Carbon Framework Reference

Carbon

2007-10-31

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Simultaneously published in the United States and Canada.

Contents

Figures and Tables

Introduction

Framework	/System/Library/Frameworks/Carbon
Header file directories	/System/Library/Frameworks/Carbon.framework/Headers
Declared in	AEDataModel.h ASDebugging.h ASRegistry.h Appearance.h AppleHelp.h AppleScript.h CarbonEvents.h CarbonEventsCore.h ColorPicker.h ControlDefinitions.h Controls.h Dialogs.h Drag.h Events.h FontPanel.h HIAccessibility.h HIArchive.h HIButtonViews.h HIClockView.h HICocoaView.h HIComboBox.h HIContainerViews.h HIDataBrowser.h HIDisclosureViews.h HIGeometry.h HIImageViews.h HILittleArrows.h HIMenuView.h HIObject.h HIPopupButton.h HIProgressViews.h HIRelevanceBar.h HIScrollView.h HISearchField.h HISegmentedView.h HISeparator.h HIShape.h HISlider.h HITabbedView.h HITextViews.h HIToolbar.h HIToolboxDebugging.h HIView.h

HIWindowViews.h
HTMLRendering.h
IBCarbonRuntime.h
IOMacOSTypes.h
Ink.h
Keyboards.h
KeychainCore.h
KeychainHI.h
Lists.h
MacApplication.h
MacHelp.h
MacTextEditor.h
MacWindows.h
Menus.h
Navigation.h
Notification.h
NumberFormatting.h
OSA.h
OSAComp.h
OSAGeneric.h
PMApplication.h
PMApplicationDeprecated.h
PMIOModule.h
PMPluginHeader.h
PMPrinterBrowsers.h
PMPrinterModuleDeprecated.h
PMPrintingDialogExtensionsDeprecated.h
QuickdrawTypes.h
Scrap.h
SpeechRecognition.h
StringCompare.h
TSMTE.h
TextEdit.h
TextServices.h
TextUtils.h
Translation.h
TranslationExtensions.h
TypeSelect.h
URLAccess.h
cssmspi.h

This collection of documents provides the API reference for many fundamental user experience features—such as windows, menus, text handling, and event management—used in Carbon applications.

The Carbon framework also includes support for a number of legacy technologies—such as TextEdit and the Event Manager—that have been superseded by newer technologies like Multilingual Text Engine (MLTE) and the Carbon Event Manager.

Managers

Appearance Manager Reference

Framework: Carbon/Carbon.h

Declared in Appearance.h
HITheme.h

Overview

The Appearance Manager coordinates the look of human interface elements in Mac OS X. You can use the Appearance Manager to adapt any nonstandard interface elements in your program to the same coordinated look as the rest of Mac OS X. 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.

In Mac OS X v10.3 and later, the Appearance Manager provides a new API called HITheme for drawing appearance primitives. Currently, this API is documented in the `HITheme.h` interface file. The HITheme API is similar to the legacy Appearance Manager API, but many parameters have been modified to use Quartz 2D types instead of QuickDraw types. For example, HITheme functions draw into a Quartz graphics context instead of the current QuickDraw graphics port. The legacy Appearance Manager API is implemented on top of the HITheme API, so using the new API can provide a significant performance advantage.

Functions by Task

Accessing Theme Information

`CopyThemeIdentifier` (page 28)
Retrieves a string identifying the current theme variant.

`GetTheme` (page 59)
Obtains a collection containing data describing the current theme.

`GetThemeMetric` (page 70)
Retrieves the value of a metric property of a user interface element.

`GetThemeFont` (page 65) Deprecated in Mac OS X v10.5
Obtains information about a system font in the current theme. (Deprecated. Some theme fonts cannot be drawn using QuickDraw; use `HIThemeDrawTextBox` instead.)

`IsValidAppearanceFileType` (page 90) Deprecated in Mac OS X v10.5
Returns whether the system can interpret files of a given file type as appearance files. (Deprecated. There is no replacement function.)

`IterateThemes` (page 91) Deprecated in Mac OS X v10.5
: Iterates over all themes installed on a system. (Deprecated. There is no replacement function.)

`SetTheme` (page 97) Deprecated in Mac OS X v10.5
: Sets a specified collection as the current theme. (Deprecated. There is no replacement function.)

`UseThemeFont` (page 103) Deprecated in Mac OS X v10.5
: Sets the font of the current graphics port to one of the current theme's system fonts. (Deprecated. Use `HIThemeDrawTextBox` instead.)

Drawing Theme-Compliant Controls

`GetThemeCheckBoxStyle` (page 64)
: Obtains the system preference for the type of mark to use in a checkbox.

`GetThemeScrollBarArrowStyle` (page 70)
: Obtains the system preference for the type of scroll bar arrows to be used.

`GetThemeScrollBarThumbStyle` (page 71)
: Obtains the system preference for the type of scroll box to be used.

`DrawThemeButton` (page 31) Deprecated in Mac OS X v10.5
: Draws a button. (Deprecated. Use `HIThemeDrawButton` instead.)

`DrawThemeChasingArrows` (page 33) Deprecated in Mac OS X v10.5
: Draws an asynchronous arrows indicator. (Deprecated. Use `HIThemeDrawChasingArrows` instead.)

`DrawThemeEditTextFrame` (page 34) Deprecated in Mac OS X v10.5
: Draws an editable text frame. (Deprecated. Use `HIThemeDrawFrame` instead.)

`DrawThemeFocusRect` (page 34) Deprecated in Mac OS X v10.5
: Draws or erases a focus ring around a specified rectangle. (Deprecated. Use `HIThemeDrawFocusRect` instead.)

`DrawThemeFocusRegion` (page 35) Deprecated in Mac OS X v10.5
: Draws or erases a focus ring around a specified region. (Deprecated. Use `HIThemeDrawFocusRect` instead.)

`DrawThemeGenericWell` (page 36) Deprecated in Mac OS X v10.5
: Draws an image well frame. (Deprecated. Use `HIThemeDrawGenericWell` instead.)

`DrawThemeListBoxFrame` (page 37) Deprecated in Mac OS X v10.5
: Draws a list box frame. (Deprecated. Use `HIThemeDrawFrame` instead.)

`DrawThemePlacard` (page 42) Deprecated in Mac OS X v10.5
: Draws a placard. (Deprecated. Use `HIThemeDrawPlacard` instead.)

`DrawThemePopupArrow` (page 43) Deprecated in Mac OS X v10.5
: Draws a pop-up arrow. (Deprecated. Use `HIThemeDrawPopupArrow` instead.)

`DrawThemePrimaryGroup` (page 44) Deprecated in Mac OS X v10.5
: Draws a primary group box frame. (Deprecated. Use `HIThemeDrawGroupBox` instead.)

`DrawThemeScrollBarArrows` (page 45) Deprecated in Mac OS X v10.5
: Draws scroll bar arrows consistent with the current system preferences. (Deprecated. Use `HIThemeDrawTrack`, which draws the entire scrollbar including both the track and arrows.)

`DrawThemeSecondaryGroup` (page 46) Deprecated in Mac OS X v10.5
: Draws a secondary group box frame. (Deprecated. Use `HIThemeDrawGroupBox` instead.)

`DrawThemeSeparator` (page 47) Deprecated in Mac OS X v10.5
Draws a separator line. (Deprecated. Use `HIThemeDrawSeparator` instead.)

`DrawThemeTab` (page 50) Deprecated in Mac OS X v10.5
Draws a tab. (Deprecated. Use `HIThemeDrawTab` instead.)

`DrawThemeTabPane` (page 51) Deprecated in Mac OS X v10.5
Draws a tab pane. (Deprecated. Use `HIThemeDrawTabPane` instead.)

`DrawThemeTickMark` (page 53) Deprecated in Mac OS X v10.5
Draws a tick mark. (Deprecated. Use `HIThemeDrawTickMark` instead.)

`DrawThemeTrack` (page 55) Deprecated in Mac OS X v10.5
Draws a track. (Deprecated. Use `HIThemeDrawTrack` instead.)

`DrawThemeTrackTickMarks` (page 56) Deprecated in Mac OS X v10.5
Draws tick marks for a track. (Deprecated. Use `HIThemeDrawTrackTickMarks` instead.)

`GetThemeButtonBackgroundBounds` (page 62) Deprecated in Mac OS X v10.5
Obtains the rectangle that contains a button. (Deprecated. Use `HIThemeGetButtonBackgroundBounds` instead.)

`GetThemeButtonContentBounds` (page 62) Deprecated in Mac OS X v10.5
Obtains the rectangle where content can be drawn for a button. (Deprecated. Use `HIThemeGetButtonContentBounds` instead.)

`GetThemeButtonRegion` (page 63) Deprecated in Mac OS X v10.5
Obtains the region occupied by a button. (Deprecated. Use `HIThemeGetButtonShape` instead.)

`GetThemeScrollBarTrackRect` (page 71) Deprecated in Mac OS X v10.5
Obtains the area containing the track portion of a scroll bar. (Deprecated. Use `HIThemeGetScrollBarTrackRect` instead.)

`GetThemeTabRegion` (page 73) Deprecated in Mac OS X v10.5
Obtains the region occupied by a tab. (Deprecated. Use `HIThemeGetTabDrawShape` instead.)

`GetThemeTrackBounds` (page 76) Deprecated in Mac OS X v10.5
Obtains the bounding rectangle of a track. (Deprecated. Use `HIThemeGetTrackBounds` instead.)

`GetThemeTrackDragRect` (page 77) Deprecated in Mac OS X v10.5
Obtains the area in which the user may drag a track's indicator. (Deprecated. Use `HIThemeGetTrackDragRect` instead.)

`GetThemeTrackLiveValue` (page 78) Deprecated in Mac OS X v10.5
Obtains the current value of a track's indicator, given its relative position. (Deprecated. Use `HIThemeGetTrackLiveValue` instead.)

`GetThemeTrackThumbPositionFromOffset` (page 79) Deprecated in Mac OS X v10.5
Obtains the relative position of a track's indicator, given an offset from its prior position. (Deprecated. Use `HIThemeGetTrackThumbPositionFromOffset` instead.)

`GetThemeTrackThumbPositionFromRegion` (page 79) Deprecated in Mac OS X v10.5
Obtains the relative position of a track's indicator, given its current position. (Deprecated. use `HIThemeGetTrackThumbPositionFromBounds` instead.)

`GetThemeTrackThumbRgn` (page 80) Deprecated in Mac OS X v10.5
Obtains the region containing a track's indicator. (Deprecated. Use `HIThemeGetTrackThumbShape` instead.)

`HitTestThemeScrollBarArrows` (page 83) Deprecated in Mac OS X v10.5
Returns whether the user clicked upon the specified scroll bar's arrows. (Deprecated. Use `HIThemeHitTestScrollBarArrows` instead.)

`HitTestThemeTrack` (page 84) Deprecated in Mac OS X v10.5
Returns whether the user clicked upon the specified track. (Deprecated. Use `HIThemeHitTestTrack` instead.)

Drawing Theme-Compliant Menus

`GetThemeMenuBarHeight` (page 67)
Obtains the height of a menu bar.

`GetThemeMenuItemExtra` (page 68)
Obtains a measurement of the space surrounding a menu item.

`GetThemeMenuSeparatorHeight` (page 68)
Obtains the height of a menu separator line.

`GetThemeMenuTitleExtra` (page 69)
Obtains a measurement of the space to either side of a menu title.

`DrawThemeMenuBackground` (page 37) Deprecated in Mac OS X v10.5
Draws a menu background. (Deprecated. Use `HIThemeDrawMenuBackground` instead.)

`DrawThemeMenuBarBackground` (page 38) Deprecated in Mac OS X v10.5
Draws a menu bar background. (Deprecated. Use `HIThemeDrawMenuBarBackground` instead.)

`DrawThemeMenuItem` (page 38) Deprecated in Mac OS X v10.5
Draws a menu item. (Deprecated. Use `HIThemeDrawMenuItem` instead.)

`DrawThemeMenuSeparator` (page 40) Deprecated in Mac OS X v10.5
Draws a menu item separator line. (Deprecated. Use `HIThemeDrawMenuSeparator` instead.)

`DrawThemeMenuTitle` (page 40) Deprecated in Mac OS X v10.5
Draws a menu title. (Deprecated. Use `HIThemeDrawMenuTitle` instead.)

`GetThemeMenuBackgroundRegion` (page 66) Deprecated in Mac OS X v10.5
Obtains the background region for a menu. (Deprecated. Use `HIThemeGetMenuBackgroundShape` instead.)

Drawing Theme-Compliant Windows

`DrawThemeModelessDialogFrame` (page 42) Deprecated in Mac OS X v10.5
Draws a beveled outline inside the content area of a modeless dialog box. (Deprecated. Use `HIThemeDrawWindowFrame` instead.)

`DrawThemeScrollBarDelimiters` (page 46) Deprecated in Mac OS X v10.5
Outlines a window's scroll bars. (Deprecated. Use `HIThemeDrawScrollBarDelimiters` instead.)

`DrawThemeStandaloneGrowBox` (page 48) Deprecated in Mac OS X v10.5
Draws a size box. (Deprecated. Use `HIThemeDrawGrowBox` instead.)

`DrawThemeStandaloneNoGrowBox` (page 49) Deprecated in Mac OS X v10.5
Draws a fill image for use in the corner space between scroll bars. (Deprecated. Use `HIThemeDrawGrowBox` instead.)

`DrawThemeTitleBarWidget` (page 54) Deprecated in Mac OS X v10.5
Draws a close box, zoom box, or collapse box. (Deprecated. Use `HIThemeDrawTitleBarWidget` instead.)

`DrawThemeWindowFrame` (page 57) Deprecated in Mac OS X v10.5
Draws a window frame. (Deprecated. Use `HIThemeDrawWindowFrame` instead.)

`DrawThemeWindowHeader` (page 58) Deprecated in Mac OS X v10.5
Draws a window header. (Deprecated. Use `HIThemeDrawHeader` instead.)

`DrawThemeWindowListViewHeader` (page 58) Deprecated in Mac OS X v10.5
Draws a window list view header. (Deprecated. Use `HIThemeDrawHeader` instead.)

`GetThemeStandaloneGrowBoxBounds` (page 72) Deprecated in Mac OS X v10.5
Obtains the bounds of a size box. (Deprecated. Use `HIThemeGetGrowBoxBounds` instead.)

`GetThemeWindowRegion` (page 81) Deprecated in Mac OS X v10.5
Obtains the specified window region. (Deprecated. Use `HIThemeGetWindowShape` instead.)

`GetThemeWindowRegionHit` (page 82) Deprecated in Mac OS X v10.5
Obtains the part of the window that the user clicked upon. (Deprecated. Use `HIThemeGetWindowRegionHit` instead.)

Playing Theme Sounds

The theme sound functions do nothing in Mac OS X.

`BeginThemeDragSound` (page 27)
Continuously plays a theme-specific sound associated with the user's movement of a given interface object.

`EndThemeDragSound` (page 59)
Terminates the playing of a sound associated with the user's movement of a given interface object.

`PlayThemeSound` (page 94)
Plays an asynchronous sound associated with the specified state change.

Registering With the Appearance Manager

`IsAppearanceClient` (page 88) Deprecated in Mac OS X v10.5
Returns whether a given process is currently registered as a client of the Appearance Manager. (Deprecated. There is no replacement function.)

`RegisterAppearanceClient` (page 95) Deprecated in Mac OS X v10.5
Registers your program with the Appearance Manager. (Deprecated. There is no replacement function.)

`UnregisterAppearanceClient` (page 103) Deprecated in Mac OS X v10.5
Informs the Appearance Manager that your program is no longer its client. (Deprecated. There is no replacement function.)

Specifying Theme-Compliant Cursors

`SetAnimatedThemeCursor` (page 96)
Animates a version of the specified cursor type that is consistent with the current theme.

`SetThemeCursor` (page 98)
Sets the cursor to a version of the specified cursor type that is consistent with the current theme.

Using Theme-Compliant Colors and Patterns

`DisposeThemeDrawingState` (page 29)
Releases the memory associated with a reference to a graphics port's drawing state.

`GetThemeBrushAsColor` (page 61)
Obtains the color that corresponds to a given theme brush type under the current theme.

`GetThemeDrawingState` (page 65)
Obtains the drawing state of the current graphics port.

`GetThemeTextColor` (page 74)
Obtains the text color used for a specified element under the current theme.

`NormalizeThemeDrawingState` (page 94)
Sets the current graphics port to a default drawing state.

`SetThemeDrawingState` (page 99)
Sets the drawing state of the current graphics port.

`ApplyThemeBackground` (page 26) Deprecated in Mac OS X v10.5
Sets the background color or pattern of the current port to be consistent with that of an embedding object. (Deprecated. Use `HIThemeApplyBackground` instead.)

`GetThemeAccentColors` (page 60) Deprecated in Mac OS X v10.5
Obtains a copy of a theme's accent colors. (Deprecated. There is no replacement function.)

`IsThemeInColor` (page 89) Deprecated in Mac OS X v10.5
Returns whether the current theme would draw in color in the given environment. (Deprecated. There is no replacement function.)

`SetThemeBackground` (page 98) Deprecated in Mac OS X v10.5
Applies a theme-compliant color or pattern to the background of the current port. (Deprecated. Use `HIThemeSetFill` and draw using Quartz 2D.)

`SetThemePen` (page 100) Deprecated in Mac OS X v10.5
Applies a theme-compliant color or pattern to the foreground of the current port. (Deprecated. Use `HIThemeSetStroke` and draw using Quartz 2D.)

`SetThemeTextColor` (page 101) Deprecated in Mac OS X v10.5
Sets the current text color to be consistent with that of a specified element. (Deprecated. Use `HIThemeSetTextFill` and draw with Quartz 2D, ATSUI, or `HIThemeDrawTextBox`.)

Drawing Theme-Compliant Text

`GetThemeTextShadowOutset` (page 76)
Tells you the amount of space taken up by the shadow for a given font and drawing state combination.

