and the other containing a downward-pointing triangle—placed back to back. This button should not be confused with the arrow button.

`kThemeSmallBevelButton`
Identifies a bevel button with a small-width bevel.

`kThemeMediumBevelButton`
Identifies a bevel button with a medium-width bevel; this value is the same as `kThemeBevelButton`.

`kThemeLargeBevelButton`
Identifies a bevel button with a large-width bevel.

Theme Button Value Constants

You can use constants of type `ThemeButtonValue` in the `ThemeButtonDrawInfo` (page 165) structure to specify that button controls are drawn with the correct values. The `ThemeButtonValue` constants are available with Appearance Manager 1.1 and later.

```
enum {
    kThemeButtonOff         = 0,
    kThemeButtonOn          = 1,
    kThemeButtonMixed       = 2,
    kThemeDisclosureRight   = 0,
    kThemeDisclosureDown    = 1,
    kThemeDisclosureLeft    = 2
};
typedef UInt16 ThemeButtonValue;
```

Constant descriptions

`kThemeButtonOff` Identifies a button that is not selected.

`kThemeButtonOn` Identifies a button that is selected.

`kThemeButtonMixed` Identifies a button that is in the mixed state, indicating that a setting is on for some elements in a selection and off for others. This value typically applies to checkboxes and radio buttons.

`kThemeDisclosureRight`
Identifies a disclosure triangle that is pointing to the right.

`kThemeDisclosureDown`
Identifies a disclosure triangle that is pointing down.

`kThemeDisclosureLeft`
Identifies a disclosure triangle that is pointing to the left.

Theme Checkbox Style Constants

You can call the function `GetThemeCheckBoxStyle` (page 116) to obtain the type of checkbox mark being used in the current theme. The `ThemeCheckBoxStyle` constants are available with Appearance Manager 1.1 and later.

```
enum {
    kThemeCheckBoxClassicX                = 0,
    kThemeCheckBoxCheckMark               = 1
};
typedef UInt16 ThemeCheckBoxStyle;
```

Constant descriptions

`kThemeCheckBoxClassicX`
An “X” type of checkbox mark.

`kThemeCheckBoxCheckMark`
A checkmark type of checkbox mark.

Theme Collection Tags

Your application may use the following collection tags with the functions `SetTheme` (page 73) and `GetTheme` (page 70) to access aspects of a theme. The data type contained in each of the collection items accessed is noted below. Theme collection tags are available with Appearance Manager 1.1 and later. See *Inside Macintosh: QuickDraw GX Environment and Utilities* for a discussion of collections items.

```
enum {
    kThemeNameTag                    = 'name',
    kThemeAppearanceFileNameTag      = 'thme',
    kThemeVariantNameTag             = 'varn',
    kThemeSystemFontTag              = 'lgsf',
    kThemeSmallSystemFontTag         = 'smsf',
    kThemeViewsFontTag               = 'vfnt',
    kThemeViewsFontSizeTag           = 'vfsz',
    kThemeDesktopPatternNameTag      = 'patn',
    kThemeDesktopPatternTag          = 'patt',
    kThemeDesktopPictureNameTag      = 'dpnm',
    kThemeDesktopPictureAliasTag     = 'dpal',
    kThemeDesktopPictureAlignmentTag= 'dpan',
    kThemeHighlightColorNameTag     = 'hcnm',
    kThemeHighlightColorTag         = 'hcol',
    kThemeExamplePictureIDTag       = 'epic',
    kThemeSoundsEnabledTag          = 'snds',
    kThemeSoundTrackNameTag         = 'sndt',
    kThemeSoundMaskTag              = 'smsk',
    kThemeUserDefinedTag            = 'user',
    kThemeScrollBarArrowStyleTag    = 'sbar',
    kThemeScrollBarThumbStyleTag    = 'sbth',
    kThemeSmoothFontEnabledTag      = 'smoo',
    kThemeSmoothFontMinSizeTag      = 'smos',
    kThemeDblClickCollapseTag       = 'coll'
};
```

Constant descriptions

`kThemeNameTag`
: Identifies a collection item containing the name of the theme, e.g. "Mac OS Default". The Appearance Manager only uses this collection item to identify themes within the Appearance control panel, so the `GetTheme` function does not return this collection item. To specify a theme name, you must create a new collection item of this type before calling the function `SetTheme`.
Collection data type: `Str255`

`kThemeAppearanceFileNameTag`
: Identifies a collection item containing the name of the appearance, e.g. "Apple platinum".
Collection data type: `Str255`

`kThemeVariantNameTag`
: Identifies a collection item containing the color variation used for menus and controls in the theme.
Collection data type: `Str255`

`kThemeSystemFontTag`
: Identifies a collection item containing the name of the large system font for the theme.
Collection data type: `Str255`

`kThemeSmallSystemFontTag`
: Identifies a collection item containing the name of the small system font for the theme.
Collection data type: `Str255`

`kThemeViewsFontTag`
: Identifies a collection item containing the name of the views font for the theme.
Collection data type: `Str255`

`kThemeViewsFontSizeTag`
: Identifies a collection item containing the size of the views font for the theme.
Collection data type: `SInt16`

`kThemeDesktopPatternNameTag`
: Identifies a collection item containing the name of the desktop pattern for the theme.
Collection data type: `Str255`

`kThemeDesktopPatternTag`
: Identifies a collection item containing a flattened version of the desktop pattern for the theme.
Collection data type: variable-length data

`kThemeDesktopPictureNameTag`
: Identifies a collection item containing the name of the desktop picture for the theme.
Collection data type: `Str255`

`kThemeDesktopPictureAliasTag`
: Identifies a collection item containing an alias handle for the desktop picture for the theme.
Collection data type: `AliasHandle`

`kThemeDesktopPictureAlignmentTag`
: Identifies a collection item containing a value specifying how to position the desktop picture for the theme. Possible values are `kTiledOnScreen` (the picture draws repeatedly), `kCenterOnScreen` (the picture is its actual size, or clipped if necessary, with the desktop pattern showing to the side of the picture if it is smaller than the desktop), `kFitToScreen` (the picture is reduced if necessary), `kFillScreen` (the picture's aspect ratio is altered if necessary), and `kUseBestGuess` (the picture is automatically positioned).
Collection data type: `UInt32`

`kThemeHighlightColorNameTag`
: Identifies a collection item containing the name of the text highlight color for the theme.
Collection data type: `Str255`

`kThemeHighlightColorTag`
: Identifies a collection item containing the text highlight color for the theme.
Collection data type: an `RGBColor` structure

`kThemeExamplePictureIDTag`
: Identifies a collection item containing the ID of the example picture for the theme.
Collection data type: `SInt16`

`kThemeSoundsEnabledTag`
: Identifies a collection item specifying whether theme sounds are enabled for the theme.
Collection data type: `Boolean`

`kThemeSoundTrackNameTag`
: Identifies a collection item containing the name of the soundtrack for the theme.
Collection data type: `Str255`

`kThemeSoundMaskTag`
: Identifies a collection item containing an unsigned 32-bit integer whose bits are set to reflect the classes of sounds that are enabled for a theme. Possibilities include sounds for menus, windows, controls, and the Finder. See "Theme Sound Mask Constants" (page 223) for descriptions of possible sound mask values.
Collection data type: `UInt32`

`kThemeUserDefinedTag`
: Identifies a collection item specifying whether the theme is user-defined; the value contained in a `kThemeUserDefinedTag` collection should always be `true` if the `kThemeUserDefinedTag` collection is present. The Appearance Manager uses this collection item to identify themes that the user can delete. Note that the `GetTheme` function does not return this collection item.
Collection data type: `Boolean`

`kThemeScrollBarArrowStyleTag`
: Identifies a collection item containing a value of type `ThemeScrollBarArrowStyle` identifying the type of scroll bar arrows used in the theme.
Collection data type: `ThemeScrollBarArrowStyle`

`kThemeScrollBarThumbStyleTag`
: Identifies a collection item containing a value of type `ThemeScrollBarThumbStyle` identifying the type of scroll boxes used in the theme.
Collection data type: `ThemeScrollBarThumbStyle`

`kThemeSmoothFontEnabledTag`
: Identifies a collection item specifying whether font smoothing is enabled in the theme.
Collection data type: `Boolean`

`kThemeSmoothFontMinSizeTag`
: Identifies a collection item containing the minimum point size at which font smoothing may be enabled in the theme. Possible values range from 12 to 24, inclusive.
Collection data type: `UInt16`

`kThemeDblClickCollapseTag`
: Identifies a collection item specifying whether the ability to double-click to collapse a window is enabled for the theme.
Collection data type: `Boolean`

Theme Cursor Constants

You can pass constants of type `ThemeCursor` to the functions `SetThemeCursor` (page 91) and `SetAnimatedThemeCursor` (page 90) to specify the category of cursor to be displayed for your application. The Appearance Manager substitutes the theme-specific instance of the cursor for the cursor category as is appropriate. The `ThemeCursor` constants are available with Appearance Manager 1.1 and later.

```
enum {
    kThemeArrowCursor                   = 0,
    kThemeCopyArrowCursor               = 1,
    kThemeAliasArrowCursor              = 2,
    kThemeContextualMenuArrowCursor     = 3,
    kThemeIBeamCursor                   = 4,
    kThemeCrossCursor                   = 5,
    kThemePlusCursor                    = 6,
    kThemeWatchCursor                   = 7,
    kThemeClosedHandCursor              = 8,
    kThemeOpenHandCursor                = 9,
    kThemePointingHandCursor            = 10,
    kThemeCountingUpHandCursor          = 11,
    kThemeCountingDownHandCursor        = 12,
    kThemeCountingUpAndDownHandCursor   = 13,
    kThemeSpinningCursor                = 14,
    kThemeResizeLeftCursor              = 15,
    kThemeResizeRightCursor             = 16,
    kThemeResizeLeftRightCursor         = 17
};
typedef UInt32 ThemeCursor;
```

Constant descriptions

`kThemeArrowCursor`	The cursor identified by this constant is typically used as the standard cursor.
`kThemeCopyArrowCursor`	The cursor identified by this constant is typically used when the cursor is over a location where a drag action would initiate a copy.
`kThemeAliasArrowCursor`	The cursor identified by this constant is typically used when the cursor is over a location where a drag action would create an alias or link.
`kThemeContextualMenuArrowCursor`	The cursor identified by this constant is typically used when the Control key is being pressed and the cursor is over a location where a contextual menu can be activated.
`kThemeIBeamCursor`	The cursor identified by this constant is typically used when the cursor is over an area where the user can select text.
`kThemeCrossCursor`	The cursor identified by this constant is typically used when the cursor is over an area where the user can draw graphics.
`kThemePlusCursor`	The cursor identified by this constant is typically used when the cursor is over an area where the user can select table cells.
`kThemeWatchCursor`	The cursor identified by this constant is typically used to indicate that an operation is in progress. You can animate this cursor so that a hand of the watch appears to move.
`kThemeClosedHandCursor`	The cursor identified by this constant is typically used to

indicate that an object has been grabbed and is being moved by the user.