`DrawThemeTextBox` (page 51) Deprecated in Mac OS X v10.5
Draws text into the area you specify. (Deprecated. Use `HIThemeDrawTextBox` instead.)

`GetThemeTextDimensions` (page 74) Deprecated in Mac OS X v10.5
Tells you the height, width, and baseline for a string. (Deprecated. Use `HIThemeGetTextDimensions` instead.)

`TruncateThemeText` (page 102) Deprecated in Mac OS X v10.5
Truncates text to fit within the width you specify. (Deprecated. There is no replacement function; use `HIThemeGetTextDimensions` or `HIThemeDrawTextBox` instead.)

Creating and Disposing Universal Procedure Pointers to Appearance Manager Callbacks

`DisposeMenuItemDrawingUPP` (page 28) Deprecated in Mac OS X v10.5
Disposes of the UPP to a menu item drawing function. (Deprecated. There is no replacement function.)

`DisposeMenuTitleDrawingUPP` (page 29) Deprecated in Mac OS X v10.5
Disposes of the UPP to a menu title drawing function. (Deprecated. There is no replacement function.)

`DisposeThemeButtonDrawUPP` (page 29) Deprecated in Mac OS X v10.5
Disposes of the UPP to a button drawing function. (Deprecated. There is no replacement function.)

`DisposeThemeEraseUPP` (page 30) Deprecated in Mac OS X v10.5
Disposes of the UPP to a background drawing callback function. (Deprecated. There is no replacement function.)

`DisposeThemeIteratorUPP` (page 30) Deprecated in Mac OS X v10.5
Disposes of the UPP to a theme iteration callback function. (Deprecated. There is no replacement function.)

`DisposeThemeTabTitleDrawUPP` (page 31) Deprecated in Mac OS X v10.5
Disposes of the UPP to a tab title drawing function. (Deprecated. There is no replacement function.)

`DisposeWindowTitleDrawingUPP` (page 31) Deprecated in Mac OS X v10.5
Disposes of the UPP to a window title drawing function. (Deprecated. There is no replacement function.)

`InvokeMenuItemDrawingUPP` (page 85) Deprecated in Mac OS X v10.5
Invokes your menu item drawing function. (Deprecated. There is no replacement function.)

`InvokeMenuTitleDrawingUPP` (page 85) Deprecated in Mac OS X v10.5
Invokes your menu title drawing function. (Deprecated. There is no replacement function.)

`InvokeThemeButtonDrawUPP` (page 86) Deprecated in Mac OS X v10.5
Invokes your button drawing function. (Deprecated. There is no replacement function.)

`InvokeThemeEraseUPP` (page 86) Deprecated in Mac OS X v10.5
Invokes your background drawing callback function. (Deprecated. There is no replacement function.)

`InvokeThemeIteratorUPP` (page 87) Deprecated in Mac OS X v10.5
Invokes your theme iteration callback function. (Deprecated. There is no replacement function.)

`InvokeThemeTabTitleDrawUPP` (page 87) Deprecated in Mac OS X v10.5
Invokes your tab title drawing function. (Deprecated. There is no replacement function.)

`InvokeWindowTitleDrawingUPP` (page 88) Deprecated in Mac OS X v10.5
Invokes your window title drawing function. (Deprecated. There is no replacement function.)

`NewMenuItemDrawingUPP` (page 91) Deprecated in Mac OS X v10.5
Creates a new universal procedure pointer (UPP) to a menu item drawing function. (Deprecated. There is no replacement function.)

`NewMenuTitleDrawingUPP` (page 92) Deprecated in Mac OS X v10.5
Creates a new universal procedure pointer (UPP) to a menu title drawing function. (Deprecated. There is no replacement function.)

`NewThemeButtonDrawUPP` (page 92) Deprecated in Mac OS X v10.5
Creates a new universal procedure pointer (UPP) to a button drawing function. (Deprecated. There is no replacement function.)

`NewThemeEraseUPP` (page 92) Deprecated in Mac OS X v10.5
Creates a new universal procedure pointer (UPP) to a background drawing callback function. (Deprecated. There is no replacement function.)

`NewThemeIteratorUPP` (page 93) Deprecated in Mac OS X v10.5
: Creates a new universal procedure pointer (UPP) to a theme iteration callback function. (Deprecated. There is no replacement function.)

`NewThemeTabTitleDrawUPP` (page 93) Deprecated in Mac OS X v10.5
: Creates a new universal procedure pointer (UPP) to a tab title drawing function. (Deprecated. There is no replacement function.)

`NewWindowTitleDrawingUPP` (page 94) Deprecated in Mac OS X v10.5
: Creates a new universal procedure pointer (UPP) to a window title drawing function. (Deprecated. There is no replacement function.)

Functions

ApplyThemeBackground

Sets the background color or pattern of the current port to be consistent with that of an embedding object. (Deprecated in Mac OS X v10.5. Use `HIThemeApplyBackground` instead.)

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

Parameters

inKind
: A value of type `ThemeBackgroundKind`. Pass a constant specifying the type of embedding object. See "Theme Backgrounds" (page 144) 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 Drawing States" (page 126) 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.

Return Value

A result code. See "Appearance Manager Result Codes" (page 217).

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 98), 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.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Appearance.h`

BeginThemeDragSound

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

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

Parameters
kind
A value of type `ThemeDragSoundKind`. Pass a constant specifying the sound to play; see "Theme Drag Sounds" (page 210) for descriptions of possible values.

Return Value
A result code. See "Appearance Manager Result Codes" (page 217).

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 59), 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.

Special Considerations

This function is not implemented in Mac OS X.

Availability

Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In

`Appearance.h`

CopyThemeIdentifier

Retrieves a string identifying the current theme variant.

```
OSStatus CopyThemeIdentifier (
   CFStringRef *outIdentifier
);
```

Parameters

outIdentifier

A pointer to a string that, on output, contains the current theme variant. When you no longer need the string, you should release it.

Return Value

A result code. See "Appearance Manager Result Codes" (page 217).

Availability

Available in Mac OS X v10.1 and later.

Declared In

`Appearance.h`

DisposeMenuItemDrawingUPP

Disposes of the UPP to a menu item drawing function. (Deprecated in Mac OS X v10.5. There is no replacement function.)

```
void DisposeMenuItemDrawingUPP (
   MenuItemDrawingUPP userUPP
);
```

Parameters

userUPP

The UPP to dispose of.

Availability

Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In

`Appearance.h`

DisposeMenuTitleDrawingUPP

Disposes of the UPP to a menu title drawing function. (Deprecated in Mac OS X v10.5. There is no replacement function.)

```
void DisposeMenuTitleDrawingUPP (
   MenuTitleDrawingUPP userUPP
);
```

Parameters

userUPP

The UPP to dispose of.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`Appearance.h`

DisposeThemeButtonDrawUPP

Disposes of the UPP to a button drawing function. (Deprecated in Mac OS X v10.5. There is no replacement function.)

```
void DisposeThemeButtonDrawUPP (
   ThemeButtonDrawUPP userUPP
);
```

Parameters

userUPP

The UPP to dispose of.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`Appearance.h`

DisposeThemeDrawingState

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

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

Parameters

inState

A value of type `ThemeDrawingState`. 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 65).

Return Value
A result code. See "Appearance Manager Result Codes" (page 217).

Discussion
Also see the function `SetThemeDrawingState` (page 99).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Appearance.h`

DisposeThemeEraseUPP

Disposes of the UPP to a background drawing callback function. (Deprecated in Mac OS X v10.5. There is no replacement function.)

```
void DisposeThemeEraseUPP (
   ThemeEraseUPP userUPP
);
```

Parameters
userUPP
The UPP to dispose of.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`Appearance.h`

DisposeThemeIteratorUPP

Disposes of the UPP to a theme iteration callback function. (Deprecated in Mac OS X v10.5. There is no replacement function.)

```
void DisposeThemeIteratorUPP (
   ThemeIteratorUPP userUPP
);
```

Parameters
userUPP
The UPP to dispose of.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`Appearance.h`

DisposeThemeTabTitleDrawUPP

Disposes of the UPP to a tab title drawing function. (Deprecated in Mac OS X v10.5. There is no replacement function.)

```
void DisposeThemeTabTitleDrawUPP (
   ThemeTabTitleDrawUPP userUPP
);
```

Parameters

userUPP

The UPP to dispose of.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`Appearance.h`

DisposeWindowTitleDrawingUPP

Disposes of the UPP to a window title drawing function. (Deprecated in Mac OS X v10.5. There is no replacement function.)

```
void DisposeWindowTitleDrawingUPP (
   WindowTitleDrawingUPP userUPP
);
```

Parameters

userUPP

The UPP to dispose of.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`Appearance.h`

DrawThemeButton

Draws a button. (Deprecated in Mac OS X v10.5. Use `HIThemeDrawButton` instead.)

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

Parameters

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 Buttons" (page 153) for descriptions of possible values.

inNewInfo

A pointer to a structure of type `ThemeButtonDrawInfo`. 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`. 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`. If you have a custom background, use this parameter to pass a universal function pointer to an application-defined function such as that described in `ThemeEraseProcPtr` (page 108). `DrawThemeButton` calls this function to erase the background before drawing the button. If you pass `NULL`, `DrawThemeButton`'s default behavior is to erase the background for you.

inLabelProc

A value of type `ThemeButtonDrawUPP`. If you pass a universal function pointer to an application-defined function such as that described in `ThemeButtonDrawProcPtr` (page 107), `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.

Return Value

A result code. See "Appearance Manager Result Codes" (page 217).

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 62) to obtain the actual rectangle containing the pixels belonging to a button under the current theme.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Appearance.h`

DrawThemeChasingArrows

Draws an asynchronous arrows indicator. (Deprecated in Mac OS X v10.5. Use `HIThemeDrawChasingArrows` instead.)

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

Parameters

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 Drawing States" (page 126). The asynchronous arrows indicator can be drawn as active or inactive; passing `kThemeStatePressed` produces an error.

eraseProc
A value of type `ThemeEraseUPP`. If you have a custom background, pass a universal function pointer to an application-defined function such as that described in `ThemeEraseProcPtr` (page 108). `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.

Return Value
A result code. See "Appearance Manager Result Codes" (page 217).

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

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Appearance.h`

DrawThemeEditTextFrame

Draws an editable text frame. (Deprecated in Mac OS X v10.5. Use `HIThemeDrawFrame` instead.)

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

Parameters

`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 Drawing States" (page 126). The frame can be drawn as active or inactive; passing `kThemeStatePressed` produces an error.

Return Value
A result code. See "Appearance Manager Result Codes" (page 217).

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 34).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Appearance.h`

DrawThemeFocusRect

Draws or erases a focus ring around a specified rectangle. (Deprecated in Mac OS X v10.5. Use `HIThemeDrawFocusRect` instead.)

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

Parameters

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.

Return Value

A result code. See "Appearance Manager Result Codes" (page 217).

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.

If you are drawing a focus ring around an element for which you have drawn a frame using `DrawThemeEditTextFrame` (page 34) or `DrawThemeListBoxFrame` (page 37), 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.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Appearance.h`

DrawThemeFocusRegion

Draws or erases a focus ring around a specified region. (Deprecated in Mac OS X v10.5. Use `HIThemeDrawFocusRect` instead.)

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

Parameters

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.

Return Value
A result code. See "Appearance Manager Result Codes" (page 217).

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.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Appearance.h`

DrawThemeGenericWell

Draws an image well frame. (Deprecated in Mac OS X v10.5. Use `HIThemeDrawGenericWell` instead.)

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

Parameters

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 Drawing States" (page 126). 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`.

Return Value
A result code. See "Appearance Manager Result Codes" (page 217).

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.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Appearance.h`

DrawThemeListBoxFrame

Draws a list box frame. (Deprecated in Mac OS X v10.5. Use `HIThemeDrawFrame` instead.)

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

Parameters

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 Drawing States" (page 126). The frame can be drawn as active or inactive; passing `kThemeStatePressed` produces an error.

Return Value

A result code. See "Appearance Manager Result Codes" (page 217).

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 34).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Appearance.h`

DrawThemeMenuBackground

Draws a menu background. (Deprecated in Mac OS X v10.5. Use `HIThemeDrawMenuBackground` instead.)

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

Parameters

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 Types" (page 176) for descriptions of possible values.

Return Value

A result code. See "Appearance Manager Result Codes" (page 217).

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

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Appearance.h`

DrawThemeMenuBarBackground

Draws a menu bar background. (Deprecated in Mac OS X v10.5. Use `HIThemeDrawMenuBarBackground` instead.)

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

Parameters

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 States" (page 177).

inAttributes
A value indicating the attributes of the menu bar. Pass 0 for a standard menu bar or `kThemeMenuSquareMenuBar` for a menu bar with square corners.

Return Value
A result code. See "Appearance Manager Result Codes" (page 217).

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

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Appearance.h`

DrawThemeMenuItem

Draws a menu item. (Deprecated in Mac OS X v10.5. Use `HIThemeDrawMenuItem` instead.)

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

Parameters

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 68) 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 States" (page 176).

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 Types" (page 177) for descriptions of possible values.

inDrawProc

A value of type `MenuItemDrawingUPP`. Pass a universal function pointer to a menu item drawing function such as `MenuItemDrawingProcPtr` (page 104). The value of the `inDrawProc` parameter can be a valid universal function pointer or `NULL`.

inUserData

An unsigned 32-bit integer. Provide any data to be passed in to the `inUserData` parameter of `MenuItemDrawingProcPtr` (page 104).

Return Value
A result code. See "Appearance Manager Result Codes" (page 217).

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

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Appearance.h`

DrawThemeMenuSeparator

Draws a menu item separator line. (Deprecated in Mac OS X v10.5. Use `HIThemeDrawMenuSeparator` instead.)

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

Parameters

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 68).

Return Value
A result code. See "Appearance Manager Result Codes" (page 217).

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

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Appearance.h`

DrawThemeMenuTitle

Draws a menu title. (Deprecated in Mac OS X v10.5. Use `HIThemeDrawMenuTitle` instead.)

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

Parameters

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 67) 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 69) 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 States" (page 176).

inAttributes

Reserved. Pass 0.

inTitleProc

A value of type `MenuTitleDrawingUPP`. Pass a universal function pointer to a menu title drawing function such as `MenuTitleDrawingProcPtr` (page 105), defining how to draw the contents of the menu title. The value of the `inTitleProc` parameter can be a valid universal function pointer or `NULL`.

inTitleData

An unsigned 32-bit integer. Provide any data to be passed in to the `inUserData` parameter of `MenuTitleDrawingProcPtr` (page 105).

Return Value

A result code. See "Appearance Manager Result Codes" (page 217).

Discussion

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

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Appearance.h`

DrawThemeModelessDialogFrame

Draws a beveled outline inside the content area of a modeless dialog box. (Deprecated in Mac OS X v10.5. Use `HIThemeDrawWindowFrame` instead.)

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

Parameters

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 Drawing States" (page 126) for descriptions of possible values. The frame can be drawn as active or inactive; passing `kThemeStatePressed` produces an error.

Return Value

A result code. See "Appearance Manager Result Codes" (page 217).

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.

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.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Appearance.h`

DrawThemePlacard

Draws a placard. (Deprecated in Mac OS X v10.5. Use `HIThemeDrawPlacard` instead.)

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

Parameters

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 Drawing States" (page 126). The placard can be drawn as active, inactive, or pressed.

Return Value
A result code. See "Appearance Manager Result Codes" (page 217).

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

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Appearance.h`

DrawThemePopupArrow

Draws a pop-up arrow. (Deprecated in Mac OS X v10.5. Use `HIThemeDrawPopupArrow` instead.)

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

Parameters

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 Orientations" (page 160) 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 Sizes" (page 161) 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 Drawing States" (page 126) for descriptions of possible values.

eraseProc

A value of type `ThemeEraseUPP`. If you have a custom background, pass a universal function pointer to an application-defined function such as that described in `ThemeEraseProcPtr` (page 108). `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.

Return Value
A result code. See "Appearance Manager Result Codes" (page 217).

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.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Appearance.h`

DrawThemePrimaryGroup

Draws a primary group box frame. (Deprecated in Mac OS X v10.5. Use `HIThemeDrawGroupBox` instead.)

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

Parameters

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 Drawing States" (page 126). The frame can be drawn as active or inactive; passing `kThemeStatePressed` produces an error.

Return Value
A result code. See "Appearance Manager Result Codes" (page 217).

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.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Appearance.h`

DrawThemeScrollBarArrows

Draws scroll bar arrows consistent with the current system preferences. (Deprecated in Mac OS X v10.5. Use `HIThemeDrawTrack`, which draws the entire scrollbar including both the track and arrows.)

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

Parameters

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 States" (page 185) 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 States" (page 187) 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.

Return Value

A result code. See "Appearance Manager Result Codes" (page 217).

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

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Appearance.h`

DrawThemeScrollBarDelimiters

Outlines a window's scroll bars. (Deprecated in Mac OS X v10.5. Use `HIThemeDrawScrollBarDelimiters` instead.)

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

Parameters

flavor

A value of type `ThemeWindowType`. Pass a constant specifying the type of window for which to draw scroll bar delimiters. See "Theme Window Types" (page 189) 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 Drawing States" (page 126) 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 Attributes" (page 191) for descriptions of possible values. Pass 0 if the window has none of the enumerated attributes.

Return Value

A result code. See "Appearance Manager Result Codes" (page 217).

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.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Appearance.h`

DrawThemeSecondaryGroup

Draws a secondary group box frame. (Deprecated in Mac OS X v10.5. Use `HIThemeDrawGroupBox` instead.)

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

Parameters

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 Drawing States" (page 126). The frame can be drawn as active or inactive; passing `kThemeStatePressed` produces an error.

Return Value

A result code. See "Appearance Manager Result Codes" (page 217).

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.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Appearance.h`

DrawThemeSeparator

Draws a separator line. (Deprecated in Mac OS X v10.5. Use `HIThemeDrawSeparator` instead.)