`kThemeOpenHandCursor`
: The cursor identified by this constant is typically used to indicate that an object may be grabbed or moved by the user.

`kThemePointingHandCursor`
: The cursor identified by this constant has the appearance of a pointing hand. You would typically use this constant to indicate that the user may select an object by pressing the mouse button.

`kThemeCountingUpHandCursor`
: The cursor identified by this constant is typically used to indicate that an operation is in progress. You can animate this cursor so that the fingers appear to open from the palm one by one.

`kThemeCountingDownHandCursor`
: The cursor identified by this constant is typically used to indicate that an operation is in progress. You can animate this cursor so that the fingers appear to fold into the palm one by one.

`kThemeCountingUpAndDownHandCursor`
: The cursor identified by this constant is typically used to indicate that an operation is in progress. You can animate this cursor so that the fingers appear to alternate between opening from the palm one by one and folding into the palm one by one.

`kThemeSpinningCursor`
: The cursor identified by this constant is typically used to indicate that an operation is in progress.

`kThemeResizeLeftCursor`
: The cursor identified by this constant is typically used to indicate that an object may be resized by dragging to the left.

`kThemeResizeRightCursor`
: The cursor identified by this constant is typically used to indicate that an object may be resized by dragging to the right.

`kThemeResizeLeftRightCursor`
: The cursor identified by this constant is typically used to indicate that an object may be resized in either direction horizontally.

Theme Drag Sound Constants

Your application can pass constants of type `ThemeDragSoundKind` to the function `BeginThemeDragSound` (page 88) to play a theme-specific sound when a user drags an interface object or otherwise holds the mouse button down for an extended action. Dragging sounds are looped for the duration of the drag and cease when your application calls `EndThemeDragSound` (page 88) when the drag has finished. Only one drag sound may occur at a time. The `ThemeDragSoundKind` constants are available with Appearance Manager 1.1 and later.

```
enum {
    kThemeDragSoundNone                        = 0,
    kThemeDragSoundMoveWindow                  = 'wmov',
    kThemeDragSoundGrowWindow                  = 'wgro',
    kThemeDragSoundMoveUtilWindow              = 'umov',
    kThemeDragSoundGrowUtilWindow              = 'ugro',
    kThemeDragSoundMoveDialog                  = 'dmov',
    kThemeDragSoundMoveAlert                   = 'amov',
    kThemeDragSoundMoveIcon                    = 'imov',
    kThemeDragSoundSliderThumb                 = 'slth',
    kThemeDragSoundSliderGhost                 = 'slgh',
    kThemeDragSoundScrollBarThumb              = 'sbth',
    kThemeDragSoundScrollBarGhost              = 'sbgh',
    kThemeDragSoundScrollBarArrowDecreasing = 'sbad',
    kThemeDragSoundScrollBarArrowIncreasing = 'sbai',
    kThemeDragSoundDragging                    = 'drag'
};
typedef OSType ThemeDragSoundKind;
```

Constant descriptions

`kThemeDragSoundNone`
: Specifies that no drag sound is used.

`kThemeDragSoundMoveWindow`
: Specifies a sound to be played when the user moves a document window.

`kThemeDragSoundGrowWindow`
: Specifies a sound to be played when the user resizes a window by dragging the size box.

`kThemeDragSoundMoveUtilWindow`
: Specifies a sound to be played when the user moves a utility window.

`kThemeDragSoundGrowUtilWindow`
: Specifies a sound to be played when the user resizes a utility window by dragging the size box.

`kThemeDragSoundMoveDialog`
: Specifies a sound to be played when the user moves a dialog box.

`kThemeDragSoundMoveAlert`
: Specifies a sound to be played when the user moves an alert box.

`kThemeDragSoundMoveIcon`
: Specifies a sound to be played when the user moves an icon.

`kThemeDragSoundSliderThumb`
: Specifies a sound to be played when the user drags the indicator of a slider control that supports live feedback.

`kThemeDragSoundSliderGhost`
: Specifies a sound to be played when the user drags the indicator of a slider control that does not support live feedback.

`kThemeDragSoundScrollBarThumb`
: Specifies a sound to be played when the user drags a scroll box belonging to a scroll bar that supports live feedback.

`kThemeDragSoundScrollBarGhost`
: Specifies a sound to be played when the user drags a scroll box belonging to a scroll bar that does not support live feedback.

`kThemeDragSoundScrollBarArrowDecreasing`
: Specifies a sound to be played when the user presses and holds the mouse button while the cursor is over the scroll bar arrow that decreases the scroll bar's value.

`kThemeDragSoundScrollBarArrowIncreasing`
: Specifies a sound to be played when the user presses and

holds the mouse button while the cursor is over the scroll bar arrow that increases the scroll bar's value.

`kThemeDragSoundDragging`
Specifies a sound to be played during a Drag Manager drag.

Theme Draw State Constants

You can use constants of type `ThemeDrawState` with many Appearance Manager functions to specify the state in which human interface elements are drawn.

```
enum {
    kThemeStateInactive     = 0,
    kThemeStateActive       = 1,
    kThemeStatePressed      = 2,
    kThemeStatePressedUp    = 2,
    kThemeStatePressedDown  = 3
};
typedef UInt32 ThemeDrawState;
```

Constant descriptions

`kThemeStateInactive`
The element is drawn in the inactive state.

`kThemeStateActive`
The element is drawn in the active state.

`kThemeStatePressed`
The element is drawn in the selected state.

`kThemeStatePressedUp`
For increment/decrement buttons (also known as "little arrows" controls), the increment button is drawn in the selected state. Available with Appearance Manager 1.1 and later.

`kThemeStatePressedDown`
For increment/decrement buttons (also known as "little arrows" controls), the decrement button is drawn in the selected state. Available with Appearance Manager 1.1 and later.

Theme Font ID Constants

Constants of type `ThemeFontID` identify the kinds of system fonts. Your application may pass a `ThemeFontID` constant to the function `UseThemeFont` (page 74) to apply a font of the specified kind to the current port, or it may pass a `ThemeFontID` constant to the function `GetThemeFont` (page 71) to retrieve information about the specified kind of font. The `ThemeFontID` constants are available with Appearance Manager 1.1 and later.

```
enum {
    kThemeSystemFont                = 0,
    kThemeSmallSystemFont           = 1,
    kThemeSmallEmphasizedSystemFont = 2,
    kThemeViewsFont                 = 3
};
typedef UInt16 ThemeFontID;
```

Constant descriptions

`kThemeSystemFont`	The current (large) system font.
`kThemeSmallSystemFont`	The current small system font.
`kThemeSmallEmphasizedSystemFont`	The current small, emphasized system font.
`kThemeViewsFont`	The current views font.

Theme Menu Bar State Constants

You can pass constants of type `ThemeMenuBarState` in the `inState` parameter of `DrawThemeMenuBarBackground` (page 146) to specify whether theme-compliant menu bars are drawn as normal or selected. The `ThemeMenuBarState` constants are available with Appearance Manager 1.0.1 and later.

```
enum{
    kThemeMenuBarNormal        = 0,
    kThemeMenuBarSelected      = 1
};
typedef UInt16 ThemeMenuBarState;
```

Constant descriptions

`kThemeMenuBarNormal`
: Menu bar is drawn in its normal state.

`kThemeMenuBarSelected`
: Menu bar is drawn in its selected state.

If you wish the menu bar to be drawn with square upper corners (as for a laptop system) instead of rounded ones (as for a desktop system), your application should set the bit for the attribute `kThemeMenuSquareMenuBar`.

```
enum {
    kThemeMenuSquareMenuBar= (1 << 0)
};
```

Constant descriptions

`kThemeMenuSquareMenuBar`
: Menu bar is drawn with square corners.

Theme Menu Item Type Constants

Constants of type `ThemeMenuItemType` identify types of menu items. Your application may pass a `ThemeMenuItemType` constant to the function `DrawThemeMenuItem` (page 146) to draw a menu item of the specified type, or it may pass a `ThemeMenuItemType` constant to the function `GetThemeMenuItemExtra` (page 152) to retrieve spatial information for the given menu item type under the current theme.

```
enum {
    kThemeMenuItemPlain            = 0,
    kThemeMenuItemHierarchical     = 1,
    kThemeMenuItemScrollUpArrow    = 2,
    kThemeMenuItemScrollDownArrow  = 3,
    kThemeMenuItemAtTop            = 0x0100,
    kThemeMenuItemAtBottom         = 0x0200,
    kThemeMenuItemHierBackground   = 0x0400,
    kThemeMenuItemPopUpBackground  = 0x0800,
    kThemeMenuItemHasIcon          = 0x8000
};
typedef UInt16 ThemeMenuItemType;
```

Constant descriptions

`kThemeMenuItemPlain`
: A plain menu item. Available with Appearance Manager 1.0.1 and later.

`kThemeMenuItemHierarchical`
: A hierarchical menu item. Available with Appearance Manager 1.0.1 and later.

`kThemeMenuItemScrollUpArrow`
: A scroll-up arrow. Available with Appearance Manager 1.0.1 and later.

`kThemeMenuItemScrollDownArrow`
: A scroll-down arrow. Available with Appearance Manager 1.0.1 and later.

`kThemeMenuItemAtTop`
: This value may be added to other `ThemeMenuItemType` constants to specify that the item being drawn appears at the top of the menu. Available with Appearance Manager 1.1 and later.

`kThemeMenuItemAtBottom`
: This value may be added to other `ThemeMenuItemType` constants to specify that the item being drawn appears at the bottom of the menu. Available with Appearance Manager 1.1 and later.

`kThemeMenuItemHierBackground`
: This value may be added to other `ThemeMenuItemType` constants to specify that the item being drawn is located in a hierarchical menu. Available with Appearance Manager 1.1 and later.

`kThemeMenuItemPopUpBackground`
: This value may be added to other `ThemeMenuItemType` constants to specify that the item being drawn is located in a pop-up menu. Available with Appearance Manager 1.1 and later.

`kThemeMenuItemHasIcon`
: This value may be added to the `kThemeMenuItemPlain` or `kThemeMenuItemHierarchical` constants, to specify that an icon is drawn along with the item text. This value may not be used with the `kThemeMenuItemScrollUpArrow` and

`kThemeMenuItemScrollDownArrow` constants. Available with Appearance Manager 1.1 and later.

Theme Menu State Constants

You can pass constants of type `ThemeMenuState` in the `inState` parameter of `DrawThemeMenuItem` (page 146) and `DrawThemeMenuTitle` (page 149) to specify the state in which theme-compliant menus are drawn. The `ThemeMenuState` constants are available with Appearance Manager 1.0.1 and later.

```
enum{
    kThemeMenuActive            = 0,
    kThemeMenuSelected          = 1,
    kThemeMenuInactive          = 3
};
typedef UInt16 ThemeMenuState;
```

Constant descriptions

`kThemeMenuActive`	Menu is drawn in its active state.
`kThemeMenuSelected`	Menu is drawn in its selected state.
`kThemeMenuInactive`	Menu is drawn in its inactive, disabled state.

Theme Menu Type Constants

You can pass constants of type `ThemeMenuType` in the `inMenuType` parameter of `GetThemeMenuBackgroundRegion` (page 150) and `DrawThemeMenuBackground` (page 145) to specify a type of menu.

```
enum {
    kThemeMenuTypePullDown      = 0,
    kThemeMenuTypePopUp         = 1,
    kThemeMenuTypeHierarchical  = 2,
    kThemeMenuTypeInactive      = 0x0100
};
typedef UInt16 ThemeMenuType;
```

Constant descriptions

`kThemeMenuTypePullDown`	A pull-down menu. Available with Appearance Manager 1.0.1 and later.
`kThemeMenuTypePopUp`	A pop-up menu. Available with Appearance Manager 1.0.1 and later.
`kThemeMenuTypeHierarchical`	A hierarchical menu. Available with Appearance Manager 1.0.1 and later.
`kThemeMenuTypeInactive`	An inactive menu. Add this value to any other menu type if the entire menu is inactive. Available with Appearance Manager 1.1 and later.

Theme Pop-Up Arrow Orientation Constants

You can use a constant of type `ThemeArrowOrientation` in the function `DrawThemePopupArrow` (page 102) to specify the direction in which a pop-up arrow is drawn on a button. The `ThemeArrowOrientation` constants are available with Appearance Manager 1.1 and later.

```
enum {
    kThemeArrowLeft = 0,
    kThemeArrowDown = 1,
    kThemeArrowRight= 2,
    kThemeArrowUp   = 3
};
typedef UInt16 ThemeArrowOrientation;
```

Constant descriptions

`kThemeArrowLeft`	A left-pointing arrow.
`kThemeArrowDown`	A downward-pointing arrow.
`kThemeArrowRight`	A right-pointing arrow.
`kThemeArrowUp`	An upward-pointing arrow.

Theme Pop-Up Arrow Size Constants

You can use a constant of type `ThemePopupArrowSize` in the function `DrawThemePopupArrow` (page 102) to specify the size of the pop-up arrow that is drawn on a button. The `ThemePopupArrowSize` constants are available with Appearance Manager 1.1 and later.

```
enum {
    kThemeArrow3pt  = 0,
    kThemeArrow5pt  = 1,
    kThemeArrow7pt  = 2,
    kThemeArrow9pt  = 3
};
typedef UInt16 ThemePopupArrowSize;
```

Constant descriptions

`kThemeArrow3pt`	Identifies a pop-up arrow with a 3-pixel base.
`kThemeArrow5pt`	Identifies a pop-up arrow with a 5-pixel base.
`kThemeArrow7pt`	Identifies a pop-up arrow with a 7-pixel base.
`kThemeArrow9pt`	Identifies a pop-up arrow with a 9-pixel base.

Theme Scroll Bar Arrow Style Constants

You can call the function `GetThemeScrollBarArrowStyle` (page 116) to obtain the type of scroll bar arrows being used in the current theme. The `ThemeScrollBarArrowStyle` constants are available with Appearance Manager 1.1 and later.

```
enum {
    kThemeScrollBarArrowsSingle     = 0,
    kThemeScrollBarArrowsLowerRight = 1
};
typedef UInt16 ThemeScrollBarArrowStyle;
```

Constant descriptions

`kThemeScrollBarArrowsSingle`
: Specifies the use of a single arrow at each end of a scroll bar.

`kThemeScrollBarArrowsLowerRight`
: Specifies the use of double arrows at one end of a scroll bar. For vertical scroll bars, the double arrows are located at the lower end of the scroll bar. For horizontal scroll bars, the double arrows are located at the right end of the scroll bar.

Theme Scroll Box Style Constants

You can call the function `GetThemeScrollBarThumbStyle` (page 117) to obtain the type of scroll boxes (also known as "scroll indicators" or "thumbs") being used in the current theme. The `ThemeScrollBarThumbStyle` constants are available with Appearance Manager 1.1 and later.

```
enum {
    kThemeScrollBarThumbNormal         = 0,
    kThemeScrollBarThumbProportional   = 1
};
typedef UInt16 ThemeScrollBarThumbStyle;
```

Constant descriptions

`kThemeScrollBarThumbNormal`
: A classic scroll box.

`kThemeScrollBarThumbProportional`
: A proportional scroll box.

Theme Size Box Direction Constants

The `ThemeGrowDirection` enumeration defines masks your application can use to specify the directions in which a window may be resized. You may use constants of type `ThemeGrowDirection` with the function `DrawThemeStandaloneGrowBox` (page 131) to draw a size box and with the function `GetThemeStandaloneGrowBoxBounds` (page 138) to obtain the bounding rectangle of a size box. The constants may be combined to set more than one direction of growth. The `ThemeGrowDirection` constants are available with Appearance Manager 1.1 and later.

```
enum {
    kThemeGrowLeft  = (1 << 0),
    kThemeGrowRight = (1 << 1),
    kThemeGrowUp    = (1 << 2),
    kThemeGrowDown  = (1 << 3)
};
typedef UInt16 ThemeGrowDirection;
```

Constant descriptions

`kThemeGrowLeft`	If the bit specified by this mask is set, the object can grow to the left.
`kThemeGrowRight`	If the bit specified by this mask is set, the object can grow to the right.
`kThemeGrowUp`	If the bit specified by this mask is set, the object can grow upward. Note: This functionality is not available with Appearance Manager 1.1 or prior versions of Appearance.
`kThemeGrowDown`	If the bit specified by this mask is set, the object can grow downward.

Theme Slider Indicator Direction Constants

You can use constants of type `ThemeThumbDirection` in the `SliderTrackInfo` (page 168) structure to identify the direction in which the indicator points in a slider control. You may use these constants with either horizontal or vertical sliders, and the Appearance Manager interprets the direction of the indicator appropriately. The `ThemeThumbDirection` constants are available with Appearance Manager 1.1 and later.

```
enum {
    kThemeThumbPlain    = 0,
    kThemeThumbUpward   = 1,
    kThemeThumbDownward = 2
};
typedef UInt8 ThemeThumbDirection;
```

Constant descriptions

`kThemeThumbPlain`	A plain indicator; that is, one that does not point in any direction.
`kThemeThumbUpward`	For a horizontal slider, an upward-pointing indicator. For a vertical slider, a left-pointing indicator.

`kThemeThumbDownward`
For a horizontal slider, a downward-pointing indicator. For a vertical slider, a right-pointing indicator.