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

Parameters

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 Drawing States" (page 126). The separator line can be drawn as active or inactive; passing `kThemeStatePressed` produces an error.

Return Value

A result code. See "Appearance Manager Result Codes" (page 217).

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.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Appearance.h`

DrawThemeStandaloneGrowBox

Draws a size box. (Deprecated in Mac OS X v10.5. Use `HIThemeDrawGrowBox` instead.)

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

Parameters

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 Directions" (page 180) 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 Drawing States" (page 126) for descriptions of these values.

Return Value
A result code. See "Appearance Manager Result Codes" (page 217).

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.

Also see the function `DrawThemeStandaloneNoGrowBox` (page 49).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Appearance.h`

DrawThemeStandaloneNoGrowBox

Draws a fill image for use in the corner space between scroll bars. (Deprecated in Mac OS X v10.5. Use `HIThemeDrawGrowBox` instead.)

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

Parameters

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 Directions" (page 180) 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 Drawing States" (page 126) for descriptions of these values.

Return Value
A result code. See "Appearance Manager Result Codes" (page 217).

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

Also see the function `DrawThemeStandaloneGrowBox` (page 48).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Appearance.h`

DrawThemeTab

Draws a tab. (Deprecated in Mac OS X v10.5. Use `HIThemeDrawTab` instead.)

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

Parameters

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 Styles" (page 183) 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 Directions" (page 182) for descriptions of possible values.

labelProc

A value of type `ThemeTabTitleDrawUPP`. Pass a universal function pointer to an application-defined function such as that described in `ThemeTabTitleDrawProcPtr` (page 111). `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.

Return Value
A result code. See "Appearance Manager Result Codes" (page 217).

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 51) to draw the tab pane. The Appearance Manager coordinates the appearance of the pane and frontmost tab automatically.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Appearance.h`

DrawThemeTabPane

Draws a tab pane. (Deprecated in Mac OS X v10.5. Use `HIThemeDrawTabPane` instead.)

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

Parameters

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 Drawing States" (page 126). The tab pane can be drawn as active or inactive; passing `kThemeStatePressed` produces an error.

Return Value
A result code. See "Appearance Manager Result Codes" (page 217).

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 50) to draw the tab. The Appearance Manager coordinates the appearance of the pane and frontmost tab automatically.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Appearance.h`

DrawThemeTextBox

Draws text into the area you specify. (Deprecated in Mac OS X v10.5. Use `HIThemeDrawTextBox` instead.)

```
OSStatus DrawThemeTextBox (
   CFStringRef inString,
   ThemeFontID inFontID,
   ThemeDrawState inState,
   Boolean inWrapToWidth,
   const Rect *inBoundingBox,
   SInt16 inJust,
   CGContextRef inContext
);
```

Parameters

inString

A `CFStringRef` containing the unicode characters you wish to render. You must not pass in a `CFStringRef` that was allocated with any of the "NoCopy" CFString creation functions; a string created with a "NoCopy" function has transient storage which is incompatible with `DrawThemeTextBox`'s caches.

inFontID

The `ThemeFontID` describing the font you'd like to render the text with. See "Theme Font IDs" (page 166) for the values you can use here.

inState

The `ThemeDrawState` describing the state of the interface element you are drawing the text for. If, for example, you are drawing text for an inactive window, you would pass `kThemeStateInactive`. The `ThemeDrawState` is generally only used to determine the shadow characteristics for the text on Mac OS X.

See "Theme Drawing States" (page 126) for the values you can use here.

Note that the `ThemeDrawState` does not imply a color. It is not used as a mechanism for graying the text. If you wish to draw grayed text, you must set up the desired gray color and apply it to either the current QuickDraw port or the `CGContextRef`, as appropriate.

inWrapToWidth

A Boolean value indicating whether you want to draw multiple lines of text wrapped to a bounding box. `False` indicates that only one line of text should be drawn without any sort of wrapping.

inBoundingBox

The rectangle, in coordinates relative to the current QuickDraw port, describing the area to draw the text within. The first line of text will be top-justified to this rectangle. Wrapping, if desired, will happen at the horizontal extent of this rectangle. Regardless of the amount of text in your `CFStringRef`, all drawn text will be clipped to this rectangle.

inJust

The horizontal alignment you would like for your text. You can use one of the standard alignment constants from TextEdit.h.

inContext

The `CGContextRef` into which you would like to draw the text. On Mac OS X, all text drawing happens in `CGContextRef`s; if you pass `NULL`, a transient `CGContextRef` will be allocated and deallocated for use within the single function call. Relying on the system behavior if transiently creating `CGContextRef`s may result in performance problems. On Mac OS 9, this parameter is ignored.

Return Value

A result code. See "Appearance Manager Result Codes" (page 217).

Discussion
`DrawThemeTextBox` allows you to draw theme-savvy —that is, Aqua-savvy on Mac OS X— text. It is unicode savvy, although only partially so under CarbonLib, and allows you to customize certain text rendering characteristics such as the font, wrapping behavior, and justification. The text is drawn into the `CGContextRef` you provide, or into the current QuickDraw port if no `CGContextRef` is provided. None of `DrawThemeTextBox`'s parameters imply a color, so you must set up the desired text color separately before calling `DrawThemeTextBox`. If you provide a `CGContextRef`, its fill color will be used to draw the text. If you do not provide a `CGContextRef`, a color based on the current QuickDraw port's foreground color and the `grayishTextOr` mode, if set, will be used to draw the text.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Appearance.h`

DrawThemeTickMark

Draws a tick mark. (Deprecated in Mac OS X v10.5. Use `HIThemeDrawTickMark` instead.)

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

Parameters

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 Drawing States" (page 126). The tick mark can be drawn as active or inactive; passing `kThemeStatePressed` produces an error.

Return Value
A result code. See "Appearance Manager Result Codes" (page 217).

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 56).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Appearance.h`

DrawThemeTitleBarWidget

Draws a close box, zoom box, or collapse box. (Deprecated in Mac OS X v10.5. Use `HIThemeDrawTitleBarWidget` instead.)

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

Parameters

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 Types" (page 189) 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 Drawing States" (page 126) for descriptions of these values.

metrics

A pointer to a structure of type `ThemeWindowMetrics`. 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 Attributes" (page 191) 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 Items" (page 192) for descriptions of these values.

Return Value

A result code. See "Appearance Manager Result Codes" (page 217).

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.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
Appearance.h

DrawThemeTrack

Draws a track. (Deprecated in Mac OS X v10.5. Use HIThemeDrawTrack instead.)

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

Parameters

drawInfo

A pointer to a structure of type ThemeTrackDrawInfo. 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. If you have a custom background, pass a universal function pointer to an application-defined function such as that described in ThemeEraseProcPtr (page 108). 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.

Return Value
A result code. See "Appearance Manager Result Codes" (page 217).

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 45) to draw the scroll bar's arrows. If you use DrawThemeTrack to draw a slider, use DrawThemeTrackTickMarks (page 56) to draw any tick marks for the slider.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
Appearance.h

DrawThemeTrackTickMarks

Draws tick marks for a track. (Deprecated in Mac OS X v10.5. Use `HIThemeDrawTrackTickMarks` instead.)

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

Parameters

drawInfo

A pointer to a structure of type `ThemeTrackDrawInfo`. 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 76). `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`. If you have a custom background, pass a universal function pointer to an application-defined function such as that described in `ThemeEraseProcPtr` (page 108). `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.

Return Value

A result code. See "Appearance Manager Result Codes" (page 217).

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 55). To draw a single tick mark, call the function `DrawThemeTickMark` (page 53).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Appearance.h`

DrawThemeWindowFrame

Draws a window frame. (Deprecated in Mac OS X v10.5. Use `HIThemeDrawWindowFrame` instead.)

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

Parameters

flavor

A value of type `ThemeWindowType`. Pass a constant specifying the type of window for which to draw a frame. See "Theme Window Types" (page 189) 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 Drawing States" (page 126) for descriptions of these values.

metrics

A pointer to a structure of type `ThemeWindowMetrics`. 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 Attributes" (page 191) for descriptions of possible values. Pass 0 if the window has none of the enumerated attributes.

titleProc

A value of type `WindowTitleDrawingUPP`. If you pass the value `kThemeWindowHasTitleText` in the `attributes` parameter, you should pass a universal function pointer to an application-defined function such as that described in `WindowTitleDrawingProcPtr` (page 112) 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.

Return Value

A result code. See "Appearance Manager Result Codes" (page 217).

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.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`Appearance.h`

DrawThemeWindowHeader

Draws a window header. (Deprecated in Mac OS X v10.5. Use `HIThemeDrawHeader` instead.)

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

Parameters

`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 Drawing States" (page 126). The header can be drawn as active or inactive; passing `kThemeStatePressed` produces an error.

Return Value
A result code. See "Appearance Manager Result Codes" (page 217).

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

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Appearance.h`

DrawThemeWindowListViewHeader

Draws a window list view header. (Deprecated in Mac OS X v10.5. Use `HIThemeDrawHeader` instead.)

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

Parameters

`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 Drawing States" (page 126). The header can be drawn as active or inactive; passing `kThemeStatePressed` produces an error.

Return Value
A result code. See "Appearance Manager Result Codes" (page 217).

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.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Appearance.h`

EndThemeDragSound

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

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

Return Value
A result code. See "Appearance Manager Result Codes" (page 217).

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 27), 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.

Special Considerations
This function is not implemented in Mac OS X.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Appearance.h`

GetTheme

Obtains a collection containing data describing the current theme.

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

Parameters

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.

Return Value

A result code. See "Appearance Manager Result Codes" (page 217).

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 122) 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.

Also see the function `SetTheme` (page 97).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Appearance.h`

GetThemeAccentColors

Obtains a copy of a theme's accent colors. (Deprecated in Mac OS X v10.5. There is no replacement function.)

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

Parameters

outColors

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

Return Value

A result code. `GetThemeAccentColors` returns the result `appearanceThemeHasNoAccents` if the current theme has no accent colors.

Discussion

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

Special Considerations

In Mac OS X, theme accent colors are not supported.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Appearance.h`

GetThemeBrushAsColor

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

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

Parameters

inBrush
A value of type `ThemeBrush`. Pass a constant specifying the theme brush type for which you wish to obtain a color; see "Theme Brushes" (page 145) 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.

Return Value
A result code. See "Appearance Manager Result Codes" (page 217).

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 98) and `SetThemePen` (page 100) for greatest fidelity with the current theme.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Appearance.h`

GetThemeButtonBackgroundBounds

Obtains the rectangle that contains a button. (Deprecated in Mac OS X v10.5. Use `HIThemeGetButtonBackgroundBounds` instead.)

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

Parameters

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 Buttons" (page 153) for descriptions of possible values.

inDrawInfo

A pointer to a structure of type `ThemeButtonDrawInfo`. 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.

Return Value

A result code. See "Appearance Manager Result Codes" (page 217).

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.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Appearance.h`

GetThemeButtonContentBounds

Obtains the rectangle where content can be drawn for a button. (Deprecated in Mac OS X v10.5. Use `HIThemeGetButtonContentBounds` instead.)

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

Parameters

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 Buttons" (page 153) for descriptions of possible values.

inDrawInfo

A pointer to a structure of type `ThemeButtonDrawInfo`. 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.

Return Value

A result code. See "Appearance Manager Result Codes" (page 217).

Discussion

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

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Appearance.h`

GetThemeButtonRegion

Obtains the region occupied by a button. (Deprecated in Mac OS X v10.5. Use `HIThemeGetButtonShape` instead.)

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

Parameters

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 Buttons" (page 153) for descriptions of possible values.

inNewInfo

A pointer to a structure of type `ThemeButtonDrawInfo`. 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.

Return Value
A result code. See "Appearance Manager Result Codes" (page 217).

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.

Special Considerations

This function is available with Appearance Manager 1.1 and later.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`Appearance.h`

GetThemeCheckBoxStyle

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

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

Parameters

outStyle

A pointer to a value of type `ThemeCheckBoxStyle`. On return, the value specifies the type of mark being used. See "Theme Checkbox Styles" (page 161) for descriptions of possible values.

Return Value
A result code. See "Appearance Manager Result Codes" (page 217).

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.

Availability
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
`Appearance.h`

GetThemeDrawingState

Obtains the drawing state of the current graphics port.

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

Parameters

outState

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

Return Value
A result code. See "Appearance Manager Result Codes" (page 217).

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 99), providing the reference obtained in the `outState` parameter of `GetThemeDrawingState`. You can also call `DisposeThemeDrawingState` (page 29) to release the allocated memory.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Appearance.h`

GetThemeFont

Obtains information about a system font in the current theme. (Deprecated in Mac OS X v10.5. Some theme fonts cannot be drawn using QuickDraw; use `HIThemeDrawTextBox` instead.)

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

Parameters

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.

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.

Return Value
A result code. See "Appearance Manager Result Codes" (page 217).

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.

Also see the function `UseThemeFont` (page 103).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Appearance.h`

GetThemeMenuBackgroundRegion

Obtains the background region for a menu. (Deprecated in Mac OS X v10.5. Use `HIThemeGetMenuBackgroundShape` instead.)

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

Parameters

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 Types" (page 176) 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.

Return Value
A result code. See "Appearance Manager Result Codes" (page 217).

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

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Appearance.h`

GetThemeMenuBarHeight

Obtains the height of a menu bar.

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

Parameters

outHeight

A pointer to a signed 16-bit integer. On return, the integer value represents the height (in pixels) of the menu bar.

Return Value
A result code. See "Appearance Manager Result Codes" (page 217).

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 Events" (page 120) 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.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
Appearance.h

GetThemeMenuItemExtra

Obtains a measurement of the space surrounding a menu item.

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

Parameters

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 Types" (page 177).

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.

Return Value
A result code. See "Appearance Manager Result Codes" (page 217).

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.

Availability
Available in Mac OS X v10.0 and later.

Declared In
Appearance.h

GetThemeMenuSeparatorHeight

Obtains the height of a menu separator line.

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

Parameters

outHeight

A pointer to a signed 16-bit integer. On return, the integer value represents the height (in pixels) of the menu separator line.

Return Value
A result code. See "Appearance Manager Result Codes" (page 217).

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.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Appearance.h`

GetThemeMenuTitleExtra

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

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

Parameters

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.

Return Value
A result code. See "Appearance Manager Result Codes" (page 217).

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.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Appearance.h`

GetThemeMetric

Retrieves the value of a metric property of a user interface element.

```
OSStatus GetThemeMetric (
   ThemeMetric inMetric,
   SInt32 *outMetric
);
```

Parameters

inMetric

The user interface metric to retrieve. See "Theme Metrics" (page 128) for a list of possible metrics.

outMetric

A pointer to an integer value. On output, contains the value of the specified metric property, generally in points.

Return Value

A result code. See "Appearance Manager Result Codes" (page 217).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Appearance.h`

GetThemeScrollBarArrowStyle

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

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

Parameters

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 Styles" (page 179) for descriptions of possible values.

Return Value

A result code. See "Appearance Manager Result Codes" (page 217).

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.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Appearance.h`

GetThemeScrollBarThumbStyle

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

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

Parameters

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 Styles" (page 180) for descriptions of possible values.

Return Value

A result code. See "Appearance Manager Result Codes" (page 217).

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.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Appearance.h`

GetThemeScrollBarTrackRect

Obtains the area containing the track portion of a scroll bar. (Deprecated in Mac OS X v10.5. Use `HIThemeGetScrollBarTrackRect` instead.)

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

Parameters

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 States" (page 185) 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 States" (page 187) 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.

Return Value
A result code. See "Appearance Manager Result Codes" (page 217).

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.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`Appearance.h`

GetThemeStandaloneGrowBoxBounds

Obtains the bounds of a size box. (Deprecated in Mac OS X v10.5. Use `HIThemeGetGrowBoxBounds` instead.)

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

Parameters

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 Directions" (page 180) 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.

Return Value
A result code. See "Appearance Manager Result Codes" (page 217).

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 49).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Appearance.h`

GetThemeTabRegion

Obtains the region occupied by a tab. (Deprecated in Mac OS X v10.5. Use `HIThemeGetTabDrawShape` instead.)

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

Parameters

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 Styles" (page 183) 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 Directions" (page 182) 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.

Return Value
A result code. See "Appearance Manager Result Codes" (page 217).

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.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Appearance.h`

GetThemeTextColor

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

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

Parameters

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 Colors" (page 169) 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.

Return Value
A result code. See "Appearance Manager Result Codes" (page 217).

Discussion
Also see the function `SetThemeTextColor` (page 101).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Appearance.h`

GetThemeTextDimensions

Tells you the height, width, and baseline for a string. (Deprecated in Mac OS X v10.5. Use `HIThemeGetTextDimensions` instead.)

```
OSStatus GetThemeTextDimensions (
   CFStringRef inString,
   ThemeFontID inFontID,
   ThemeDrawState inState,
   Boolean inWrapToWidth,
   Point *ioBounds,
   SInt16 *outBaseline
);
```

Parameters

inString

A `CFStringRef` containing the unicode characters you wish to measure. You must not pass in a `CFStringRef` that was allocated with any of the "NoCopy" CFString creation functions, as mentioned in the description of the `DrawThemeTextBox` (page 51) function.

inFontID

The `ThemeFontID` describing the font you'd like to measure the text with. See "Theme Font IDs" (page 166) for the values you can use here.

inState

The `ThemeDrawState` which matches the state you will ultimately render the string with. Drawing state may affect text measurement, so you should be sure the value you pass to `GetThemeTextDimensions` matches the value you will eventually use for drawing. See "Theme Drawing States" (page 126) for the values you can use here.

inWrapToWidth

A Boolean indicating whether you want the measurements based on wrapping the text to a specific width. If you pass `true`, you must specify the desired width in `ioBounds->h`.

ioBounds

On output, `ioBounds->v` contains the height of the text. If you pass `false` in the `inWrapToWidth` parameter, `ioBounds->h` will contain the width of the text on output. If you pass `true` in `inWrapToWidth`, `ioBounds->h` must (on input) contain the desired width for wrapping; on output, `ioBounds->h` contains the same value you specified on input.

outBaseline

On output, `outBaseline` contains the offset (in QuickDraw space) from the bottom edge of the last line of text to the baseline of the first line of text. `outBaseline` will generally be a negative value.

Return Value

A result code. See "Appearance Manager Result Codes" (page 217).

Discussion

`GetThemeTextDimensions` measures the given string using the font and drawing state you specify. It always reports the actual height and baseline. It sometimes reports the actual width. It can measure a string that wraps. It is unicode savvy, although only partially so under CarbonLib.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Appearance.h`

GetThemeTextShadowOutset

Tells you the amount of space taken up by the shadow for a given font and drawing state combination.

```
OSStatus GetThemeTextShadowOutset (
   ThemeFontID inFontID,
   ThemeDrawState inState,
   Rect *outOutset
);
```

Parameters

inFontID

The `ThemeFontID` describing the font you'd like the shadow characteristics of. Font and drawing state both determine the amount of shadow that will be used on rendered text. See "Theme Font IDs" (page 166) for the values you can use here.

inState

The `ThemeDrawState` which matches the drawing state you'd like the shadow characteristics of. Font and state both determine the amount of shadow that will be used on rendered text. See "Theme Drawing States" (page 126) for the values you can use here.

outOutset

On output, this parameter contains the amount of space the shadow will take up beyond each edge of the text bounding rectangle returned by `GetThemeTextDimensions`. The fields of this parameter will either be positive values or zero.

Return Value

A result code. See "Appearance Manager Result Codes" (page 217).

Discussion

`GetThemeTextShadowOutset` passes back the maximum amount of space the shadow will take up for text drawn in the specified font and state. While `GetThemeTextDimensions` tells you how much space is taken up by the character glyphs themselves, it does not incorporate the font or state shadow into its calculations. If you need to know how much total space including the shadow will be taken up, call `GetThemeTextDimensions` followed by `GetThemeTextShadowOutset`.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Appearance.h`

GetThemeTrackBounds

Obtains the bounding rectangle of a track. (Deprecated in Mac OS X v10.5. Use `HIThemeGetTrackBounds` instead.)

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

Parameters

drawInfo

A pointer to a structure of type `ThemeTrackDrawInfo`. 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.

Return Value

A result code. See "Appearance Manager Result Codes" (page 217).

Discussion

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

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Appearance.h`

GetThemeTrackDragRect

Obtains the area in which the user may drag a track's indicator. (Deprecated in Mac OS X v10.5. Use `HIThemeGetTrackDragRect` instead.)

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

Parameters

drawInfo

A pointer to a structure of type `ThemeTrackDrawInfo`. 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.

Return Value
A result code. See "Appearance Manager Result Codes" (page 217).

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.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Appearance.h`

GetThemeTrackLiveValue

Obtains the current value of a track's indicator, given its relative position. (Deprecated in Mac OS X v10.5. Use `HIThemeGetTrackLiveValue` instead.)

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

Parameters

drawInfo
A pointer to a structure of type `ThemeTrackDrawInfo`. 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 79) or `GetThemeTrackThumbPositionFromRegion` (page 79).

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

Return Value
A result code. See "Appearance Manager Result Codes" (page 217).

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

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Appearance.h`

GetThemeTrackThumbPositionFromOffset

Obtains the relative position of a track's indicator, given an offset from its prior position. (Deprecated in Mac OS X v10.5. Use `HIThemeGetTrackThumbPositionFromOffset` instead.)

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

Parameters

drawInfo

A pointer to a structure of type `ThemeTrackDrawInfo`. 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.

Return Value

A result code. See "Appearance Manager Result Codes" (page 217).

Discussion

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

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Appearance.h`

GetThemeTrackThumbPositionFromRegion

Obtains the relative position of a track's indicator, given its current position. (Deprecated in Mac OS X v10.5. use `HIThemeGetTrackThumbPositionFromBounds` instead.)

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

Parameters

drawInfo

A pointer to a structure of type `ThemeTrackDrawInfo`. 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.

Return Value

A result code. See "Appearance Manager Result Codes" (page 217).

Discussion

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

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Appearance.h`

GetThemeTrackThumbRgn

Obtains the region containing a track's indicator. (Deprecated in Mac OS X v10.5. Use `HIThemeGetTrackThumbShape` instead.)

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

Parameters

drawInfo

A pointer to a structure of type `ThemeTrackDrawInfo`. 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.

Return Value

A result code. See "Appearance Manager Result Codes" (page 217).

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

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Appearance.h`

GetThemeWindowRegion

Obtains the specified window region. (Deprecated in Mac OS X v10.5. Use `HIThemeGetWindowShape` instead.)

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

Parameters

flavor
A value of type `ThemeWindowType`. Pass a constant specifying the type of window to be examined. See "Theme Window Types" (page 189) 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 Drawing States" (page 126) for descriptions of these values.

metrics
A pointer to a structure of type `ThemeWindowMetrics`. 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 Attributes" (page 191) 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.

Return Value
A result code. See "Appearance Manager Result Codes" (page 217).

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

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Appearance.h`

GetThemeWindowRegionHit

Obtains the part of the window that the user clicked upon. (Deprecated in Mac OS X v10.5. Use `HIThemeGetWindowRegionHit` instead.)

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

Parameters

flavor
A value of type `ThemeWindowType`. Pass a constant specifying the type of window to be examined. See "Theme Window Types" (page 189) 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 Drawing States" (page 126) for descriptions of these values.

metrics
A pointer to a structure of type `ThemeWindowMetrics`. 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 Attributes" (page 191) 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.

Return Value
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.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`Appearance.h`

HitTestThemeScrollBarArrows

Returns whether the user clicked upon the specified scroll bar's arrows. (Deprecated in Mac OS X v10.5. Use `HIThemeHitTestScrollBarArrows` instead.)

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

Parameters

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 States" (page 185) 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 States" (page 187) 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.

Return Value

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 `HitTestThemeScrollBarArrow` 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 84), in order to determine whether the mouse-down event occurred in the track part of the scroll bar.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Appearance.h`

HitTestThemeTrack

Returns whether the user clicked upon the specified track. (Deprecated in Mac OS X v10.5. Use `HIThemeHitTestTrack` instead.)

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

Parameters

drawInfo

A pointer to a structure of type `ThemeTrackDrawInfo`. 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.

Return Value
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 83) 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.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Appearance.h`

InvokeMenuItemDrawingUPP

Invokes your menu item drawing function. (Deprecated in Mac OS X v10.5. There is no replacement function.)

```
void InvokeMenuItemDrawingUPP (
   const Rect *inBounds,
   SInt16 inDepth,
   Boolean inIsColorDevice,
   SRefCon inUserData,
   MenuItemDrawingUPP userUPP
);
```

Discussion
You should not need to use the function `InvokeMenuItemDrawingUPP`, as the system calls your menu item drawing function for you.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`Appearance.h`

InvokeMenuTitleDrawingUPP

Invokes your menu title drawing function. (Deprecated in Mac OS X v10.5. There is no replacement function.)

```
void InvokeMenuTitleDrawingUPP (
   const Rect *inBounds,
   SInt16 inDepth,
   Boolean inIsColorDevice,
   SRefCon inUserData,
   MenuTitleDrawingUPP userUPP
);
```

Discussion
You should not need to use the function `InvokeMenuTitleDrawingUPP`, as the system calls your menu title drawing function for you.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`Appearance.h`

InvokeThemeButtonDrawUPP

Invokes your button drawing function. (Deprecated in Mac OS X v10.5. There is no replacement function.)

```
void InvokeThemeButtonDrawUPP (
   const Rect *bounds,
   ThemeButtonKind kind,
   const ThemeButtonDrawInfo *info,
   URefCon userData,
   SInt16 depth,
   Boolean isColorDev,
   ThemeButtonDrawUPP userUPP
);
```

Discussion
You should not need to use the function `InvokeThemeButtonDrawUPP`, as the system calls your button drawing function for you.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`Appearance.h`

InvokeThemeEraseUPP

Invokes your background drawing callback function. (Deprecated in Mac OS X v10.5. There is no replacement function.)

```
void InvokeThemeEraseUPP (
   const Rect *bounds,
   URefCon eraseData,
   SInt16 depth,
   Boolean isColorDev,
   ThemeEraseUPP userUPP
);
```

Discussion
You should not need to use the function `InvokeThemeEraseUPP`, as the system calls your `ThemeEraseProcPtr` (page 108) callback function for you.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`Appearance.h`

InvokeThemeIteratorUPP

Invokes your theme iteration callback function. (Deprecated in Mac OS X v10.5. There is no replacement function.)

```
Boolean InvokeThemeIteratorUPP (
   ConstStr255Param inFileName,
   SInt16 resID,
   Collection inThemeSettings,
   PRefCon inUserData,
   ThemeIteratorUPP userUPP
);
```

Discussion
You should not need to use the function `InvokeThemeIteratorUPP`, as the system calls your `ThemeIteratorProcPtr` (page 109) callback function for you.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`Appearance.h`

InvokeThemeTabTitleDrawUPP

Invokes your tab title drawing function. (Deprecated in Mac OS X v10.5. There is no replacement function.)

```
void InvokeThemeTabTitleDrawUPP (
   const Rect *bounds,
   ThemeTabStyle style,
   ThemeTabDirection direction,
   SInt16 depth,
   Boolean isColorDev,
   URefCon userData,
   ThemeTabTitleDrawUPP userUPP
);
```

Discussion
You should not need to use the function `InvokeThemeTabTitleDrawUPP`, as the system calls your tab title drawing function for you.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`Appearance.h`

InvokeWindowTitleDrawingUPP

Invokes your window title drawing function. (Deprecated in Mac OS X v10.5. There is no replacement function.)

```
void InvokeWindowTitleDrawingUPP (
   const Rect *bounds,
   SInt16 depth,
   Boolean colorDevice,
   URefCon userData,
   WindowTitleDrawingUPP userUPP
);
```

Discussion
You should not need to use the function `InvokeWindowTitleDrawingUPP`, as the system calls your window title drawing function for you.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`Appearance.h`

IsAppearanceClient

Returns whether a given process is currently registered as a client of the Appearance Manager. (Deprecated in Mac OS X v10.5. There is no replacement function.)

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

Parameters

process

A pointer to a value of type `ProcessSerialNumber`. Pass the serial number of the process to examine.

Return Value

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 95).

Special Considerations

This function always returns `true` in Mac OS X.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Appearance.h`

IsThemeInColor

Returns whether the current theme would draw in color in the given environment. (Deprecated in Mac OS X v10.5. There is no replacement function.)

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

Parameters

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.

Return Value

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.

Special Considerations

In Mac OS X, this function always returns true because the Aqua theme is always drawn in color.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`Appearance.h`

IsValidAppearanceFileType

Returns whether the system can interpret files of a given file type as appearance files. (Deprecated in Mac OS X v10.5. There is no replacement function.)

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

Parameters

`fileType`

A four-character code. Pass the file type to be examined.

Return Value
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 Types" (page 121), are valid appearance file types. Your application typically does not need to call this function.

Special Considerations

This function always returns false in Mac OS X.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`Appearance.h`

IterateThemes

Iterates over all themes installed on a system. (Deprecated in Mac OS X v10.5. There is no replacement function.)

```
OSStatus IterateThemes (
   ThemeIteratorUPP inProc,
   PRefCon inUserData
);
```

Parameters

inProc

A universal function pointer to an application-defined function such as that described in `ThemeIteratorProcPtr` (page 109). `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.

Return Value

A result code. See "Appearance Manager Result Codes" (page 217).

Discussion

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

Special Considerations

This function does nothing in Mac OS X; it does not call the theme iterator callback function.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Appearance.h`

NewMenuItemDrawingUPP

Creates a new universal procedure pointer (UPP) to a menu item drawing function. (Deprecated in Mac OS X v10.5. There is no replacement function.)

```
MenuItemDrawingUPP NewMenuItemDrawingUPP (
   MenuItemDrawingProcPtr userRoutine
);
```

Return Value

A UPP. See `MenuItemDrawingProcPtr` (page 104) for information on the menu item drawing function. See the description of the `MenuItemDrawingUPP` data type.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In
`Appearance.h`

NewMenuTitleDrawingUPP

Creates a new universal procedure pointer (UPP) to a menu title drawing function. (Deprecated in Mac OS X v10.5. There is no replacement function.)

```
MenuTitleDrawingUPP NewMenuTitleDrawingUPP (
   MenuTitleDrawingProcPtr userRoutine
);
```

Return Value
A UPP. See `MenuTitleDrawingProcPtr` (page 105) for information on the menu title drawing function. See the description of the `MenuTitleDrawingUPP` data type.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`Appearance.h`

NewThemeButtonDrawUPP

Creates a new universal procedure pointer (UPP) to a button drawing function. (Deprecated in Mac OS X v10.5. There is no replacement function.)

```
ThemeButtonDrawUPP NewThemeButtonDrawUPP (
   ThemeButtonDrawProcPtr userRoutine
);
```

Return Value
A UPP. See `ThemeButtonDrawProcPtr` (page 107) for information on the button drawing function. See the description of the `ThemeButtonDrawUPP` data type.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`Appearance.h`

NewThemeEraseUPP

Creates a new universal procedure pointer (UPP) to a background drawing callback function. (Deprecated in Mac OS X v10.5. There is no replacement function.)

```
ThemeEraseUPP NewThemeEraseUPP (
   ThemeEraseProcPtr userRoutine
);
```

Return Value
A UPP. See the description of the `ThemeEraseUPP` data type.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`Appearance.h`

NewThemeIteratorUPP

Creates a new universal procedure pointer (UPP) to a theme iteration callback function. (Deprecated in Mac OS X v10.5. There is no replacement function.)

```
ThemeIteratorUPP NewThemeIteratorUPP (
   ThemeIteratorProcPtr userRoutine
);
```

Return Value
A UPP. See the description of the `ThemeIteratorUPP` data type.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`Appearance.h`

NewThemeTabTitleDrawUPP

Creates a new universal procedure pointer (UPP) to a tab title drawing function. (Deprecated in Mac OS X v10.5. There is no replacement function.)

```
ThemeTabTitleDrawUPP NewThemeTabTitleDrawUPP (
   ThemeTabTitleDrawProcPtr userRoutine
);
```

Return Value
A UPP. See `ThemeTabTitleDrawProcPtr` (page 111) for information on the tab title drawing function. See the description of the `ThemeTabTitleDrawUPP` data type.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`Appearance.h`

NewWindowTitleDrawingUPP

Creates a new universal procedure pointer (UPP) to a window title drawing function. (Deprecated in Mac OS X v10.5. There is no replacement function.)

```
WindowTitleDrawingUPP NewWindowTitleDrawingUPP (
   WindowTitleDrawingProcPtr userRoutine
);
```

Return Value
A UPP. See `WindowTitleDrawingProcPtr` (page 112) for information on the window title drawing function. See the description of the `WindowTitleDrawingUPP` data type.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`Appearance.h`

NormalizeThemeDrawingState

Sets the current graphics port to a default drawing state.

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

Return Value
A result code. See "Appearance Manager Result Codes" (page 217).

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.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Appearance.h`

PlayThemeSound

Plays an asynchronous sound associated with the specified state change.

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

Parameters

kind

A value of type `ThemeSoundKind`. Pass a constant specifying the sound to play; see "Theme Sounds" (page 194) for descriptions of possible values.

Return Value

A result code. See "Appearance Manager Result Codes" (page 217).

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.

Special Considerations

This function is not implemented in Mac OS X.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Appearance.h`

RegisterAppearanceClient

Registers your program with the Appearance Manager. (Deprecated in Mac OS X v10.5. There is no replacement function.)

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

Return Value

A result code. The result code `appearanceProcessRegisteredErr` indicates that your program was already registered when you called the `RegisterAppearanceClient` function.

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 Events" (page 120) 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 also the function `UnregisterAppearanceClient` (page 103).

Special Considerations

This function does nothing in Mac OS X.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Appearance.h`

SetAnimatedThemeCursor

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

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

Parameters

inCursor

A value of type `ThemeCursor`. Pass a constant specifying the type of cursor to set; see "Theme Cursors" (page 162) 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`.

Return Value

A result code. See "Appearance Manager Result Codes" (page 217).

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 98).

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.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Appearance.h`

SetTheme

Sets a specified collection as the current theme. (Deprecated in Mac OS X v10.5. There is no replacement function.)

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

Parameters

`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 122) for descriptions of the possible theme collection items.

Return Value

A result code. See "Appearance Manager Result Codes" (page 217).

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.

Your application can use the `GetTheme` (page 59) function to obtain a collection containing a copy of the data for the current theme.

Special Considerations

This function does not modify the current theme in Mac OS X.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Appearance.h`

SetThemeBackground

Applies a theme-compliant color or pattern to the background of the current port. (Deprecated in Mac OS X v10.5. Use `HIThemeSetFill` and draw using Quartz 2D.)

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

Parameters

inBrush

A value of type `ThemeBrush`. Pass a constant specifying the theme brush to which to set the background; see "Theme Brushes" (page 145) 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.

Return Value

A result code. The result code `appearanceBadBrushIndexErr` indicates that the brush constant passed was not valid.

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 26).

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 65) and `SetThemeDrawingState` (page 99) 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.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Appearance.h`

SetThemeCursor

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

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

Parameters

inCursor

A value of type `ThemeCursor`. Pass a constant specifying the type of cursor to set; see "Theme Cursors" (page 162) for a description of possible values.

Return Value

A result code. See "Appearance Manager Result Codes" (page 217).

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 96).

Special Considerations

Do not call `SetThemeCursor` at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

CarbonCocoa_PictureCursor

CarbonSketch

QTCarbonShell

Declared In

`Appearance.h`

SetThemeDrawingState

Sets the drawing state of the current graphics port.