Theme Sound Constants

Your application can pass constants of type `ThemeSoundKind` to the function `PlayThemeSound` (page 89) to play a theme-specific sound for an interface object when it changes state. Each sound plays asynchronously until complete, then stops automatically. The `ThemeSoundKind` constants are available with Appearance Manager 1.1 and later.

```
enum {
    kThemeSoundNone                    = 0,
    /* menu sounds */
    kThemeSoundMenuOpen                = 'mnuo',
    kThemeSoundMenuClose               = 'mnuc',
    kThemeSoundMenuItemHilite          = 'mnui',
    kThemeSoundMenuItemRelease         = ‘mnus’,
    /* window sounds */
    kThemeSoundWindowClosePress        = 'wclp',
    kThemeSoundWindowCloseEnter        = 'wcle',
    kThemeSoundWindowCloseExit         = 'wclx',
    kThemeSoundWindowCloseRelease      = 'wclr',
    kThemeSoundWindowZoomPress         = 'wzmp',
    kThemeSoundWindowZoomEnter         = 'wzme',
    kThemeSoundWindowZoomExit          = 'wzmx',
    kThemeSoundWindowZoomRelease       = 'wzmr',
    kThemeSoundWindowCollapsePress     = 'wcop',
    kThemeSoundWindowCollapseEnter     = 'wcoe',
    kThemeSoundWindowCollapseExit      = 'wcox',
    kThemeSoundWindowCollapseRelease   = 'wcor',
    kThemeSoundWindowDragBoundary      = 'wdbd',
    kThemeSoundUtilWinClosePress       = 'uclp',
    kThemeSoundUtilWinCloseEnter       = 'ucle',
    kThemeSoundUtilWinCloseExit        = 'uclx',
    kThemeSoundUtilWinCloseRelease     = 'uclr',
    kThemeSoundUtilWinZoomPress        = 'uzmp',
    kThemeSoundUtilWinZoomEnter        = 'uzme',
    kThemeSoundUtilWinZoomExit         = 'uzmx',
    kThemeSoundUtilWinZoomRelease       = 'uzmr',
    kThemeSoundUtilWinCollapsePress     = 'ucop',
    kThemeSoundUtilWinCollapseEnter     = 'ucoe',
    kThemeSoundUtilWinCollapseExit      = 'ucox',
    kThemeSoundUtilWinCollapseRelease   = 'ucor',
    kThemeSoundUtilWinDragBoundary      = 'udbd',
    kThemeSoundWindowOpen               = ‘wopn’,
    kThemeSoundWindowClose              = 'wcls',
    kThemeSoundWindowZoomIn             = 'wzmi',
    kThemeSoundWindowZoomOut            = 'wzmo',
    kThemeSoundWindowCollapseUp         = 'wcol',
    kThemeSoundWindowCollapseDown       = 'wexp',
    kThemeSoundWindowActivate           = 'wact',
    kThemeSoundUtilWindowOpen           = ‘uopn’,
    kThemeSoundUtilWindowClose          = 'ucls',
    kThemeSoundUtilWindowZoomIn         = 'uzmi',
    kThemeSoundUtilWindowZoomOut        = 'uzmo',
    kThemeSoundUtilWindowCollapseUp     = 'ucol',
    kThemeSoundUtilWindowCollapseDown   = 'uexp',
    kThemeSoundUtilWindowActivate       = 'uact',
    kThemeSoundDialogOpen               = ‘dopn’,
    kThemeSoundDialogClose              = 'dlgc',
    kThemeSoundAlertOpen                = ‘aopn’,
    kThemeSoundAlertClose               = 'altc',
    kThemeSoundPopopWindowOpen          = 'pwop',
    kThemeSoundPopupWindowClose         = 'pwcl',
    /* push button sounds */
    kThemeSoundButtonPress              = 'btnp',
    kThemeSoundButtonEnter              = 'btne',
    kThemeSoundButtonExit               = 'btnx',
    kThemeSoundButtonRelease            = 'btnr',
    kThemeSoundDefaultButtonPress       = 'dbtp',
    kThemeSoundDefaultButtonEnter       = 'dbte',
    kThemeSoundDefaultButtonExit        = 'dbtx',
    kThemeSoundDefaultButtonRelease     = 'dbtr',
    kThemeSoundCancelButtonPress        = 'cbtp',
    kThemeSoundCancelButtonEnter        = 'cbte',
    kThemeSoundCancelButtonExit         = 'cbtx',
    kThemeSoundCancelButtonRelease      = 'cbtr',
    /* checkbox sounds */
    kThemeSoundCheckboxPress            = 'chkp',
    kThemeSoundCheckboxEnter            = 'chke',
    kThemeSoundCheckboxExit             = 'chkx',
    kThemeSoundCheckboxRelease          = 'chkr',
    /* radio button sounds */
    kThemeSoundRadioPress               = 'radp',
    kThemeSoundRadioEnter               = 'rade',
    kThemeSoundRadioExit                = 'radx',
    kThemeSoundRadioRelease             = 'radr',
    /* scroll bar sounds */
    kThemeSoundScrollArrowPress         = 'sbap',
    kThemeSoundScrollArrowEnter         = 'sbae',
    kThemeSoundScrollArrowExit          = 'sbax',
    kThemeSoundScrollArrowRelease       = 'sbar',
    kThemeSoundScrollEndOfTrack         = 'sbte',
    kThemeSoundScrollTrackPress         = 'sbtp',
    /* slider sounds */
    kThemeSoundSliderEndOfTrack         = 'slte',
    kThemeSoundSliderTrackPress         = 'sltp',
    /* help balloon sounds */
    kThemeSoundBalloonOpen              = 'blno',
    kThemeSoundBalloonClose             = 'blnc',
    /* bevel button sounds */
    kThemeSoundBevelPress               = 'bevp',
    kThemeSoundBevelEnter               = 'beve',
    kThemeSoundBevelExit                = 'bevx',
    kThemeSoundBevelRelease             = 'bevr',
    /* increment/decrement button sounds */
    kThemeSoundLittleArrowUpPress       = 'laup',
    kThemeSoundLittleArrowDnPress       = 'ladp',
    kThemeSoundLittleArrowEnter         = 'lare',
    kThemeSoundLittleArrowExit          = 'larx',
    kThemeSoundLittleArrowUpRelease     = 'laur',
    kThemeSoundLittleArrowDnRelease     = 'ladr',
    /* pop-up button sounds */
    kThemeSoundPopupPress               = 'popp',
    kThemeSoundPopupEnter               = 'pope',
    kThemeSoundPopupExit                = 'popx',
    kThemeSoundPopupRelease             = 'popr',
    /* disclosure triangle sounds */
    kThemeSoundDisclosurePress          = 'dscp',
    kThemeSoundDisclosureEnter          = 'dsce',
    kThemeSoundDisclosureExit           = 'dscx',
    kThemeSoundDisclosureRelease        = 'dscr',
    /* tabs sounds */
    kThemeSoundTabPressed               = 'tabp',
    kThemeSoundTabEnter                 = 'tabe',
    kThemeSoundTabExit                  = 'tabx',
    kThemeSoundTabRelease               = 'tabr'
    /* Drag Manager sounds */
    kThemeSoundDragTargetHilite         = 'dthi',
    kThemeSoundDragTargetUnhilite       = 'dtuh',
    kThemeSoundDragTargetDrop           = 'dtdr',
    /* Finder sounds */
    kThemeSoundEmptyTrash               = 'ftrs',
    kThemeSoundSelectItem               = 'fsel',
    kThemeSoundNewItem                  = 'fnew',
    kThemeSoundReceiveDrop              = 'fdrp',
    kThemeSoundCopyDone                 = 'fcpd',
    kThemeSoundResolveAlias             = 'fral',
    kThemeSoundLaunchApp                = 'flap',
    kThemeSoundDiskInsert               = 'dski',
    kThemeSoundDiskEject                = 'dske',
    kThemeSoundFinderDragOnIcon         = 'fdon',
    kThemeSoundFinderDragOffIcon        = 'fdof'
};
typedef OSType ThemeSoundKind;
```

Constant descriptions

`kThemeSoundNone`	Specifies that no sound is played.
`kThemeSoundMenuOpen`	Identifies a sound to be played when the user opens a menu.
`kThemeSoundMenuClose`	Identifies a sound to be played when the user closes a menu.
`kThemeSoundMenuItemHilite`	Identifies a sound to be played when the user highlights a menu item.
`kThemeSoundMenuItemRelease`	Identifies a sound to be played when the user selects a menu item.

`kThemeSoundWindowClosePress`
: Identifies a sound to be played when the user presses the mouse button while the cursor is over a window's close box.

`kThemeSoundWindowCloseEnter`
: Identifies a sound to be played when the user moves the cursor over a window's close box after having moved the cursor away from the close box without releasing the mouse button.

`kThemeSoundWindowCloseExit`
: Identifies a sound to be played when the user moves the cursor away from a position over a window's close box, while the mouse button remains pressed.

`kThemeSoundWindowCloseRelease`
: Identifies a sound to be played when the user releases the mouse button while the cursor is over a window's close box.

`kThemeSoundWindowZoomPress`
: Identifies a sound to be played when the user presses the mouse button while the cursor is over a window's zoom box.

`kThemeSoundWindowZoomEnter`
: Identifies a sound to be played when the user moves the cursor over a window's zoom box after having moved the cursor away from the zoom box without releasing the mouse button.

`kThemeSoundWindowZoomExit`
: Identifies a sound to be played when the user moves the cursor away from a position over a window's zoom box, while the mouse button remains pressed.

`kThemeSoundWindowZoomRelease`
: Identifies a sound to be played when the user releases the mouse button while the cursor is over a window's zoom box.

`kThemeSoundWindowCollapsePress`
: Identifies a sound to be played when the user presses the mouse button while the cursor is over a window's collapse box.

`kThemeSoundWindowCollapseEnter`
Identifies a sound to be played when the user moves the cursor over a window's collapse box after having moved the cursor away from the collapse box without releasing the mouse button.

`kThemeSoundWindowCollapseExit`
Identifies a sound to be played when the user moves the cursor away from a position over a window's collapse box, while the mouse button remains pressed.

`kThemeSoundWindowCollapseRelease`
Identifies a sound to be played when the user releases the mouse button while the cursor is over a window's collapse box.

`kThemeSoundWindowDragBoundary`
Identifies a sound to be played when the user drags a window to the edge of the area where it can be dragged. Note: This functionality is not available under Appearance Manager 1.1 or prior versions of Appearance.

`kThemeSoundUtilWinClosePress`
Identifies a sound to be played when the user presses the mouse button while the cursor is over a utility (floating) window's close box.

`kThemeSoundUtilWinCloseEnter`
Identifies a sound to be played when the user moves the cursor over a utility (floating) window's close box after having moved the cursor away from the close box without releasing the mouse button.

`kThemeSoundUtilWinCloseExit`
Identifies a sound to be played when the user moves the cursor away from a position over a utility (floating) window's close box, while the mouse button remains pressed.

`kThemeSoundUtilWinCloseRelease`
Identifies a sound to be played when the user releases the mouse button while the cursor is over a utility (floating) window's close box.

`kThemeSoundUtilWinZoomPress`
Identifies a sound to be played when the user presses the

mouse button while the cursor is over a utility (floating) window's zoom box.

`kThemeSoundUtilWinZoomEnter`
: Identifies a sound to be played when the user moves the cursor over a utility (floating) window's zoom box after having moved the cursor away from the zoom box without releasing the mouse button.

`kThemeSoundUtilWinZoomExit`
: Identifies a sound to be played when the user moves the cursor away from a position over a utility (floating) window's zoom box, while the mouse button remains pressed.

`kThemeSoundUtilWinZoomRelease`
: Identifies a sound to be played when the user releases the mouse button while the cursor is over a utility (floating) window's zoom box.

`kThemeSoundUtilWinCollapsePress`
: Identifies a sound to be played when the user presses the mouse button while the cursor is over a utility (floating) window's collapse box.

`kThemeSoundUtilWinCollapseEnter`
: Identifies a sound to be played when the user moves the cursor over a utility (floating) window's collapse box after having moved the cursor away from the collapse box without releasing the mouse button.

`kThemeSoundUtilWinCollapseExit`
: Identifies a sound to be played when the user moves the cursor away from a position over a utility (floating) window's collapse box, while the mouse button remains pressed.

`kThemeSoundUtilWinCollapseRelease`
: Identifies a sound to be played when the user releases the mouse button while the cursor is over a utility (floating) window's collapse box.

`kThemeSoundUtilWinDragBoundary`
: Identifies a sound to be played when the user drags a utility (floating) window to the edge of the area where it can be dragged. Note: This functionality is not available

under Appearance Manager 1.1 or prior versions of Appearance.

`kThemeSoundWindowOpen`
: Identifies a sound to be played when the user opens a window.

`kThemeSoundWindowClose`
: Identifies a sound to be played when the user closes a window.

`kThemeSoundWindowZoomIn`
: Identifies a sound to be played when the user zooms a window in, that is, to the user state.

`kThemeSoundWindowZoomOut`
: Identifies a sound to be played when the user zooms a window out, that is, to the standard state.

`kThemeSoundWindowCollapseUp`
: Identifies a sound to be played when the user collapses a window.

`kThemeSoundWindowCollapseDown`
: Identifies a sound to be played when the user uncollapses a window.

`kThemeSoundWindowActivate`
: Identifies a sound to be played when the user presses the mouse button while the cursor is over an inactive window, thus activating it.

`kThemeSoundUtilWindowOpen`
: Identifies a sound to be played when the user opens a utility (floating) window.

`kThemeSoundUtilWindowClose`
: Identifies a sound to be played when the user closes a utility (floating) window.

`kThemeSoundUtilWindowZoomIn`
: Identifies a sound to be played when the user zooms a utility (floating) window in, that is, to the user state.

`kThemeSoundUtilWindowZoomOut`
: Identifies a sound to be played when the user zooms a utility (floating) window out, that is, to the standard state.

`kThemeSoundUtilWindowCollapseUp`
: Identifies a sound to be played when the user collapses a utility (floating) window.

`kThemeSoundUtilWindowCollapseDown`
: Identifies a sound to be played when the user uncollapses a utility (floating) window.

`kThemeSoundUtilWindowActivate`
: Identifies a sound to be played when the user presses the mouse button while the cursor is over an inactive utility (floating) window, thus activating it.

`kThemeSoundDialogOpen`
: Identifies a sound to be played when a dialog box opens.

`kThemeSoundDialogClose`
: Identifies a sound to be played when a dialog box closes.

`kThemeSoundAlertOpen`
: Identifies a sound to be played when an alert box opens.

`kThemeSoundAlertClose`
: Identifies a sound to be played when an alert box closes.

`kThemeSoundPopopWindowOpen`
: Identifies a sound to be played when a pop-up window opens.

`kThemeSoundPopupWindowClose`
: Identifies a sound to be played when a pop-up window closes.

`kThemeSoundButtonPress`
: Identifies a sound to be played when the user presses the mouse button while the cursor is over a push button.

`kThemeSoundButtonEnter`
: Identifies a sound to be played when the user moves the cursor over a push button after having moved the cursor away from the button without releasing the mouse button.

`kThemeSoundButtonExit`
: Identifies a sound to be played when the user moves the cursor away from a position over a push button, while the mouse button remains pressed.

`kThemeSoundButtonRelease`
: Identifies a sound to be played when the user releases the mouse button while the cursor is over a push button.

`kThemeSoundDefaultButtonPress`
: Identifies a sound to be played when the user presses the mouse button while the cursor is over a default button.

`kThemeSoundDefaultButtonEnter`
: Identifies a sound to be played when the user moves the cursor over a default button after having moved the cursor away from the button without releasing the mouse button.

`kThemeSoundDefaultButtonExit`
: Identifies a sound to be played when the user moves the cursor away from a position over a default button, while the mouse button remains pressed.

`kThemeSoundDefaultButtonRelease`
: Identifies a sound to be played when the user releases the mouse button while the cursor is over a default button.

`kThemeSoundCancelButtonPress`
: Identifies a sound to be played when the user presses the mouse button while the cursor is over a Cancel button.

`kThemeSoundCancelButtonEnter`
: Identifies a sound to be played when the user moves the cursor over a Cancel button after having moved the cursor away from the button without releasing the mouse button.

`kThemeSoundCancelButtonExit`
: Identifies a sound to be played when the user moves the cursor away from a position over a Cancel button, while the mouse button remains pressed.

`kThemeSoundCancelButtonRelease`
: Identifies a sound to be played when the user releases the mouse button while the cursor is over a Cancel button.

`kThemeSoundCheckboxPress`
: Identifies a sound to be played when the user presses the mouse button while the cursor is over a checkbox.

`kThemeSoundCheckboxEnter`
: Identifies a sound to be played when the user moves the cursor over a checkbox after having moved the cursor away from the checkbox without releasing the mouse button.

`kThemeSoundCheckboxExit`
: Identifies a sound to be played when the user moves the

cursor away from a position over a checkbox, while the mouse button remains pressed.

`kThemeSoundCheckboxRelease`
: Identifies a sound to be played when the user releases the mouse button while the cursor is over a checkbox.

`kThemeSoundRadioPress`
: Identifies a sound to be played when the user presses the mouse button while the cursor is over a radio button.

`kThemeSoundRadioEnter`
: Identifies a sound to be played when the user moves the cursor over a radio button after having moved the cursor away from the radio button without releasing the mouse button.

`kThemeSoundRadioExit`
: Identifies a sound to be played when the user moves the cursor away from a position over a radio button, while the mouse button remains pressed.

`kThemeSoundRadioRelease`
: Identifies a sound to be played when the user releases the mouse button while the cursor is over a radio button.

`kThemeSoundScrollArrowPress`
: Identifies a sound to be played when the user presses the mouse button while the cursor is over a scroll bar arrow.

`kThemeSoundScrollArrowEnter`
: Identifies a sound to be played when the user moves the cursor over a scroll bar arrow after having moved the cursor away from the arrow without releasing the mouse button.

`kThemeSoundScrollArrowExit`
: Identifies a sound to be played when the user moves the cursor away from a position over a scroll bar arrow, while the mouse button remains pressed.

`kThemeSoundScrollArrowRelease`
: Identifies a sound to be played when the user releases the mouse button while the cursor is over a scroll bar arrow.

`kThemeSoundScrollEndOfTrack`
: Identifies a sound to be played when a scroll box arrives at the end of a scroll bar and can go no further. Note: This

functionality is not available under Appearance Manager 1.1 or prior versions of Appearance.

`kThemeSoundScrollTrackPress`
: Identifies a sound to be played when the user presses the mouse button while the cursor is over the track part of a scroll bar (this area does not include the scroll box or scroll bar arrows).

`kThemeSoundSliderEndOfTrack`
: Identifies a sound to be played when a slider indicator arrives at the end of a slider track and can go no further. Note: This functionality is not available under Appearance Manager 1.1 or prior versions of Appearance.

`kThemeSoundSliderTrackPress`
: Identifies a sound to be played when the user presses the mouse button while the cursor is over the track part of a slider (this area does not include the slider indicator).

`kThemeSoundBalloonOpen`
: Identifies a sound to be played when a help balloon appears.

`kThemeSoundBalloonClose`
: Identifies a sound to be played when a help balloon disappears.

`kThemeSoundBevelPress`
: Identifies a sound to be played when the user presses the mouse button while the cursor is over a bevel button.

`kThemeSoundBevelEnter`
: Identifies a sound to be played when the user moves the cursor over a bevel button after having moved the cursor away from the bevel button without releasing the mouse button.

`kThemeSoundBevelExit`
: Identifies a sound to be played when the user moves the cursor away from a position over a bevel button, while the mouse button remains pressed.

`kThemeSoundBevelRelease`
: Identifies a sound to be played when the user releases the mouse button while the cursor is over a bevel button.

`kThemeSoundLittleArrowUpPress`
: Identifies a sound to be played when the user presses the mouse button while the cursor is over the upward-pointing arrow of an increment/decrement button.

`kThemeSoundLittleArrowDnPress`
: Identifies a sound to be played when the user presses the mouse button while the cursor is over the downward-pointing arrow of an increment/decrement button.