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

Parameters

inState

A value of type `ThemeDrawingState`. Pass a `ThemeDrawingState` value such as that produced in the `outState` parameter of `GetThemeDrawingState` (page 65).

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 29) to dispose of the memory any time before your application terminates.

Return Value
A result code. See "Appearance Manager Result Codes" (page 217).

Discussion
Your application can save the port state by calling the function `GetThemeDrawingState` (page 65) 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.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Appearance.h`

SetThemePen

Applies a theme-compliant color or pattern to the foreground of the current port. (Deprecated in Mac OS X v10.5. Use `HIThemeSetStroke` and draw using Quartz 2D.)

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

Parameters

inBrush

A value of type `ThemeBrush`. Pass a constant specifying the theme brush type to which to set the pen; see "Theme Brushes" (page 145) 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.

Return Value
A result code. The result code `appearanceBadBrushIndexErr` indicates that the brush constant passed in was not valid.

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 65) and SetThemeDrawingState (page 99) 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.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
Appearance.h

SetThemeTextColor

Sets the current text color to be consistent with that of a specified element. (Deprecated in Mac OS X v10.5. Use HIThemeSetTextFill and draw with Quartz 2D, ATSUI, or HIThemeDrawTextBox.)

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

Parameters

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 Colors" (page 169) 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.

Return Value
A result code. The result code appearanceBadTextColorIndexErr indicates that the text color index passed was not valid.

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

Also see the function GetThemeTextColor (page 74).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Appearance.h`

TruncateThemeText

Truncates text to fit within the width you specify. (Deprecated in Mac OS X v10.5. There is no replacement function; use `HIThemeGetTextDimensions` or `HIThemeDrawTextBox` instead.)

```
OSStatus TruncateThemeText (
   CFMutableStringRef inString,
   ThemeFontID inFontID,
   ThemeDrawState inState,
   SInt16 inPixelWidthLimit,
   TruncCode inTruncWhere,
   Boolean *outTruncated
);
```

Parameters

inString

A `CFMutableStringRef` containing the unicode characters you wish to truncate. On output, this string may have been altered to fit within the specified width.

You must not pass in a CFString that was allocated with any of the "NoCopy" CFString creation functions, as mentioned in the description of the `DrawThemeTextBox` (page 51) function.

inFontID

The `ThemeFontID` to use for text measurements. See "Theme Font IDs" (page 166) for the values you can use here.

inState

The `ThemeDrawState` which matches the state you will ultimately render the string with. This may affect text measurement during truncation, so you should be sure the value you pass to `TruncateThemeText` matches the value you will eventually use for drawing. See "Theme Drawing States" (page 126) for the values you can use here.

inPixelWidthLimit

The maximum width, in pixels, that the resulting truncated string may have.

inTruncWhere

A `TruncCode` indicating where you would like truncation to occur.

outTruncated

On output, this Boolean value indicates whether the string was truncated. `True` means the string was truncated. `False` means the string was not—and did not need to be—truncated.

Return Value
A result code. See "Appearance Manager Result Codes" (page 217).

Discussion
`TruncateThemeText` alters a unicode string to fit within a width that you specify. It is unicode savvy, although only partially so under CarbonLib, and makes its calculations—and any subsequent string alteration—based on the font and drawing state you specify. If the string needs to be truncated, it will be reduced to the maximum number of characters which, with the addition of an ellipsis character, fits within the specified width.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`Appearance.h`

UnregisterAppearanceClient

Informs the Appearance Manager that your program is no longer its client. (Deprecated in Mac OS X v10.5. There is no replacement function.)

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

Return Value
A result code. The result code `appearanceProcessNotRegisteredErr` indicates that your program was not registered when you called the `UnregisterAppearanceClient` function.

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.

See also the function `RegisterAppearanceClient` (page 95).

Special Considerations

This function does nothing in Mac OS X.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Appearance.h`

UseThemeFont

Sets the font of the current graphics port to one of the current theme's system fonts. (Deprecated in Mac OS X v10.5. Use `HIThemeDrawTextBox` instead.)

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

Parameters

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 IDs" (page 166) 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.

Return Value

A result code. See "Appearance Manager Result Codes" (page 217).

Discussion

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

Also see the function `GetThemeFont` (page 65).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Appearance.h`

Callbacks

MenuItemDrawingProcPtr

Draws a menu item.

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

If you name your function `MyMenuItemDrawingCallback`, you would declare it like this:

```
void MyMenuItemDrawingCallback (
    const Rect * inBounds,
    SInt16 inDepth,
    Boolean inIsColorDevice,
    SInt32 inUserData
```

```
);
```

Parameters

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 38).

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 `MyMenuItemDrawingCallback` function for every device that the `inBounds` rectangle intersects.

You should refer to your `MyMenuItemDrawingCallback` function using a `MenuItemDrawingUPP`, which you can create with `NewMenuItemDrawingUPP`.

You typically use the `NewMenuItemDrawingUPP` function like this:

```
MenuItemDrawingUPP myMenuItemDrawingUPP;
myMenuItemDrawingUPP = NewMenuItemDrawingUPP(MyMenuItemDrawingCallback);
```

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

This function is available with Appearance Manager 1.0.1 and later.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Appearance.h`

MenuTitleDrawingProcPtr

Draws a menu title.

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

If you name your function `MyMenuTitleDrawingCallback`, you would declare it like this:

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

Parameters

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 40).

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 `MyMenuTitleDrawingCallback` function for every device that the `inBounds` rectangle intersects.

You should refer to your `MyMenuTitleDrawingCallback` function using a `MenuTitleDrawingUPP`, which you can create with the `NewMenuTitleDrawingUPP` function.

You typically use the `NewMenuTitleDrawingUPP` function like this:

```
MenuTitleDrawingUPP myMenuTitleDrawingUPP;
myMenuTitleDrawingUPP = NewMenuTitleDrawingUPP(MyMenuTitleDrawingCallback);
```

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
This function is available with Appearance Manager 1.0.1 and later.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Appearance.h`

ThemeButtonDrawProcPtr

Draws a button label.

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

If you name your function `MyThemeButtonDrawCallback`, you would declare it like this:

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

Parameters

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 Buttons" (page 153) for descriptions of possible values.

info
A pointer to a structure of type `ThemeButtonDrawInfo` (page 115). 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 31).

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 `MyThemeButtonDrawCallback` function for every device that the `bounds` rectangle intersects.

You should refer to your `MyThemeButtonDrawCallback` function using a `ThemeButtonDrawUPP`, which you can create with the `NewThemeButtonDrawUPP` function.

You typically use the `NewThemeButtonDrawUPP` function like this:

```
ThemeButtonDrawUPP myThemeButtonDrawUPP;
myThemeButtonDrawUPP = NewThemeButtonDrawUPP(MyThemeButtonDrawCallback);
```

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

This function is available with Appearance Manager 1.1 and later.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Appearance.h`

ThemeEraseProcPtr

Draws a background.

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

If you name your function `MyThemeEraseCallback`, you would declare it like this:

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

Parameters

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 33), `DrawThemePopupArrow` (page 43), `DrawThemeTrack` (page 55), or `DrawThemeTrackTickMarks` (page 56) or from the `inUserData` parameter of `DrawThemeButton` (page 31).

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 `MyThemeEraseCallback` function for every device that the `bounds` rectangle intersects, so your application does not need to call the `DeviceLoop` function itself.

You should refer to your `MyThemeEraseCallback` function using a `ThemeEraseUPP`, which you can create with the `NewThemeEraseUPP` function.

You typically use the `NewThemeEraseUPP` function like this:

```
ThemeEraseUPP myThemeEraseUPP;
myThemeEraseUPP = NewThemeEraseUPP(MyThemeEraseCallback);
```

Version Notes

This function is available with Appearance Manager 1.1 and later.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Appearance.h`

ThemeIteratorProcPtr

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

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

If you name your function `MyThemeIteratorCallback`, you would declare it like this:

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

Parameters

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 91).

Return Value

A value of type `Boolean`. If you return `true`, `IterateThemes` continues iterating. Set to `false` to terminate the iteration.

Discussion

You should refer to your `MyThemeIteratorCallback` function using a `ThemeIteratorUPP`, which you can create using the `NewThemeIteratorUPP` function.

You typically use the `NewThemeIteratorUPP` function like this:

```
ThemeIteratorUPP myThemeIteratorUPP;
myThemeIteratorUPP = NewThemeIteratorUPP(MyThemeIteratorCallback);
```

Special Considerations

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

Version Notes

This function is available with Appearance Manager 1.1 and later.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`Appearance.h`

ThemeTabTitleDrawProcPtr

Draws a tab title.

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

If you name your function `MyThemeTabTitleDrawCallback`, you would declare it like this:

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

Parameters

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 Styles" (page 183) 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 Directions" (page 182) 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 50).

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 `MyThemeTabTitleDrawCallback` function for every device that the `bounds` rectangle intersects.

You should refer to your `MyThemeTabTitleDrawCallback` function using a `ThemeTabTitleDrawUPP`, which you can create with the `NewThemeTabTitleDrawUPP` function.

You typically use the `NewThemeTabTitleDrawUPP` function like this:

```
ThemeTabTitleDrawUPP myThemeTabTitleDrawUPP;
myThemeTabTitleDrawUPP = NewThemeTabTitleDrawUPP(MyThemeTabTitleDrawCallback);
```

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
This function is available with Appearance Manager 1.1 and later.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Appearance.h`

WindowTitleDrawingProcPtr

Draws a window title.

```
typedef void (*WindowTitleDrawingProcPtr)
(
    const Rect * bounds,
    SInt16 depth,
    Boolean colorDevice,
    UInt32 userData
);
```

If you name your function `MyWindowTitleDrawingCallback`, you would declare it like this:

```
void MyWindowTitleDrawingCallback (
    const Rect * bounds,
    SInt16 depth,
    Boolean colorDevice,
    UInt32 userData
);
```

Parameters

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 57).

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 `MyWindowTitleDrawingCallback` function for every device that the `bounds` rectangle intersects.

You should refer to your `MyWindowTitleDrawingCallback` function using a `WindowTitleDrawingUPP`, which you can create with the `NewWindowTitleDrawingUPP` function.

You typically use the `NewWindowTitleDrawingUPP` function like this:

```
WindowTitleDrawingUPP myWindowTitleDrawingUPP;
myWindowTitleDrawingUPP = NewWindowTitleDrawingUPP(MyWindowTitleDrawingCallback);
```

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

This function is available with Appearance Manager 1.1 and later.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Appearance.h`

Data Types

ProgressTrackInfo

Describes the progress bar–specific features of a given track control.

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

Fields

`phase`

A value specifying the current animation phase for an indeterminate progress bar. You can pass any value of type `UInt8`. Increment this value to animate the progress bar. Set this field to 0 for a determinate progress bar.

Discussion
Your application supplies a `ProgressTrackInfo` structure in the `ThemeTrackDrawInfo` (page 116) structure.

Version Notes
The `ProgressTrackInfo` structure is available with Appearance Manager 1.1 and later.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Appearance.h`

ScrollBarTrackInfo

Describes the scroll bar–specific features of a given track control.

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

Fields

`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 States" (page 187) for descriptions of possible values. Pass 0 if nothing is currently pressed.

Discussion
Your application uses the `ScrollBarTrackInfo` structure in the `ThemeTrackDrawInfo` (page 116) structure.

Version Notes
The `ScrollBarTrackInfo` structure is available with Appearance Manager 1.1 and later.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Appearance.h`

SliderTrackInfo

Describes the slider-specific features of a given track control.

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

Fields

`thumbDir`

A value of type `ThemeThumbDirection`, specifying the direction in which the slider indicator points. See "Theme Thumb Directions" (page 181) 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 States" (page 187) for descriptions of possible values. Pass 0 if nothing is currently pressed.

Discussion

Your application supplies a `SliderTrackInfo` structure to the `ThemeTrackDrawInfo` (page 116) structure.

Version Notes

The `SliderTrackInfo` structure is available with Appearance Manager 1.1 and later.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Appearance.h`

ThemeButtonDrawInfo

Describes the changeable visual characteristics of a button.

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

Fields

`state`

A value of type `ThemeDrawState`, specifying the state of the button, such as whether it is active, inactive, or pressed. See "Theme Drawing States" (page 126) 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 Values" (page 159) 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 Adornments" (page 156) for descriptions of possible values.

Discussion
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 Buttons" (page 153) for a description of `ThemeButtonKind` values.

Your application uses the `ThemeButtonDrawInfo` structure in the function `DrawThemeButton` (page 31) to draw a theme-compliant button and in the functions `GetThemeButtonRegion` (page 63) and `GetThemeButtonContentBounds` (page 62) to obtain information about a specific button type.

Version Notes
The `ThemeButtonDrawInfo` structure is available with Appearance Manager 1.1 and later.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Appearance.h`

ThemeTrackDrawInfo

Describes a track control.

```
struct ThemeTrackDrawInfo {
    ThemeTrackKind kind
    Rect bounds
    SInt32 min
    SInt32 max
    SInt32 value
    UInt32 reserved
    ThemeTrackAttributes attributes
    ThemeTrackEnableState enableState
    UInt8 filler1
    union {
        ScrollBarTrackInfo scrollbar;
        SliderTrackInfo slider;
        ProgressTrackInfo progress;
    } trackInfo;
};
typedef struct ThemeTrackDrawInfo ThemeTrackDrawInfo;
```

Fields

`kind`
A value of type `ThemeTrackKind`, specifying the type of track to be drawn. See "Theme Track Kinds" (page 186) 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" (page 184) for descriptions of possible values.

`enableState`
A value of type `ThemeTrackEnableState` specifying the current state of the track control; see "Theme Track States" (page 185) for descriptions of possible values.

`filler1`

`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 114), `SliderTrackInfo` (page 114), and `ProgressTrackInfo` (page 113) for details on these structures.

Discussion
Your application fills out the applicable fields of a `ThemeTrackDrawInfo` structure to fully describe any given track control.

Version Notes
The `ThemeTrackDrawInfo` structure is available with Appearance Manager 1.1 and later.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Appearance.h`

ThemeWindowMetrics

Describes the dimensions of the parts of a window.

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

Fields
`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. See "Pop-up Window Tab Positions" (page 193) for the values you can use in this field.

Discussion
Your application uses the `ThemeWindowMetrics` structure to inform the Appearance Manager of the dimensions of specific parts of your window.

Version Notes
The `ThemeWindowMetrics` structure is available with Appearance Manager 1.1 and later.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Appearance.h`

ThemeDrawingState

Defines a reference to a private structure containing information about the current state of a graphics port.

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

Discussion
You can use the `ThemeDrawingState` type with the function `GetThemeDrawingState` (page 65) to obtain the current graphics port's drawing state and with the function `SetThemeDrawingState` (page 99) to restore a port's drawing state. You should dispose of the memory allocated to contain a `ThemeDrawingState` reference by calling `DisposeThemeDrawingState` (page 29) or passing a value of `true` in the `inDisposeNow` parameter of `SetThemeDrawingState`.

Version Notes
The `ThemeDrawingState` type is available with Appearance Manager 1.1 and later.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Appearance.h`

MenuItemDrawingUPP

Defines a universal procedure pointer (UPP) to a menu item drawing function.

```
typedef MenuItemDrawingProcPtr MenuItemDrawingUPP;
```

Discussion
See `MenuItemDrawingProcPtr` (page 104) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Appearance.h`

MenuTitleDrawingUPP

Defines a universal procedure pointer (UPP) to a menu title drawing function.

```
typedef MenuTitleDrawingProcPtr MenuTitleDrawingUPP;
```

Discussion
See `MenuTitleDrawingProcPtr` (page 105) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Appearance.h`

ThemeButtonDrawUPP

Defines a universal procedure pointer (UPP) to a button drawing function.

```
typedef ThemeButtonDrawProcPtr ThemeButtonDrawUPP;
```

Discussion
See `ThemeButtonDrawProcPtr` (page 107) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Appearance.h`

ThemeEraseUPP

Defines a universal procedure pointer (UPP) to a background drawing callback function.

```
typedef ThemeEraseProcPtr ThemeEraseUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Appearance.h`

ThemeIteratorUPP

Defines a universal procedure pointer (UPP) to a theme iteration callback function.

```
typedef ThemeIteratorProcPtr ThemeIteratorUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Appearance.h`

ThemeTabTitleDrawUPP

Defines a universal procedure pointer (UPP) to a tab title drawing function.

```
typedef ThemeTabTitleDrawProcPtr ThemeTabTitleDrawUPP;
```

Discussion
See `ThemeTabTitleDrawProcPtr` (page 111) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Appearance.h`

WindowTitleDrawingUPP

Defines a universal procedure pointer (UPP) to a window title drawing function.

```
typedef WindowTitleDrawingProcPtr WindowTitleDrawingUPP;
```

Discussion
See `WindowTitleDrawingProcPtr` (page 112) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Appearance.h`

Constants

Appearance Manager Apple Events

Identify Apple events sent to Appearance Manager clients when a change occurs in the current appearance.

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

Constants

`kAppearanceEventClass`
The event class of Appearance Manager Apple events.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kAEAppearanceChanged`
The ID of the event indicating the current appearance has changed.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kAESystemFontChanged`
The ID of the event indicating the current system font has changed.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kAESmallSystemFontChanged`
The ID of the event indicating the current small system font has changed.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kAEViewsFontChanged`
The ID of the event indicating the current views font has changed.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

Discussion

When the user changes the current appearance (that is, when a theme switch occurs), the Appearance Manager may send any of these 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 95).

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.

Appearance Manager File Types

Identify the various Appearance Manager file types.

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

Constants

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

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemePlatinumFileType`
The file type of the platinum appearance.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeCustomThemesFileType`
The file type of a file that contains user-defined themes. See `SetTheme` (page 97) for a discussion of defining your own theme.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

Theme Collection Tags

Identify items in a collection describing a theme.

```
enum {
    kThemeNameTag = 'name',
    kThemeVariantNameTag = 'varn',
    kThemeVariantBaseTintTag = 'tint',
    kThemeHighlightColorTag = 'hcol',
    kThemeScrollBarArrowStyleTag = 'sbar',
    kThemeScrollBarThumbStyleTag = 'sbth',
    kThemeSoundsEnabledTag = 'snds',
    kThemeDblClickCollapseTag = 'coll'
    kThemeAppearanceFileNameTag = 'thme',
    kThemeSystemFontTag = 'lgsf',
    kThemeSmallSystemFontTag = 'smsf',
    kThemeViewsFontTag = 'vfnt',
    kThemeViewsFontSizeTag = 'vfsz',
    kThemeDesktopPatternNameTag = 'patn',
    kThemeDesktopPatternTag = 'patt',
    kThemeDesktopPictureNameTag = 'dpnm',
    kThemeDesktopPictureAliasTag = 'dpal',
    kThemeDesktopPictureAlignmentTag = 'dpan',
    kThemeHighlightColorNameTag = 'hcnm',
    kThemeExamplePictureIDTag = 'epic',
    kThemeSoundTrackNameTag = 'sndt',
    kThemeSoundMaskTag = 'smsk',
    kThemeUserDefinedTag = 'user',
    kThemeSmoothFontEnabledTag = 'smoo',
    kThemeSmoothFontMinSizeTag = 'smos'
};
```

Constants

`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`

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeVariantNameTag`

Identifies a collection item containing the color variation used for menus and controls in the theme.

Collection data type: `Str255`

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeHighlightColorTag`

Identifies a collection item containing the text highlight color for the theme.

Collection data type: an `RGBColor` structure

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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`

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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`

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundsEnabledTag`
Identifies a collection item specifying whether theme sounds are enabled for the theme.

Collection data type: `Boolean`

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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`

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeAppearanceFileNameTag`
Identifies a collection item containing the name of the appearance, e.g. "Apple platinum".

Collection data type: `Str255`

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSystemFontTag`
Identifies a collection item containing the name of the large system font for the theme.

Collection data type: `Str255`

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSmallSystemFontTag`
Identifies a collection item containing the name of the small system font for the theme.

Collection data type: `Str255`

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeViewsFontTag`
Identifies a collection item containing the name of the views font for the theme.

Collection data type: `Str255`

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeViewsFontSizeTag`
Identifies a collection item containing the size of the views font for the theme.
Collection data type: `SInt16`
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeDesktopPatternNameTag`
Identifies a collection item containing the name of the desktop pattern for the theme.
Collection data type: `Str255`
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeDesktopPatternTag`
Identifies a collection item containing a flattened version of the desktop pattern for the theme.
Collection data type: variable-length data
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeDesktopPictureNameTag`
Identifies a collection item containing the name of the desktop picture for the theme.
Collection data type: `Str255`
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeDesktopPictureAliasTag`
Identifies a collection item containing an alias handle for the desktop picture for the theme.
Collection data type: `AliasHandle`
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeDesktopPictureAlignmentTag`
Identifies a collection item containing a value specifying how to position the desktop picture for the theme. Possible values are described in "Desktop Picture Alignments" (page 212).
Collection data type: `UInt32`
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeHighlightColorNameTag`
Identifies a collection item containing the name of the text highlight color for the theme.
Collection data type: `Str255`
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeExamplePictureIDTag`
Identifies a collection item containing the ID of the example picture for the theme.
Collection data type: `SInt16`
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeSoundTrackNameTag`
: Identifies a collection item containing the name of the soundtrack for the theme.

  Collection data type: `Str255`

  Available in Mac OS X v10.0 and later.

  Declared in `Appearance.h`.

`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 Masks" (page 193) for descriptions of possible sound mask values.

  Collection data type: `UInt32`

  Available in Mac OS X v10.0 and later.

  Declared in `Appearance.h`.

`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`

  Available in Mac OS X v10.0 and later.

  Declared in `Appearance.h`.

`kThemeSmoothFontEnabledTag`
: Identifies a collection item specifying whether font smoothing is enabled in the theme.

  Collection data type: `Boolean`

  Available in Mac OS X v10.0 and later.

  Declared in `Appearance.h`.

`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`

  Available in Mac OS X v10.0 and later.

  Declared in `Appearance.h`.

Discussion
Your application may use these collection tags with the functions `SetTheme` (page 97) and `GetTheme` (page 59) to access aspects of a theme. The data type contained in each of the collection items accessed is noted below.

Theme Drawing States

Specify the state in which human interface elements are drawn.

```
enum {
    kThemeStateInactive = 0,
    kThemeStateActive = 1,
    kThemeStatePressed = 2,
    kThemeStateRollover = 6,
    kThemeStateUnavailable = 7,
    kThemeStateUnavailableInactive = 8,
    kThemeStatePressedUp = 2,
    kThemeStatePressedDown = 3
};
typedef UInt32 ThemeDrawState;
```

Constants

`kThemeStateInactive`
The drawing state of elements in an inactive window.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeStateActive`
The drawing state of elements in an active window.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeStatePressed`
The drawing state of elements in which a mouse click is occurring.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeStateRollover`
The drawing state of elements over which the mouse is positioned.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeStateUnavailable`
The drawing state of elements that are disabled. This state is used to indicate that an element cannot be clicked.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeStateUnavailableInactive`
The drawing state of elements that are disabled and are not in the currently active window.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeStatePressedUp`
For stepper controls, the drawing state of the increment button in which a mouse click is occurring.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeStatePressedDown`
For stepper controls, the drawing state of the decrement button in which a mouse click is occurring.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

Theme Metrics

Specify metric properties of user interface elements in the current environment.

```
typedef UInt32 ThemeMetric;
enum {
    kThemeMetricScrollBarWidth = 0,
    kThemeMetricSmallScrollBarWidth = 1,
    kThemeMetricCheckBoxHeight = 2,
    kThemeMetricRadioButtonHeight = 3,
    kThemeMetricEditTextWhitespace = 4,
    kThemeMetricEditTextFrameOutset = 5,
    kThemeMetricListBoxFrameOutset = 6,
    kThemeMetricFocusRectOutset = 7,
    kThemeMetricImageWellThickness = 8,
    kThemeMetricScrollBarOverlap = 9,
    kThemeMetricLargeTabHeight = 10,
    kThemeMetricLargeTabCapsWidth = 11,
    kThemeMetricTabFrameOverlap = 12,
    kThemeMetricTabIndentOrStyle = 13,
    kThemeMetricTabOverlap = 14,
    kThemeMetricSmallTabHeight = 15,
    kThemeMetricSmallTabCapsWidth = 16,
    kThemeMetricPushButtonHeight = 19,
    kThemeMetricListHeaderHeight = 20,
    kThemeMetricDisclosureTriangleHeight = 25,
    kThemeMetricDisclosureTriangleWidth = 26,
    kThemeMetricLittleArrowsHeight = 27,
    kThemeMetricLittleArrowsWidth = 28,
    kThemeMetricPopupButtonHeight = 30,
    kThemeMetricSmallPopupButtonHeight = 31,
    kThemeMetricLargeProgressBarThickness = 32,
    kThemeMetricPullDownHeight = 33,
    kThemeMetricSmallPullDownHeight = 34,
    kThemeMetricResizeControlHeight = 38,
    kThemeMetricSmallResizeControlHeight = 39,
    kThemeMetricHSliderHeight = 41,
    kThemeMetricHSliderTickHeight = 42,
    kThemeMetricVSliderWidth = 45,
    kThemeMetricVSliderTickWidth = 46,
    kThemeMetricTitleBarControlsHeight = 49,
    kThemeMetricCheckBoxWidth = 50,
    kThemeMetricRadioButtonWidth = 52,
    kThemeMetricNormalProgressBarThickness = 58,
    kThemeMetricProgressBarShadowOutset = 59,
    kThemeMetricSmallProgressBarShadowOutset = 60,
    kThemeMetricPrimaryGroupBoxContentInset = 61,
    kThemeMetricSecondaryGroupBoxContentInset = 62,
    kThemeMetricMenuMarkColumnWidth = 63,
    kThemeMetricMenuExcludedMarkColumnWidth = 64,
    kThemeMetricMenuMarkIndent = 65,
    kThemeMetricMenuTextLeadingEdgeMargin = 66,
    kThemeMetricMenuTextTrailingEdgeMargin = 67,
    kThemeMetricMenuIndentWidth = 68,
    kThemeMetricMenuIconTrailingEdgeMargin = 69
};
enum {
    kThemeMetricDisclosureButtonHeight = 17,
    kThemeMetricRoundButtonSize = 18,
    kThemeMetricSmallCheckBoxHeight = 21,
    kThemeMetricDisclosureButtonWidth = 22,
    kThemeMetricSmallDisclosureButtonHeight = 23,
```

```
    kThemeMetricSmallDisclosureButtonWidth = 24,
    kThemeMetricPaneSplitterHeight = 29,
    kThemeMetricSmallPushButtonHeight = 35,
    kThemeMetricSmallRadioButtonHeight = 36,
    kThemeMetricRelevanceIndicatorHeight = 37,
    kThemeMetricLargeRoundButtonSize = 40,
    kThemeMetricSmallHSliderHeight = 43,
    kThemeMetricSmallHSliderTickHeight = 44,
    kThemeMetricSmallVSliderWidth = 47,
    kThemeMetricSmallVSliderTickWidth = 48,
    kThemeMetricSmallCheckBoxWidth = 51,
    kThemeMetricSmallRadioButtonWidth = 53,
    kThemeMetricSmallHSliderMinThumbWidth = 54,
    kThemeMetricSmallVSliderMinThumbHeight = 55,
    kThemeMetricSmallHSliderTickOffset = 56,
    kThemeMetricSmallVSliderTickOffset = 57
};
enum {
    kThemeMetricComboBoxLargeBottomShadowOffset = 70,
    kThemeMetricComboBoxLargeRightShadowOffset = 71,
    kThemeMetricComboBoxSmallBottomShadowOffset = 72,
    kThemeMetricComboBoxSmallRightShadowOffset = 73,
    kThemeMetricComboBoxLargeDisclosureWidth = 74,
    kThemeMetricComboBoxSmallDisclosureWidth = 75,
    kThemeMetricRoundTextFieldContentInsetLeft = 76,
    kThemeMetricRoundTextFieldContentInsetRight = 77,
    kThemeMetricRoundTextFieldContentInsetBottom = 78,
    kThemeMetricRoundTextFieldContentInsetTop = 79,
    kThemeMetricRoundTextFieldContentHeight = 80,
    kThemeMetricComboBoxMiniBottomShadowOffset = 81,
    kThemeMetricComboBoxMiniDisclosureWidth = 82,
    kThemeMetricComboBoxMiniRightShadowOffset = 83,
    kThemeMetricLittleArrowsMiniHeight = 84,
    kThemeMetricLittleArrowsMiniWidth = 85,
    kThemeMetricLittleArrowsSmallHeight = 86,
    kThemeMetricLittleArrowsSmallWidth = 87,
    kThemeMetricMiniCheckBoxHeight = 88,
    kThemeMetricMiniCheckBoxWidth = 89,
    kThemeMetricMiniDisclosureButtonHeight = 90,
    kThemeMetricMiniDisclosureButtonWidth = 91,
    kThemeMetricMiniHSliderHeight = 92,
    kThemeMetricMiniHSliderMinThumbWidth = 93,
    kThemeMetricMiniHSliderTickHeight = 94,
    kThemeMetricMiniHSliderTickOffset = 95,
    kThemeMetricMiniPopupButtonHeight = 96,
    kThemeMetricMiniPullDownHeight = 97,
    kThemeMetricMiniPushButtonHeight = 98,
    kThemeMetricMiniRadioButtonHeight = 99,
    kThemeMetricMiniRadioButtonWidth = 100,
    kThemeMetricMiniTabCapsWidth  = 101,
    kThemeMetricMiniTabFrameOverlap = 102,
    kThemeMetricMiniTabHeight = 103,
    kThemeMetricMiniTabOverlap = 104,
    kThemeMetricMiniVSliderMinThumbHeight = 105,
    kThemeMetricMiniVSliderTickOffset = 106,
    kThemeMetricMiniVSliderTickWidth = 107,
    kThemeMetricMiniVSliderWidth = 108,
    kThemeMetricRoundTextFieldContentInsetWithIconLeft = 109,
```

```
    kThemeMetricRoundTextFieldContentInsetWithIconRight = 110,
    kThemeMetricRoundTextFieldMiniContentHeight = 111,
    kThemeMetricRoundTextFieldMiniContentInsetBottom = 112,
    kThemeMetricRoundTextFieldMiniContentInsetLeft = 113,
    kThemeMetricRoundTextFieldMiniContentInsetRight = 114,
    kThemeMetricRoundTextFieldMiniContentInsetTop = 115,
    kThemeMetricRoundTextFieldMiniContentInsetWithIconLeft = 116,
    kThemeMetricRoundTextFieldMiniContentInsetWithIconRight = 117,
    kThemeMetricRoundTextFieldSmallContentHeight = 118,
    kThemeMetricRoundTextFieldSmallContentInsetBottom = 119,
    kThemeMetricRoundTextFieldSmallContentInsetLeft = 120,
    kThemeMetricRoundTextFieldSmallContentInsetRight = 121,
    kThemeMetricRoundTextFieldSmallContentInsetTop = 122,
    kThemeMetricRoundTextFieldSmallContentInsetWithIconLeft = 123,
    kThemeMetricRoundTextFieldSmallContentInsetWithIconRight = 124,
    kThemeMetricSmallTabFrameOverlap = 125,
    kThemeMetricSmallTabOverlap = 126,
    kThemeMetricSmallPaneSplitterHeight = 127
};
enum {
    kThemeMetricHSliderTickOffset = 128,
    kThemeMetricVSliderTickOffset = 129,
    kThemeMetricSliderMinThumbHeight = 130,
    kThemeMetricSliderMinThumbWidth = 131,
    kThemeMetricScrollBarMinThumbHeight = 132,
    kThemeMetricScrollBarMinThumbWidth = 133,
    kThemeMetricSmallScrollBarMinThumbHeight = 134,
    kThemeMetricSmallScrollBarMinThumbWidth = 135,
    kThemeMetricButtonRoundedHeight = 136,
    kThemeMetricButtonRoundedRecessedHeight = 137
};
enum {
    kThemeMetricSeparatorSize = 138,
    kThemeMetricTexturedPushButtonHeight = 139,
    kThemeMetricTexturedSmallPushButtonHeight = 140
};
```

Constants

`kThemeMetricScrollBarWidth`

The width of a scroll bar. For horizontal scroll bars, this measurement is actually the scroll bar height.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeMetricSmallScrollBarWidth`

The width of a small scroll bar. For horizontal scroll bars, this measurement is actually the scroll bar height.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeMetricCheckBoxHeight`

The height of the non-label part of a check box control.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeMetricRadioButtonHeight`
The height of the non-label part of a radio button control.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeMetricEditTextWhitespace`
The amount of white space surrounding the text rectangle of the text inside of an Edit Text control. If you select all of the text in an Edit Text control, this white space is visible. The metric is the number of pixels, per side, that the text rectangle is outset to create the whitespace rectangle.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeMetricEditTextFrameOutset`
The thickness of the Edit Text frame that surrounds the whitespace rectangle (which surrounds the text rectangle). The metric is the number of pixels, per side, that the frame rectangle is outset from the whitespace rectangle.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeMetricListBoxFrameOutset`
The number of pixels that the list box frame is outset from the content of the list box.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeMetricFocusRectOutset`
This is a deprecated metric and you should not use it. This metric describes how far from a control the focus rectangle was drawn, but control focus drawing no longer uses this information.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeMetricImageWellThickness`
The thickness of the frame drawn by `DrawThemeGenericWell`.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeMetricScrollBarOverlap`
The number of pixels a scrollbar should overlap any bounding box which surrounds it and scrollable content. This also includes the window frame when a scrollbar is along an edge of the window.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeMetricLargeTabHeight`
The height of the large tab of a tab control.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeMetricLargeTabCapsWidth`
The width of the caps, or end pieces, of the large tabs of a tab control.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeMetricTabFrameOverlap`
The amount you must add to the tab height, `kThemeMetricLargeTabHeight`, to find the rectangle height to use with the various tab drawing primitives. This amount is also the amount that each tab overlaps the tab pane.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeMetricTabIndentOrStyle`
If less than zero, this indicates that the text should be centered on each tab. If greater than zero, the text should be justified, according to the system script direction, and the amount is the offset from the edge at which the text should start drawing.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeMetricTabOverlap`
The amount of space that every tab's drawing rectangle overlapsthat of the tab on either side of it.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeMetricSmallTabHeight`
The height of the small tab of a tab control. This includes the pixels that overlap the tab pane and/or tab pane bar.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeMetricSmallTabCapsWidth`
The width of the caps, or end pieces, of the small tabs of a tab control.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeMetricPushButtonHeight`
The height and the width of the push button control.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeMetricListHeaderHeight`
The height of the list header field of the data browser control.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeMetricDisclosureTriangleHeight`
The height of a disclosure triangle control. This triangle is the not the center of the disclosure button, but its own control.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeMetricDisclosureTriangleWidth`
The width of a disclosure triangle control.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeMetricLittleArrowsHeight`
The height of a little arrows control.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeMetricLittleArrowsWidth`
The width of a little arrows control.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeMetricPopupButtonHeight`
The height of a popup button control.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeMetricSmallPopupButtonHeight`
The height of a small popup button control.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeMetricLargeProgressBarThickness`
The height of the large progress bar, not including its shadow.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeMetricPullDownHeight`
This metric is not used.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeMetricSmallPullDownHeight`
This metric is not used.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeMetricResizeControlHeight`
The height of the window grow box control.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeMetricSmallResizeControlHeight`
The width of the window grow box control.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeMetricHSliderHeight`
The height of the horizontal slider control.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeMetricHSliderTickHeight`
The height of the tick marks for a horizontal slider control.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeMetricVSliderWidth`
The width of the vertical slider control.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeMetricVSliderTickWidth`
The width of the tick marks for a vertical slider control.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeMetricTitleBarControlsHeight`
The height of the title bar widgets (grow, close, and zoom boxes) for a document window.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeMetricCheckBoxWidth`
The width of the non-label part of a check box control.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeMetricRadioButtonWidth`
The width of the non-label part of a radio button control.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeMetricNormalProgressBarThickness`
The height of the normal progress bar, not including its shadow.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeMetricProgressBarShadowOutset`
The number of pixels of shadow depth drawn below the progress bar.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeMetricSmallProgressBarShadowOutset`
The number of pixels of shadow depth drawn below the small progress bar.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeMetricPrimaryGroupBoxContentInset`
The number of pixels that the content of a primary group box is inset from the bounds of the control.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeMetricSecondaryGroupBoxContentInset`
The number of pixels that the content of a secondary group box is from the bounds of the control.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeMetricMenuMarkColumnWidth`
The width allocated to draw the mark character in a menu.