`kThemeSoundLittleArrowEnter`
: Identifies a sound to be played when the user moves the cursor over an increment/decrement button after having moved the cursor away from the button without releasing the mouse button.

`kThemeSoundLittleArrowExit`
: Identifies a sound to be played when the user moves the cursor away from a position over an increment/decrement button, while the mouse button remains pressed.

`kThemeSoundLittleArrowUpRelease`
: Identifies a sound to be played when the user releases the mouse button while the cursor is over the upward-pointing arrow of an increment/decrement button.

`kThemeSoundLittleArrowDnRelease`
: Identifies a sound to be played when the user releases the mouse button while the cursor is over the downward-pointing arrow of an increment/decrement button.

`kThemeSoundPopupPress`
: Identifies a sound to be played when the user presses the mouse button while the cursor is over a pop-up menu button.

`kThemeSoundPopupEnter`
: Identifies a sound to be played when the user moves the cursor over a pop-up menu button after having moved the cursor away from the button without releasing the mouse button.

`kThemeSoundPopupExit`
: Identifies a sound to be played when the user moves the

cursor away from a position over a pop-up menu button, while the mouse button remains pressed.

`kThemeSoundPopupRelease`
: Identifies a sound to be played when the user releases the mouse button while the cursor is over a pop-up menu button.

`kThemeSoundDisclosurePress`
: Identifies a sound to be played when the user presses the mouse button while the cursor is over a disclosure triangle.

`kThemeSoundDisclosureEnter`
: Identifies a sound to be played when the user moves the cursor over a disclosure triangle after having moved the cursor away from the disclosure triangle without releasing the mouse button.

`kThemeSoundDisclosureExit`
: Identifies a sound to be played when the user moves the cursor away from a position over a disclosure triangle, while the mouse button remains pressed.

`kThemeSoundDisclosureRelease`
: Identifies a sound to be played when the user releases the mouse button while the cursor is over a disclosure triangle.

`kThemeSoundTabPressed`
: Identifies a sound to be played when the user presses the mouse button while the cursor is over a tab.

`kThemeSoundTabEnter`
: Identifies a sound to be played when the user places the cursor over a tab.

`kThemeSoundTabExit`
: Identifies a sound to be played when the user moves the cursor over a tab after having moved the cursor away from the tab without releasing the mouse button.

`kThemeSoundTabRelease`
: Identifies a sound to be played when the user releases the mouse button while the cursor is over a tab.

`kThemeSoundDragTargetHilite`
: Identifies a sound to be played when the user drags an object over a valid drag-and-drop destination.

`kThemeSoundDragTargetUnhilite`
: Identifies a sound to be played when the user drags an object away from a valid drag-and-drop destination.

`kThemeSoundDragTargetDrop`
: Identifies a sound to be played when the user drops an object on a valid drag-and-drop destination.

`kThemeSoundEmptyTrash`
: Identifies a sound to be played when the Finder completes emptying the Trash directory.

`kThemeSoundSelectItem`
: Identifies a sound to be played when the user presses the mouse button while the cursor is over an item in the Finder.

`kThemeSoundNewItem`
: Identifies a sound to be played when the user creates a new item.

`kThemeSoundReceiveDrop`
: Identifies a sound to be played when a Finder object changes parents, such as when the user drops an icon on a folder.

`kThemeSoundCopyDone`
: Identifies a sound to be played when the Finder completes a copy operation.

`kThemeSoundResolveAlias`
: Identifies a sound to be played when the Finder resolves an alias.

`kThemeSoundLaunchApp`
: Identifies a sound to be played when the Finder launches an application.

`kThemeSoundDiskInsert`
: Identifies a sound to be played when a disk is inserted.

`kThemeSoundDiskEject`
: Identifies a sound to be played when a disk is ejected.

`kThemeSoundFinderDragOnIcon`
: Identifies a sound to be played when the user drags an object over an icon.

`kThemeSoundFinderDragOffIcon`
: Identifies a sound to be played when the user drags an object off of an icon.

Theme Sound Mask Constants

Theme sound mask constants define masks that are used to specify the classes of sounds that are enabled for a theme. You can use these masks to operate upon the unsigned 32-bit integer contained in the `kThemeSoundMaskTag` collection item, which is described in "Theme Collection Tags" (page 190). Theme sound mask constants are available with Appearance Manager 1.1 and later.

```
enum {
    kThemeNoSounds          = 0,
    kThemeWindowSoundsMask  = (1 << 0),
    kThemeMenuSoundsMask    = (1 << 1),
    kThemeControlSoundsMask = (1 << 2),
    kThemeFinderSoundsMask  = (1 << 3)
};
```

Constant descriptions

`kThemeNoSounds`	If no bits are set, no theme sounds are enabled.
`kThemeWindowSoundsMask`	If the bit specified by this mask is set, window sounds are enabled.
`kThemeMenuSoundsMask`	If the bit specified by this mask is set, menu sounds are enabled.
`kThemeControlSoundsMask`	If the bit specified by this mask is set, control sounds are enabled.
`kThemeFinderSoundsMask`	If the bit specified by this mask is set, Finder sounds are enabled.

Theme Tab Direction Constants

You can pass constants of type `ThemeTabDirection` to the function `DrawThemeTab` (page 107) to draw theme-compliant tabs that are oriented in various directions. You can also pass a `ThemeTabDirection` constant to the function `GetThemeTabRegion` (page 119) to obtain the region containing a tab oriented in a particular direction. The `ThemeTabDirection` constants are available with Appearance Manager 1.1 and later.

```
enum {
    kThemeTabNorth  = 0,
    kThemeTabSouth  = 1,
    kThemeTabEast   = 2,
    kThemeTabWest   = 3
};
typedef UInt16 ThemeTabDirection;
```

Constant descriptions

`kThemeTabNorth`	An upward-pointing tab.
`kThemeTabSouth`	A downward-pointing tab.
`kThemeTabEast`	A right-pointing tab.
`kThemeTabWest`	A left-pointing tab.

Theme Tab Style Constants

You can pass a constant of type `ThemeTabStyle` to the function `DrawThemeTab` (page 107) to draw a theme-compliant tab in a specific state. You can also pass a `ThemeTabStyle` constant to the function `GetThemeTabRegion` (page 119) to obtain the region containing a tab in a specific state. The `ThemeTabStyle` constants are available with Appearance Manager 1.1 and later.

```
enum {
    kThemeTabNonFront            = 0,
    kThemeTabNonFrontPressed     = 1,
    kThemeTabNonFrontInactive    = 2,
    kThemeTabFront               = 3,
    kThemeTabFrontInactive       = 4
};
typedef UInt16 ThemeTabStyle;
```

Constant descriptions

`kThemeTabNonFront`	An active tab that is not the frontmost in a tab control.
`kThemeTabNonFrontPressed`	A tab that is being clicked on by the user which is not the frontmost tab in a tab control.
`kThemeTabNonFrontInactive`	An inactive tab that is not the frontmost in a tab control.

	The tab may either be inactive because it has been individually disabled or because the tab control as a whole is currently inactive.
`kThemeTabFront`	The frontmost tab in an active tab control.
`kThemeTabFrontInactive`	The frontmost tab in an inactive tab control.

Theme Text Color Constants

A constant of type `ThemeTextColor` identifies a particular context in which text is used. You can pass a constant of type `ThemeTextColor` to the function `SetThemeTextColor` (page 86) to specify that the Appearance Manager substitute whatever the appropriate text color is for a given context under the current theme. You can use the function `GetThemeTextColor` (page 81) to obtain the actual color in use under the current theme for the specified `ThemeTextColor` constant.

```
enum {
    kThemeTextColorDialogActive                     = 1,
    kThemeTextColorDialogInactive                   = 2,
    kThemeTextColorAlertActive                      = 3,
    kThemeTextColorAlertInactive                    = 4,
    kThemeTextColorModelessDialogActive             = 5,
    kThemeTextColorModelessDialogInactive           = 6,
    kThemeTextColorWindowHeaderActive               = 7,
    kThemeTextColorWindowHeaderInactive             = 8,
    kThemeTextColorPlacardActive                    = 9,
    kThemeTextColorPlacardInactive                  = 10,
    kThemeTextColorPlacardPressed                   = 11,
    kThemeTextColorPushButtonActive                 = 12,
    kThemeTextColorPushButtonInactive               = 13,
    kThemeTextColorPushButtonPressed                = 14,
    kThemeTextColorBevelButtonActive                = 15,
    kThemeTextColorBevelButtonInactive              = 16,
    kThemeTextColorBevelButtonPressed               = 17,
    kThemeTextColorPopupButtonActive                = 18,
    kThemeTextColorPopupButtonInactive              = 19,
    kThemeTextColorPopupButtonPressed               = 20,
    kThemeTextColorIconLabel                        = 21,
    kThemeTextColorListView                         = 22,
    /* Text colors available in Appearance Manager 1.0.1 or later */
    kThemeTextColorDocumentWindowTitleActive        = 23,
    kThemeTextColorDocumentWindowTitleInactive      = 24,
    kThemeTextColorMovableModalWindowTitleActive    = 25,
    kThemeTextColorMovableModalWindowTitleInactive  = 26,
    kThemeTextColorUtilityWindowTitleActive         = 27,
    kThemeTextColorUtilityWindowTitleInactive       = 28,
    kThemeTextColorPopupWindowTitleActive           = 29,
    kThemeTextColorPopupWindowTitleInactive         = 30,
    kThemeTextColorRootMenuActive                   = 31,
    kThemeTextColorRootMenuSelected                 = 32,
    kThemeTextColorRootMenuDisabled                 = 33,
    kThemeTextColorMenuItemActive                   = 34,
    kThemeTextColorMenuItemSelected                 = 35,
    kThemeTextColorMenuItemDisabled                 = 36,
    kThemeTextColorPopupLabelActive                 = 37,
    kThemeTextColorPopupLabelInactive               = 38,
    /* Text colors available in Appearance Manager 1.1 or later */
    kThemeTextColorTabFrontActive                   = 39,
    kThemeTextColorTabNonFrontActive                = 40,
    kThemeTextColorTabNonFrontPressed               = 41,
    kThemeTextColorTabFrontInactive                 = 42,
    kThemeTextColorTabNonFrontInactive              = 43,
    kThemeTextColorIconLabelSelected                = 44,
    kThemeTextColorBevelButtonStickyActive          = 45,
    kThemeTextColorBevelButtonStickyInactive        = 46,
    kThemeTextColorBlack                            = -1,
    kThemeTextColorWhite                            = -2
};
typedef SInt16 ThemeTextColor;
```

Constant descriptions

`kThemeTextColorDialogActive`
: Text color for an active dialog box.