Available in Mac OS X v10.1 and later.

Declared in `Appearance.h`.

`kThemeMetricMenuExcludedMarkColumnWidth`
The width allocated for the mark character in a menu item when the menu has the attribute `kMenuAttrExcludesMarkColumn`.

Available in Mac OS X v10.1 and later.

Declared in `Appearance.h`.

`kThemeMetricMenuMarkIndent`
The indent into the interior of the mark column at which the mark character is drawn.

Available in Mac OS X v10.1 and later.

Declared in `Appearance.h`.

`kThemeMetricMenuTextLeadingEdgeMargin`
The whitespace at the leading edge of menu item text.

Available in Mac OS X v10.1 and later.

Declared in `Appearance.h`.

`kThemeMetricMenuTextTrailingEdgeMargin`
The whitespace at the trailing edge of menu item text.

Available in Mac OS X v10.1 and later.

Declared in `Appearance.h`.

`kThemeMetricMenuIndentWidth`
The width per indent level of a menu item. This indent is set by the `SetMenuItemIndent` function.

Available in Mac OS X v10.1 and later.

Declared in `Appearance.h`.

`kThemeMetricMenuIconTrailingEdgeMargin`
The whitespace at the trailing edge of a menu icon, if the item also has text.

Available in Mac OS X v10.1 and later.

Declared in `Appearance.h`.

`kThemeMetricDisclosureButtonHeight`
The height of a disclosure button.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeMetricRoundButtonSize`
The height and the width of the round button control.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeMetricSmallCheckBoxHeight`
The height of the non-label part of a small check box control.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeMetricDisclosureButtonWidth`
The width of a disclosure button.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeMetricSmallDisclosureButtonHeight`
The height of a small disclosure button.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeMetricSmallDisclosureButtonWidth`
The width of a small disclosure button.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeMetricPaneSplitterHeight`
The height (or width if vertical) of a pane splitter.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeMetricSmallPushButtonHeight`
The height of the small push button control.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeMetricSmallRadioButtonHeight`
The height of the non-label part of a small radio button control.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeMetricRelevanceIndicatorHeight`
The height of the relevance indicator control.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeMetricLargeRoundButtonSize`
The height and the width of the large round button control.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeMetricSmallHSliderHeight`
The height of the small, horizontal slider control.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeMetricSmallHSliderTickHeight`
The height of the tick marks for a small, horizontal slider control.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeMetricSmallVSliderWidth`
The width of the small, vertical slider control.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeMetricSmallVSliderTickWidth`
The width of the tick marks for a small, vertical slider control.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeMetricSmallCheckBoxWidth`
The width of the non-label part of a small check box control.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeMetricSmallRadioButtonWidth`
The width of the non-label part of a small radio button control.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeMetricSmallHSliderMinThumbWidth`
The minimum width of the thumb of a small, horizontal slider control.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeMetricSmallVSliderMinThumbHeight`
The minimum width of the thumb of a small, vertical slider control.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeMetricSmallHSliderTickOffset`
The offset of the tick marks from the appropriate side of a small horizontal slider control.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeMetricSmallVSliderTickOffset`
The offset of the tick marks from the appropriate side of a small vertical slider control.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeMetricComboBoxLargeBottomShadowOffset`
Available in Mac OS X v10.3 and later.
Declared in `Appearance.h`.

`kThemeMetricComboBoxLargeRightShadowOffset`
Available in Mac OS X v10.3 and later.
Declared in `Appearance.h`.

`kThemeMetricComboBoxSmallBottomShadowOffset`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricComboBoxSmallRightShadowOffset`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricComboBoxLargeDisclosureWidth`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricComboBoxSmallDisclosureWidth`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricRoundTextFieldContentInsetLeft`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricRoundTextFieldContentInsetRight`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricRoundTextFieldContentInsetBottom`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricRoundTextFieldContentInsetTop`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricRoundTextFieldContentHeight`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricComboBoxMiniBottomShadowOffset`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricComboBoxMiniDisclosureWidth`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricComboBoxMiniRightShadowOffset`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricLittleArrowsMiniHeight`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricLittleArrowsMiniWidth`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricLittleArrowsSmallHeight`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricLittleArrowsSmallWidth`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricMiniCheckBoxHeight`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricMiniCheckBoxWidth`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricMiniDisclosureButtonHeight`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricMiniDisclosureButtonWidth`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricMiniHSliderHeight`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricMiniHSliderMinThumbWidth`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricMiniHSliderTickHeight`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricMiniHSliderTickOffset`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricMiniPopupButtonHeight`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricMiniPullDownHeight`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricMiniPushButtonHeight`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricMiniRadioButtonHeight`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricMiniRadioButtonWidth`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricMiniTabCapsWidth`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricMiniTabFrameOverlap`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricMiniTabHeight`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricMiniTabOverlap`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricMiniVSliderMinThumbHeight`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricMiniVSliderTickOffset`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricMiniVSliderTickWidth`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricMiniVSliderWidth`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricRoundTextFieldContentInsetWithIconLeft`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricRoundTextFieldContentInsetWithIconRight`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricRoundTextFieldMiniContentHeight`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricRoundTextFieldMiniContentInsetBottom`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricRoundTextFieldMiniContentInsetLeft`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricRoundTextFieldMiniContentInsetRight`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricRoundTextFieldMiniContentInsetTop`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricRoundTextFieldMiniContentInsetWithIconLeft`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricRoundTextFieldMiniContentInsetWithIconRight`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricRoundTextFieldSmallContentHeight`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricRoundTextFieldSmallContentInsetBottom`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricRoundTextFieldSmallContentInsetLeft`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricRoundTextFieldSmallContentInsetRight`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricRoundTextFieldSmallContentInsetTop`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricRoundTextFieldSmallContentInsetWithIconLeft`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricRoundTextFieldSmallContentInsetWithIconRight`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricSmallTabFrameOverlap`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricSmallTabOverlap`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricSmallPaneSplitterHeight`

The height of a small pane splitter. Should only be used in a window with thick borders, such as a textured window.

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeMetricHSliderTickOffset`

The horizontal start offset for the first tick mark on a horizontal slider.

Available in Mac OS X v10.4 and later.

Declared in `Appearance.h`.

`kThemeMetricVSliderTickOffset`

The vertical start offset for the first tick mark on a vertical slider.

Available in Mac OS X v10.4 and later.

Declared in `Appearance.h`.

`kThemeMetricSliderMinThumbHeight`

The minimum height for a thumb on a slider.

Available in Mac OS X v10.4 and later.

Declared in `Appearance.h`.

`kThemeMetricSliderMinThumbWidth`

The minimum width for a thumb on a slider.

Available in Mac OS X v10.4 and later.

Declared in `Appearance.h`.

`kThemeMetricScrollBarMinThumbHeight`

The minimum height for a thumb on a scroll bar.

Available in Mac OS X v10.4 and later.

Declared in `Appearance.h`.

`kThemeMetricScrollBarMinThumbWidth`

The minimum width for a thumb on a scroll bar.

Available in Mac OS X v10.4 and later.

Declared in `Appearance.h`.

`kThemeMetricSmallScrollBarMinThumbHeight`

The minimum height for a thumb on a small scroll bar.

Available in Mac OS X v10.4 and later.

Declared in `Appearance.h`.

`kThemeMetricSmallScrollBarMinThumbWidth`
The minimum width for a thumb on a small scroll bar.

Available in Mac OS X v10.4 and later.

Declared in `Appearance.h`.

`kThemeMetricButtonRoundedHeight`
The height of a round-ended button (for example, the Kind button in a Finder Search query.)

Available in Mac OS X v10.5 and later.

Declared in `Appearance.h`.

`kThemeMetricButtonRoundedRecessedHeight`
The height of the inset round-ended button (for example, the Servers button in a Finder Search query.)

Available in Mac OS X v10.5 and later.

Declared in `Appearance.h`.

`kThemeMetricSeparatorSize`
The height of a horizontal separator, or the width of a vertical separator, drawn with the `HIThemeDrawSeparator` theme primitive.

Available in Mac OS X v10.5 and later.

Declared in `Appearance.h`.

`kThemeMetricTexturedPushButtonHeight`
The height of the push button control designed for use in a textured window.

Available in Mac OS X v10.5 and later.

Declared in `Appearance.h`.

`kThemeMetricTexturedSmallPushButtonHeight`
The height of the small push button control designed for use in a textured window.

Available in Mac OS X v10.5 and later.

Declared in `Appearance.h`.

Theme Backgrounds

Identify theme-compliant backgrounds.

```
enum {
    kThemeBackgroundTabPane = 1,
    kThemeBackgroundPlacard = 2,
    kThemeBackgroundWindowHeader = 3,
    kThemeBackgroundListViewWindowHeader = 4,
    kThemeBackgroundSecondaryGroupBox = 5
};
typedef UInt32 ThemeBackgroundKind;
```

Constants

`kThemeBackgroundTabPane`
The background for a tab pane.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeBackgroundPlacard`
: The background for a placard.

  Available in Mac OS X v10.0 and later.

  Declared in `Appearance.h`.

`kThemeBackgroundWindowHeader`
: The background for a window header.

  Available in Mac OS X v10.0 and later.

  Declared in `Appearance.h`.

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

  Available in Mac OS X v10.0 and later.

  Declared in `Appearance.h`.

Discussion
You can pass a constant of type `ThemeBackgroundKind` to the function `ApplyThemeBackground` (page 26) to specify that an embedded object have a background consistent with the current theme and object in which it is visually embedded.

Theme Brushes

Specify theme-compliant colors and patterns for particular human interface elements.

```
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,
    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,
    kThemeBrushNotificationWindowBackground = 48,
    kThemeBrushMovableModalBackground = 49,
    kThemeBrushSheetBackgroundOpaque = 50,
    kThemeBrushDrawerBackground = 51,
    kThemeBrushToolbarBackground = 52,
    kThemeBrushSheetBackgroundTransparent = 53,
    kThemeBrushMenuBackground = 54,
    kThemeBrushMenuBackgroundSelected = 55,
    kThemeBrushListViewOddRowBackground = 56,
    kThemeBrushListViewEvenRowBackground = 57,
```

```
    kThemeBrushListViewColumnDivider = 58,
    kThemeBrushSheetBackground = kThemeBrushSheetBackgroundOpaque,
    kThemeBrushBlack = -1,
    kThemeBrushWhite = -2,
    kThemeBrushPrimaryHighlightColor = -3,
    kThemeBrushSecondaryHighlightColor = -4,
    kThemeBrushAlternatePrimaryHighlightColor = -5
};
typedef SInt16 ThemeBrush;
```

Constants

`kThemeBrushDialogBackgroundActive`
An active dialog box's background color or pattern.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeBrushDialogBackgroundInactive`
An inactive dialog box's background color or pattern.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeBrushAlertBackgroundActive`
An active alert box's background color or pattern.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeBrushAlertBackgroundInactive`
An inactive alert box's background color or pattern.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeBrushModelessDialogBackgroundActive`
An active modeless dialog box's background color or pattern.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeBrushModelessDialogBackgroundInactive`
An inactive modeless dialog box's background color or pattern.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeBrushUtilityWindowBackgroundActive`
An active utility window's background color or pattern.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeBrushUtilityWindowBackgroundInactive`
An inactive utility window's background color or pattern.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

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

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

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

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

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

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

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

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeBrushChasingArrows`
Asynchronous arrows' color or pattern.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

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

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

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

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeBrushScrollBarDelimiterActive`
The color or pattern used to outline the sides of an active scroll bar.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeBrushScrollBarDelimiterInactive`
The color or pattern used to outline the sides of an inactive scroll bar.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeBrushFocusHighlight`
The color or pattern of the focus ring around an element that is selected.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeBrushPopupArrowActive`
The color or pattern of the arrow on an active pop-up menu button.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeBrushPopupArrowPressed`
The color or pattern of the arrow on a pop-up menu button that is being clicked on by the user.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeBrushPopupArrowInactive`
The color or pattern of the arrow on an inactive pop-up menu button.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeBrushAppleGuideCoachmark`
The color or pattern of an Apple Guide coachmark.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeBrushIconLabelBackgroundSelected`
The color or pattern of the background of an icon's label area, when the icon is selected.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeBrushStaticAreaFill`
The background color or pattern of an element that does not support user interaction.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

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

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeBrushButtonFrameActive`
The color or pattern that outlines an active button. Your application should draw the button outline outside the edge of the button.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeBrushButtonFrameInactive`
The color or pattern that outlines an inactive button. Your application should draw the button outline outside the edge of the button.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeBrushButtonFaceActive`
The color or pattern of the face of an active button.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeBrushButtonFaceInactive`
The color or pattern of the face of an inactive button.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeBrushButtonFacePressed`
The color or pattern of the face of a button that is being clicked on by the user.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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 in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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 in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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 in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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 in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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 in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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 in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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 in Mac OS X v10.0 and later.

  Declared in `Appearance.h`.

`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 in Mac OS X v10.0 and later.

  Declared in `Appearance.h`.

`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 in Mac OS X v10.0 and later.

  Declared in `Appearance.h`.

`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 in Mac OS X v10.0 and later.

  Declared in `Appearance.h`.

`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 in Mac OS X v10.0 and later.

  Declared in `Appearance.h`.

`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 in Mac OS X v10.0 and later.

  Declared in `Appearance.h`.

`kThemeBrushBevelActiveLight`
: For an active bevel button, the color or pattern of the top and left sides of the bevel.

  Available in Mac OS X v10.0 and later.

  Declared in `Appearance.h`.

`kThemeBrushBevelActiveDark`
: For an active bevel button, the color or pattern of the bottom and right sides of the bevel.

  Available in Mac OS X v10.0 and later.

  Declared in `Appearance.h`.

`kThemeBrushBevelInactiveLight`

For an inactive bevel button, the color or pattern of the top and left sides of the bevel.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeBrushBevelInactiveDark`

For an inactive bevel button, the color or pattern of the bottom and right sides of the bevel.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeBrushNotificationWindowBackground`

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeBrushMovableModalBackground`

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeBrushSheetBackgroundOpaque`

Available in Mac OS X v10.1 and later.

Declared in `Appearance.h`.

`kThemeBrushDrawerBackground`

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeBrushToolbarBackground`

Available in Mac OS X v10.1 and later.

Declared in `Appearance.h`.

`kThemeBrushSheetBackgroundTransparent`

Available in Mac OS X v10.1 and later.

Declared in `Appearance.h`.

`kThemeBrushMenuBackground`

Available in Mac OS X v10.2 and later.

Declared in `Appearance.h`.

`kThemeBrushMenuBackgroundSelected`

Available in Mac OS X v10.2 and later.

Declared in `Appearance.h`.

`kThemeBrushListViewOddRowBackground`

Available in Mac OS X v10.4 and later.

Declared in `Appearance.h`.

`kThemeBrushListViewEvenRowBackground`

Available in Mac OS X v10.4 and later.

Declared in `Appearance.h`.

`kThemeBrushListViewColumnDivider`

Available in Mac OS X v10.4 and later.

Declared in `Appearance.h`.

`kThemeBrushSheetBackground`

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeBrushBlack`

Black; this color does not change from theme to theme. You may use this constant instead of specifying a direct RGB value.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeBrushWhite`

White; this color does not change from theme to theme. You may use this constant instead of specifying a direct RGB value.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeBrushPrimaryHighlightColor`

Available in Mac OS X v10.2 and later.

Declared in `Appearance.h`.

`kThemeBrushSecondaryHighlightColor`

Available in Mac OS X v10.2 and later.

Declared in `Appearance.h`.

`kThemeBrushAlternatePrimaryHighlightColor`

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

Discussion

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 98), `SetThemePen` (page 100), and `SetThemeWindowBackground` (page 1933) to specify that the Appearance Manager substitute whatever the appropriate color or pattern is for a given human interface element in the current theme.