`kThemeTextColorDialogInactive`
: Text color for an inactive dialog box.

`kThemeTextColorAlertActive`
: Text color for an active alert box.

`kThemeTextColorAlertInactive`
: Text color for an inactive alert box.

`kThemeTextColorModelessDialogActive`
: Text color for an active modeless dialog box.

`kThemeTextColorModelessDialogInactive`
: Text color for an inactive modeless dialog box.

`kThemeTextColorWindowHeaderActive`
: Text color for the window header of an active window.

`kThemeTextColorWindowHeaderInactive`
: Text color for the window header of an inactive window.

`kThemeTextColorPlacardActive`
: Text color for a placard in an active window.

`kThemeTextColorPlacardInactive`
: Text color for a placard in an inactive window.

`kThemeTextColorPlacardPressed`
: Text color for a placard that is being clicked on by the user.

`kThemeTextColorPushButtonActive`
: Text color for an active push button.

`kThemeTextColorPushButtonInactive`
: Text color for an inactive push button.

`kThemeTextColorPushButtonPressed`
: Text color for a push button that is being clicked on by the user.

`kThemeTextColorBevelButtonActive`
: Text color for an active bevel button.

`kThemeTextColorBevelButtonInactive`
: Text color for an inactive bevel button.

`kThemeTextColorBevelButtonPressed`
: Text color for a bevel button that is being clicked on by the user.

`kThemeTextColorPopupButtonActive`
: Text color for the menu of an active pop-up menu button.

`kThemeTextColorPopupButtonInactive`
: Text color for the menu of an inactive pop-up menu button.

`kThemeTextColorPopupButtonPressed`
: Text color for the menu of a pop-up menu button that is being clicked on by the user.

`kThemeTextColorIconLabel`
: Text color for an icon label.

`kThemeTextColorListView`
: Text color for the contents of a list view column.

`kThemeTextColorDocumentWindowTitleActive`
: Text color for the title of an active document window. Available with Appearance Manager 1.0.1 and later.

`kThemeTextColorDocumentWindowTitleInactive`
: Text color for the title of an inactive document window. Available with Appearance Manager 1.0.1 and later.

`kThemeTextColorMovableModalWindowTitleActive`
: Text color for the title of an active movable modal window. Available with Appearance Manager 1.0.1 and later.

`kThemeTextColorMovableModalWindowTitleInactive`
: Text color for the title of inactive movable modal window. Available with Appearance Manager 1.0.1 and later.

`kThemeTextColorUtilityWindowTitleActive`
: Text color for the title of an active utility (floating) window. Available with Appearance Manager 1.0.1 and later.

`kThemeTextColorUtilityWindowTitleInactive`
: Text color for the title of an inactive utility (floating) window. Available with Appearance Manager 1.0.1 and later.

`kThemeTextColorPopupWindowTitleActive`
: Text color for the title of an active pop-up window. Available with Appearance Manager 1.0.1 and later.

`kThemeTextColorPopupWindowTitleInactive`
: Text color for the title of an inactive pop-up window. Available with Appearance Manager 1.0.1 and later.

`kThemeTextColorRootMenuActive`
: Text color for an active menu bar title. Available with Appearance Manager 1.0.1 and later.

`kThemeTextColorRootMenuSelected`
: Text color for a menu bar title that is being selected by the user. Available with Appearance Manager 1.0.1 and later.

`kThemeTextColorRootMenuDisabled`
: Text color for a disabled menu bar title. Available with Appearance Manager 1.0.1 and later.

`kThemeTextColorMenuItemActive`
: Text color for an active menu item. Available with Appearance Manager 1.0.1 and later.

`kThemeTextColorMenuItemSelected`
: Text color for a menu item that is being selected by the user. Available with Appearance Manager 1.0.1 and later.

`kThemeTextColorMenuItemDisabled`
: Text color for a disabled menu item. Available with Appearance Manager 1.0.1 and later.

`kThemeTextColorPopupLabelActive`
: Text color for the label of an active pop-up menu button. Available with Appearance Manager 1.0.1 and later.

`kThemeTextColorPopupLabelInactive`
: Text color for the label of an inactive pop-up menu button. Available with Appearance Manager 1.0.1 and later.

`kThemeTextColorTabFrontActive`
: Text color for the front tab of an active tab control. Available with Appearance Manager 1.1 and later.

`kThemeTextColorTabNonFrontActive`
: Text color for an active tab that is not the frontmost of a tab control. Available with Appearance Manager 1.1 and later.

`kThemeTextColorTabNonFrontPressed`
: Text color for a tab that is not the frontmost of a tab control, when the tab is being clicked on by the user. Available with Appearance Manager 1.1 and later.

`kThemeTextColorTabFrontInactive`
: Text color for the front tab of an inactive tab control. Available with Appearance Manager 1.1 and later.

`kThemeTextColorTabNonFrontInactive`
: Text color for an inactive tab that is not the frontmost of a tab control. The tab may either be inactive because it has been individually disabled or because the tab control as a whole is currently inactive. Available with Appearance Manager 1.1 and later.

`kThemeTextColorIconLabelSelected`
: Text color for the label of an icon that is currently selected. Available with Appearance Manager 1.1 and later.

`kThemeTextColorBevelButtonStickyActive`
: Text color for an active bevel button that is currently on. Available with Appearance Manager 1.1 and later.

`kThemeTextColorBevelButtonStickyInactive`
: Text color for an inactive bevel button that is currently on. Available with Appearance Manager 1.1 and later.

`kThemeTextColorBlack`
: Black; this color does not change from theme to theme. You may use this constant instead of specifying a direct RGB value. Available with Appearance Manager 1.1 and later.

`kThemeTextColorWhite`
: White; this color does not change from theme to theme. You may use this constant instead of specifying a direct RGB value. Available with Appearance Manager 1.1 and later.

Theme Title Bar Item Constants

You may pass constants of type `ThemeTitleBarWidget` to the function `DrawThemeTitleBarWidget` (page 133) to draw specific types of window title bar items. The Appearance Manager draws a theme-compliant version of the title bar item type, as is appropriate. The `ThemeTitleBarWidget` constants are available with Appearance Manager 1.1 and later.

```
enum {
    kThemeWidgetCloseBox    = 0,
    kThemeWidgetZoomBox     = 1,
    kThemeWidgetCollapseBox = 2
};
typedef UInt16 ThemeTitleBarWidget;
```

Constant descriptions

`kThemeWindowCloseBox`
: Identifies a close box.

`kThemeWindowZoomBox`
: Identifies a zoom box.

`kThemeWindowCollapseBox`
: Identifies a collapse box.

Theme Track Attributes Constants

The ThemeTrackAttributes enumeration defines masks your application can use in the ThemeTrackDrawInfo (page 166) structure to specify various attributes of track controls. The ThemeTrackAttributes constants are available with Appearance Manager 1.1 and later.

```
enum {
    kThemeTrackHorizontal   = (1 << 0),
    kThemeTrackRightToLeft  = (1 << 1),
    kThemeTrackShowThumb    = (1 << 2)
};
typedef UInt16 ThemeTrackAttributes;
```

Constant descriptions

kThemeTrackHorizontal	If the bit specified by this mask is set, the track is horizontally, not vertically, oriented.
kThemeTrackRightToLeft	If the bit specified by this mask is set, values for the track increase from right to left if the track is horizontally oriented, or from bottom to top if the track is vertically oriented.
kThemeTrackShowThumb	If the bit specified by this mask is set, an indicator is drawn for this track.

Theme Track Enable State Constants

You can use constants of type ThemeTrackEnableState in the ThemeTrackDrawInfo (page 166) structure and in the functions GetThemeScrollBarTrackRect (page 118) and HitTestThemeScrollBarArrows (page 125) to identify the state of track controls. The ThemeTrackEnableState constants are available with Appearance Manager 1.1 and later.

```
enum {
    kThemeTrackActive           = 0,
    kThemeTrackDisabled         = 1,
    kThemeTrackNothingToScroll  = 2
};
typedef UInt8 ThemeTrackEnableState;
```

Constant descriptions

`kThemeTrackActive`	A track in the active state.
`kThemeTrackDisabled`	A track in the disabled state.
`kThemeTrackNothingToScroll`	For scroll bars, the window containing the track is expanded to a sufficiently large state such that all the content is viewable and there is nothing remaining to scroll.

Theme Track Kind Constants

You may use the following constants of type `ThemeTrackKind` to identify specific kinds of track-based controls to the Appearance Manager. The `ThemeTrackKind` constants are available with Appearance Manager 1.1 and later.

```
enum {
    kThemeScrollBar         = 0,
    kThemeSmallScrollBar    = 1,
    kThemeSlider            = 2,
    kThemeProgressBar       = 3,
    kThemeIndeterminateBar  = 4
};
typedef UInt16 ThemeTrackKind;
```

Constant descriptions

`kThemeScrollBar`	A scroll bar.
`kThemeSmallScrollBar`	A small scroll bar.
`kThemeSlider`	A slider bar.
`kThemeProgressBar`	A progress bar.
`kThemeIndeterminateBar`	An indeterminate progress bar.

Theme Track Press State Constants

You can use constants of type `ThemeTrackPressState` in structures of type `ScrollBarTrackInfo` (page 167) and `SliderTrackInfo` (page 168) to identify what is pressed in an active scroll bar or slider; the press state is ignored if the control is not active. The `ThemeTrackPressState` constants are available with Appearance Manager 1.1 and later.

Note that some constants are undefined when the corresponding arrows do not exist in the current visual appearance. Prior to using the `ThemeTrackPressState` constants, your application should call the function `GetThemeScrollBarArrowStyle` (page 116) to obtain the type of scroll bar arrows currently being used.

```
enum {
    kThemeLeftOutsideArrowPressed   = 0x01,
    kThemeLeftInsideArrowPressed    = 0x02,
    kThemeLeftTrackPressed          = 0x04,
    kThemeThumbPressed              = 0x08,
    kThemeRightTrackPressed         = 0x10,
    kThemeRightInsideArrowPressed   = 0x20,
    kThemeRightOutsideArrowPressed  = 0x40,
    kThemeTopOutsideArrowPressed    = kThemeLeftOutsideArrowPressed,
    kThemeTopInsideArrowPressed     = kThemeLeftInsideArrowPressed,
    kThemeTopTrackPressed           = kThemeLeftTrackPressed,
    kThemeBottomTrackPressed        = kThemeRightTrackPressed,
    kThemeBottomInsideArrowPressed  = kThemeRightInsideArrowPressed,
    kThemeBottomOutsideArrowPressed = kThemeRightOutsideArrowPressed
};
typedef UInt8 ThemeTrackPressState;
```

Constant descriptions

`kThemeLeftOutsideArrowPressed`
: For a horizontal scroll bar containing a single pair of arrows, with one arrow at each end, indicates that the arrow on the left is selected.
For a horizontal scroll bar containing a single pair of arrows, with both arrows on the right, this constant is undefined and should not be used.

`kThemeLeftInsideArrowPressed`
: For a horizontal scroll bar containing a single pair of

arrows, with one arrow at each end, this constant is undefined and should not be used.
For a horizontal scroll bar containing a single pair of arrows, with both arrows on the right, this constant is undefined and should not be used.

`kThemeLeftTrackPressed`
For a horizontal scroll bar or slider, indicates that the end of the track to the left of the scroll box or indicator is selected.

`kThemeThumbPressed`
Indicates that the scroll box or indicator is selected.

`kThemeRightTrackPressed`
For a horizontal scroll bar or slider, indicates that the end of the track to the right of the scroll box or indicator is selected.

`kThemeRightInsideArrowPressed`
For a horizontal scroll bar containing a single pair of arrows, with one arrow at each end, this constant is undefined and should not be used.
For a horizontal scroll bar containing a single pair of arrows, with both arrows on the right, indicates that the inner arrow at the right end of the scroll bar is selected.

`kThemeRightOutsideArrowPressed`
For a horizontal scroll bar containing a single pair of arrows, with one arrow at each end, indicates that the arrow on the right is selected.
For a horizontal scroll bar containing a single pair of arrows, with both arrows on the right, indicates that the outer arrow at the right end of the scroll bar is selected.

`kThemeTopOutsideArrowPressed`
For a vertical scroll bar containing a single pair of arrows, with one arrow at each end, indicates that the arrow on the top is selected.
For a vertical scroll bar containing a single pair of arrows, with both arrows on the bottom, this constant is undefined and should not be used.

`kThemeTopInsideArrowPressed`
For a vertical scroll bar containing a single pair of arrows, with one arrow at each end, this constant is undefined and should not be used.

For a vertical scroll bar containing a single pair of arrows, with both arrows on the bottom, this constant is undefined and should not be used.

`kThemeTopTrackPressed`
: For a vertical scroll bar or slider, indicates that the end of the track above the scroll box or indicator is selected.

`kThemeBottomTrackPressed`
: For a vertical scroll bar or slider, indicates that the end of the track beneath the scroll box or indicator is selected.

`kThemeBottomInsideArrowPressed`
: For a vertical scroll bar containing a single pair of arrows, with one arrow at each end, this constant is undefined and should not be used.
For a vertical scroll bar containing a single pair of arrows, with both arrows on the bottom, indicates that the inner arrow at the bottom end of the scroll bar is selected.

`kThemeBottomOutsideArrowPressed`
: For a vertical scroll bar containing a single pair of arrows, with one arrow at each end, indicates that the arrow on the bottom is selected.
For a vertical scroll bar containing a single pair of arrows, with both arrows on the bottom, indicates that the outer arrow at the bottom end of the scroll bar is selected.

Theme Window Attribute Constants

The `ThemeWindowAttributes` enumeration defines masks your application can use to specify the various interface elements that a given window contains. The `ThemeWindowAttributes` constants are available with Appearance Manager 1.1 and later.

```
enum {
    kThemeWindowHasGrow                = (1 << 0),
    kThemeWindowHasHorizontalZoom      = (1 << 3),
    kThemeWindowHasVerticalZoom        = (1 << 4),
    kThemeWindowHasFullZoom            = kThemeWindowHasHorizontalZoom
                                        + kThemeWindowHasVerticalZoom,
    kThemeWindowHasCloseBox            = (1 << 5),
    kThemeWindowHasCollapseBox         = (1 << 6),
    kThemeWindowHasTitleText            = (1 << 7),
    kThemeWindowIsCollapsed             = (1 << 8)
};
typedef UInt32 ThemeWindowAttributes;
```

Constant descriptions

`kThemeWindowHasGrow`
: If the bit specified by this mask is set, the window contains a size box.

`kThemeWindowHasHorizontalZoom`
: If the bit specified by this mask is set, the window contains a horizontal zoom box.

`kThemeWindowHasVerticalZoom`
: If the bit specified by this mask is set, the window contains a vertical zoom box.

`kThemeWindowHasFullZoom`
: If the bit specified by this mask is set, the window contains a full (horizontal and vertical) zoom box.

`kThemeWindowHasCloseBox`
: If the bit specified by this mask is set, the window contains a close box.

`kThemeWindowHasCollapseBox`
: If the bit specified by this mask is set, the window contains a collapse box.

`kThemeWindowHasTitleText`
: If the bit specified by this mask is set, the window contains a title.

`kThemeWindowIsCollapsed`
: If the bit specified by this mask is set, the window is currently collapsed.

Theme Window Type Constants

You may use the following constants of type `ThemeWindowType` to identify windows of specific visual categories to the Appearance Manager; see "Drawing Theme-Compliant Windows" (page 128) for the functions that use the `ThemeWindowType` constants. The `ThemeWindowType` constants are available with Appearance Manager 1.1 and later.

```
enum {
    kThemeDocumentWindow        = 0,
    kThemeDialogWindow          = 1,
    kThemeMovableDialogWindow   = 2,
    kThemeAlertWindow           = 3,
    kThemeMovableAlertWindow    = 4,
    kThemePlainDialogWindow     = 5,
    kThemeShadowDialogWindow    = 6,
    kThemePopupWindow           = 7,
    kThemeUtilityWindow         = 8,
    kThemeUtilitySideWindow     = 9
};
typedef UInt16 ThemeWindowType;
```

Constant descriptions

`kThemeDocumentWindow`	A document window.
`kThemeDialogWindow`	A modal dialog box.
`kThemeMovableDialogWindow`	A movable modal dialog box.
`kThemeAlertWindow`	An alert box.
`kThemeMovableAlertWindow`	A movable alert box.
`kThemePlainDialogWindow`	A plain modal dialog box. This window visually corresponds to that produced by the `plainDBox` pre–Appearance Manager window definition ID and does not change appearance by theme.
`kThemeShadowDialogWindow`	A dialog box with shadowing.
`kThemePopupWindow`	A pop-up window.
`kThemeUtilityWindow`	A utility window.
`kThemeUtilitySideWindow`	A utility window with a side title bar.

Result Codes

The most common result codes returned by Appearance Manager functions are listed below.

`noErr`	0	No error
`paramErr`	–50	Error in parameter list
`memFullErr`	–108	Not enough memory
`themeInvalidBrushErr`	–30560	Invalid brush color constant
`themeProcessRegisteredErr`	–30561	Application already registered as Appearance Manager client
`themeProcessNotRegisteredErr`	–30562	Application not registered as Appearance Manager client
`themeBadTextColorErr`	–30563	Invalid text color constant
`themeHasNoAccentsErr`	–30564	Theme does not support accent colors
`themeBadCursorIndexErr`	–30565	Invalid cursor constant
`themeScriptFontNotFoundErr`	–30566	No font record for specified script
`themeMonitorDepthNotSupportedErr`	–30567	Theme cannot be supported on all monitors at their current bit depth

Document Version History

This document has had the following releases:

Table A-1 *Programming With the Appearance Manager* revision history

Version	**Notes**
Apr. 21, 1999	The following changes were made from the prior (seed draft) version:
	Added "Introduction," "Using the Appearance Manager," and "About the Appearance Manager" chapters to contain programming discussions, code listings, artwork, and conceptual material.
Aug. 13, 1998	Updated document for the Appearance Manager 1.1 API.
	Removed "Appearance Manager Reference" chapter from the *Mac OS 8 Toolbox Reference* document. *Inside Macintosh: Appearance Manager Reference* is seeded as an independent (draft) document.
Jan. 15, 1998	The following changes were made:
	Noted information for Appearance Manager 1.0.2 where applicable.
Dec. 2, 1997	PDF formatting improved.
Nov. 3, 1997	First release of *Mac OS 8 Toolbox Reference*, which included the chapter "Appearance Manager Reference."

Index