Theme Buttons

Identify types of buttons.

```
enum {
    kThemePushButton = 0,
    kThemeCheckBox = 1,
    kThemeRadioButton = 2,
    kThemeBevelButton = 3,
    kThemeArrowButton = 4,
    kThemePopupButton = 5,
    kThemeDisclosureButton = 6,
    kThemeIncDecButton = 7,
    kThemeSmallBevelButton = 8,
    kThemeMediumBevelButton = 3,
    kThemeLargeBevelButton = 9,
    kThemeListHeaderButton = 10,
    kThemeRoundButton = 11,
    kThemeLargeRoundButton = 12,
    kThemeSmallCheckBox = 13,
    kThemeSmallRadioButton = 14,
    kThemeRoundedBevelButton = 15,
    kThemeNormalCheckBox = 1,
    kThemeNormalRadioButton = 2
};
typedef UInt16 ThemeButtonKind;
```

Constants

`kThemePushButton`
Identifies a push button.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeCheckBox`
Identifies a checkbox.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeRadioButton`
Identifies a radio button.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeBevelButton`
Identifies a bevel button with a medium-width bevel; this value is the same as `kThemeMediumBevelButton`.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`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.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeDisclosureButton`
Identifies a disclosure triangle.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

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

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

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

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

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

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeListHeaderButton`
Identifies a sort button for the top of a list.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeRoundButton`
Identifies a round button.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeLargeRoundButton`
Identifies a large round button.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSmallCheckBox`
Identifies a small checkbox.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeSmallRadioButton`
Identifies a small radio button.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeRoundedBevelButton`
Identifies a rounded bevel button.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeNormalCheckBox`
Identifies a checkbox; this value is the same as `kThemeCheckBox`.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeNormalRadioButton`
Identifies a radio button; this value is the same as `kThemeRadioButton`.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

Discussion
You can pass constants of type `ThemeButtonKind` to the function `DrawThemeButton` (page 31) to draw a theme-compliant button of a specific type. You can also pass `ThemeButtonKind` constants to the functions `GetThemeButtonRegion` (page 63) and `GetThemeButtonContentBounds` (page 62) to retrieve information about a specific button type.

Theme Button Adornments

Specify the visual characteristics of a button control.

```
enum {
    kThemeAdornmentNone = 0,
    kThemeAdornmentDefault = (1 << 0),
    kThemeAdornmentFocus = (1 << 2),
    kThemeAdornmentRightToLeft = (1 << 4),
    kThemeAdornmentDrawIndicatorOnly = (1 << 5),
    kThemeAdornmentHeaderButtonLeftNeighborSelected = (1 <<
6),
    kThemeAdornmentHeaderButtonRightNeighborSelected = (1 <<
7),
    kThemeAdornmentHeaderButtonSortUp = (1 << 8),
    kThemeAdornmentHeaderMenuButton = (1 << 9),
    kThemeAdornmentHeaderButtonNoShadow = (1 << 10),
    kThemeAdornmentHeaderButtonShadowOnly = (1 << 11),
    kThemeAdornmentNoShadow = kThemeAdornmentHeaderButtonNoShadow,
    kThemeAdornmentShadowOnly = kThemeAdornmentHeaderButtonShadowOnly,
    kThemeAdornmentArrowLeftArrow = (1 << 6),
    kThemeAdornmentArrowDownArrow = (1 << 7),
    kThemeAdornmentArrowDoubleArrow = (1 << 8),
    kThemeAdornmentArrowUpArrow = (1 << 9)
};
typedef UInt16 ThemeButtonAdornment;
```

Constants

`kThemeAdornmentNone`

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

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeAdornmentDefault`

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

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeAdornmentFocus`

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

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeAdornmentRightToLeft`

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

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeAdornmentHeaderButtonLeftNeighborSelected`
If the bit specified by this mask is set, the left border of the button is drawn as selected (list header button only).

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeAdornmentHeaderButtonRightNeighborSelected`
If the bit specified by this mask is set, the right border of the button is drawn as selected (list header button only).

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeAdornmentHeaderButtonSortUp`
If the bit specified by this mask is set the sort indicator is drawn pointing upward (list header button only).

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeAdornmentHeaderMenuButton`
If the bit specified by this mask is set, the button is drawn as a header menu button (list header button only).

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeAdornmentHeaderButtonNoShadow`
If the bit specified by this mask is set, the non-shadow area of the button is drawn (list header button only).

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeAdornmentHeaderButtonShadowOnly`
If the bit specified by this mask is set, only the shadow area of the button is drawn (list header button only).

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeAdornmentNoShadow`
Use `kThemeAdornmentHeaderButtonNoShadow` instead.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeAdornmentShadowOnly`
Use `kThemeAdornmentHeaderButtonShadowOnly` instead.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeAdornmentArrowLeftArrow`
If the bit specified by this mask is set, a left arrow is drawn on the arrow button.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeAdornmentArrowDownArrow`
If the bit specified by this mask is set, a down arrow is drawn on the arrow button.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeAdornmentArrowDoubleArrow`
If the bit specified by this mask is set, a double arrow is drawn on the arrow button.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeAdornmentArrowUpArrow`
If the bit specified by this mask is set, an up arrow is drawn on the arrow button.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

Discussion
The `ThemeButtonAdornment` enumeration defines masks your application can use in the `ThemeButtonDrawInfo` (page 115) structure to specify that button controls are drawn with the appropriate human interface characteristics.

Theme Button Values

Specify the value of a button.

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

Constants
`kThemeButtonOff`
Identifies a button that is not selected.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeButtonOn`
Identifies a button that is selected.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeDisclosureRight`
Identifies a disclosure triangle that is pointing to the right.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeDisclosureDown`
Identifies a disclosure triangle that is pointing down.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeDisclosureLeft`
Identifies a disclosure triangle that is pointing to the left.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

Discussion
You can use constants of type `ThemeButtonValue` in the `ThemeButtonDrawInfo` (page 115) structure to specify that button controls are drawn with the correct values.

Theme Pop-Up Arrow Orientations

Specify the direction in which a pop-up arrow is drawn on a button.

```
enum {
    kThemeArrowLeft = 0,
    kThemeArrowDown = 1,
    kThemeArrowRight = 2,
    kThemeArrowUp = 3
};
typedef UInt16 ThemeArrowOrientation;
```

Constants
`kThemeArrowLeft`
A left-pointing arrow.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeArrowDown`
A downward-pointing arrow.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeArrowRight`
A right-pointing arrow.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeArrowUp`
An upward-pointing arrow.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

Discussion
You can use a constant of type `ThemeArrowOrientation` in the function `DrawThemePopupArrow` (page 43).

Theme Pop-Up Arrow Sizes

Specify the size of the pop-up arrow that is drawn on a button.

```
enum {
    kThemeArrow3pt = 0,
    kThemeArrow5pt = 1,
    kThemeArrow7pt = 2,
    kThemeArrow9pt = 3
};
typedef UInt16 ThemePopupArrowSize;
```

Constants

`kThemeArrow3pt`
Identifies a pop-up arrow with a 3-pixel base.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeArrow5pt`
Identifies a pop-up arrow with a 5-pixel base.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeArrow7pt`
Identifies a pop-up arrow with a 7-pixel base.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeArrow9pt`
Identifies a pop-up arrow with a 9-pixel base.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

Discussion
You can use a constant of type `ThemePopupArrowSize` in the function `DrawThemePopupArrow` (page 43).

Theme Checkbox Styles

Specify types of checkbox marks.

```
enum {
    kThemeCheckBoxClassicX = 0,
    kThemeCheckBoxCheckMark = 1
};
typedef UInt16 ThemeCheckBoxStyle;
```

Constants

`kThemeCheckBoxClassicX`

An "X" type of checkbox mark.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeCheckBoxCheckMark`

A checkmark type of checkbox mark.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

Discussion

You can call the function `GetThemeCheckBoxStyle` (page 64) to obtain the type of checkbox mark being used in the current theme.

Theme Cursors

Identify types of cursors.

```
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,
    kThemeNotAllowedCursor = 18,
    kThemeResizeUpCursor = 19,
    kThemeResizeDownCursor = 20,
    kThemeResizeUpDownCursor = 21,
    kThemePoofCursor = 22
};
typedef UInt32 ThemeCursor;
```

Constants

`kThemeArrowCursor`

The cursor identified by this constant is typically used as the standard cursor.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeIBeamCursor`

The cursor identified by this constant is typically used when the cursor is over an area where the user can select text.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeCrossCursor`

The cursor identified by this constant is typically used when the cursor is over an area where the user can draw graphics.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemePlusCursor`

The cursor identified by this constant is typically used when the cursor is over an area where the user can select table cells.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeOpenHandCursor`

The cursor identified by this constant is typically used to indicate that an object may be grabbed or moved by the user.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSpinningCursor`
The cursor identified by this constant is typically used to indicate that an operation is in progress.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeResizeLeftCursor`
The cursor identified by this constant is typically used to indicate that an object may be resized by dragging to the left.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeResizeRightCursor`
The cursor identified by this constant is typically used to indicate that an object may be resized by dragging to the right.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeResizeLeftRightCursor`
The cursor identified by this constant is typically used to indicate that an object may be resized in either direction horizontally.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeNotAllowedCursor`
The cursor identified by this constant is typically used to indicate that the current action is not allowed. For example, you could use this cursor to indicate that an object being dragged cannot be dropped at the current mouse position.

Available in Mac OS X v10.2 and later.

Declared in `Appearance.h`.

`kThemeResizeUpCursor`
The cursor identified by this constant is typically used to indicate that an object may be resized by dragging vertically in the up direction.

Available in Mac OS X v10.3 and later.

Declared in `Appearance.h`.

`kThemeResizeDownCursor`
: The cursor identified by this constant is typically used to indicate that an object may be resized by dragging vertically in the down direction.

  Available in Mac OS X v10.3 and later.

  Declared in `Appearance.h`.

`kThemeResizeUpDownCursor`
: The cursor identified by this constant is typically used to indicate that an object may be resized by dragging vertically in either direction.

  Available in Mac OS X v10.3 and later.

  Declared in `Appearance.h`.

`kThemePoofCursor`
: The cursor identified by this constant is typically used to indicate that a dragged object will go away if it is released at the current mouse position. When the object goes away, a poof cloud animation should occur.

  Available in Mac OS X v10.3 and later.

  Declared in `Appearance.h`.

Discussion
You can pass constants of type `ThemeCursor` to the functions `SetThemeCursor` (page 98) and `SetAnimatedThemeCursor` (page 96) 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.

Theme Font IDs

Identify types of fonts.

```
enum {
    kThemeSystemFont = 0,
    kThemeSmallSystemFont = 1,
    kThemeSmallEmphasizedSystemFont = 2,
    kThemeViewsFont = 3,
    kThemeEmphasizedSystemFont = 4,
    kThemeApplicationFont = 5,
    kThemeLabelFont = 6,
    kThemeMenuTitleFont = 100,
    kThemeMenuItemFont = 101,
    kThemeMenuItemMarkFont = 102,
    kThemeMenuItemCmdKeyFont = 103,
    kThemeWindowTitleFont = 104,
    kThemePushButtonFont = 105,
    kThemeUtilityWindowTitleFont = 106,
    kThemeAlertHeaderFont = 107,
    kThemeSystemFontDetail = 7,
    kThemeSystemFontDetailEmphasized = 8,
    kThemeCurrentPortFont = 200,
    kThemeToolbarFont = 108
};
typedef UInt16 ThemeFontID;
```

Constants

`kThemeSystemFont`

The current (large) system font. This is the font used to draw most interface elements. If you can't find a more appropriate `ThemeFontID` constant, you should use this one. This font is suitable for drawing titles on most custom widgets and buttons, as well as most static text in dialogs and windows.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSmallSystemFont`

The current small system font. This is the font used to draw interface elements when space is at a premium.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSmallEmphasizedSystemFont`

The current small, emphasized system font. This constant is identical to `kThemeSmallSystemFont`, except it draws bold or otherwise emphasized text, as is appropriate for your application's language and script.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeViewsFont`

The current views font. This is the font used to draw file and folder names in Finder windows or other browsable lists.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeEmphasizedSystemFont`

The current emphasized system font. This constant is identical to `kThemeSystemFont`, except it draws bold or otherwise emphasized text, as is appropriate for your application's language and script.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeApplicationFont`
An analog to the Script Manager's notion of the Application Font. This font is a suitable default choice for your application's document-style text editing areas.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeLabelFont`
Generally smaller than `kThemeSmallSystemFont`, this font is appropriate for drawing text labels next to image content that reinforces the meaning of the text, such as you may use with a bevel button.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeMenuTitleFont`
The font used to draw menu titles in the menu bar.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeMenuItemFont`
The font used to draw menu items in menus.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeMenuItemMarkFont`
The font used to draw menu item marks in menus.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeMenuItemCmdKeyFont`
The font used to draw menu item command key equivalents in menus.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeWindowTitleFont`
The font used to draw text in most window title bars.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemePushButtonFont`
The font used to draw text labels on push buttons.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeUtilityWindowTitleFont`
The font used to draw text in utility window title bars.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeAlertHeaderFont`
The font used to draw the first and most important message of an alert window.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeCurrentPortFont`

Unlike the other `ThemeFontID` constants, `kThemeCurrentPortFont` doesn't map to a font appropriate to your application's language or script. It maps directly to the font, size, and style of the current QuickDraw port. This allows you to get somewhat customized behavior out of the functions that take `ThemeFontID` constants.

Note, however, that `kThemeCurrentPortFont` does not support all QuickDraw styles on all platforms; in particular, outline and shadow style are not supported on Mac OS X. Additionally, `kThemeCurrentPortFont` is not completely unicode savvy; use of `kThemeCurrentPortFont` may result in errors having to do with the current port's font not being appropriate for rendering or measuring all glyphs in a given unicode string.

Because of overhead associated with gathering QuickDraw font information and converting it to the native font format on Mac OS X, using `kThemeCurrentPortFont` may slow down your text drawing and measuring significantly compared to other `ThemeFontID` constants. Use `kThemeCurrentPortFont` only as a last resort.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeToolbarFont`

The font used to draw the label of a toolbar item.

Available in Mac OS X v10.2 and later.

Declared in `Appearance.h`.

Discussion
A `ThemeFontID` constant is a virtual font ID that you can pass to one of the Appearance Manager's text-related functions. Within these functions, the `ThemeFontID` is mapped to the appropriate font or fonts, size, and style based on a number of factors, including:

- the system appearance (Platinum on Mac OS 9 and Aqua on Mac OS X)
- the string to be rendered, if any
- the language or script that the application is running in

The `ThemeFontID` constants allow you to get the correct text appearance for the platform your application is currently running on.

kPublicThemeFontCount

The total number of public `ThemeFontID` constants.

```
enum {
    kPublicThemeFontCount = 17
};
```

Theme Text Colors

Identify the text colors appropriate to the contexts in which text is used.

```
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 1.0.1 and 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 Appearamce 1.1 and later*/
    kThemeTextColorTabFrontActive = 39,
    kThemeTextColorTabNonFrontActive = 40,
    kThemeTextColorTabNonFrontPressed = 41,
    kThemeTextColorTabFrontInactive = 42,
    kThemeTextColorTabNonFrontInactive = 43,
    kThemeTextColorIconLabelSelected = 44,
    kThemeTextColorBevelButtonStickyActive = 45,
    kThemeTextColorBevelButtonStickyInactive = 46

/*Text colors available in Appearance 1.1.1 and later*/
    kThemeTextColorNotification = 47

/*Text colors available later than Mac OS X 10.1.3
    kThemeTextColorSystemDetail = 48
```

```
/*Specific colors that do not change from theme to theme*/
    kThemeTextColorBlack = -1,
    kThemeTextColorWhite = -2
};
typedef SInt16 ThemeTextColor;
```

Constants

`kThemeTextColorDialogActive`
Text color for an active dialog box.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeTextColorDialogInactive`
Text color for an inactive dialog box.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeTextColorAlertActive`
Text color for an active alert box.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeTextColorAlertInactive`
Text color for an inactive alert box.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeTextColorModelessDialogActive`
Text color for an active modeless dialog box.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeTextColorModelessDialogInactive`
Text color for an inactive modeless dialog box.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeTextColorWindowHeaderActive`
Text color for the window header of an active window.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeTextColorWindowHeaderInactive`
Text color for the window header of an inactive window.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeTextColorPlacardActive`
Text color for a placard in an active window.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeTextColorPlacardInactive`
Text color for a placard in an inactive window.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeTextColorPlacardPressed`
Text color for a placard that is being clicked on by the user.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeTextColorPushButtonActive`
Text color for an active push button.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeTextColorPushButtonInactive`
Text color for an inactive push button.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeTextColorPushButtonPressed`
Text color for a push button that is being clicked on by the user.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeTextColorBevelButtonActive`
Text color for an active bevel button.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeTextColorBevelButtonInactive`
Text color for an inactive bevel button.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeTextColorBevelButtonPressed`
Text color for a bevel button that is being clicked on by the user.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeTextColorPopupButtonActive`
Text color for the menu of an active pop-up menu button.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeTextColorPopupButtonInactive`
Text color for the menu of an inactive pop-up menu button.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeTextColorPopupButtonPressed`
Text color for the menu of a pop-up menu button that is being clicked on by the user.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeTextColorIconLabel`
Text color for an icon label.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeTextColorListView`
Text color for the contents of a list view column.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeTextColorDocumentWindowTitleActive`
Text color for the title of an active document window.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeTextColorDocumentWindowTitleInactive`
Text color for the title of an inactive document window.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeTextColorMovableModalWindowTitleActive`
Text color for the title of an active movable modal window.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeTextColorMovableModalWindowTitleInactive`
Text color for the title of inactive movable modal window.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeTextColorUtilityWindowTitleActive`
Text color for the title of an active utility (floating) window.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeTextColorUtilityWindowTitleInactive`
Text color for the title of an inactive utility (floating) window.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeTextColorPopupWindowTitleActive`
Text color for the title of an active pop-up window.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeTextColorPopupWindowTitleInactive`
Text color for the title of an inactive pop-up window.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeTextColorRootMenuActive`
Text color for an active menu bar title.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeTextColorRootMenuSelected`
Text color for a menu bar title that is being selected by the user.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeTextColorRootMenuDisabled`
Text color for a disabled menu bar title.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeTextColorMenuItemActive`
Text color for an active menu item.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeTextColorMenuItemSelected`
Text color for a menu item that is being selected by the user.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeTextColorMenuItemDisabled`
Text color for a disabled menu item.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeTextColorPopupLabelActive`
Text color for the label of an active pop-up menu button.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeTextColorPopupLabelInactive`
Text color for the label of an inactive pop-up menu button.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeTextColorTabFrontActive`
Text color for the front tab of an active tab control.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeTextColorTabNonFrontActive`
Text color for an active tab that is not the frontmost of a tab control.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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 in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeTextColorTabFrontInactive`
Text color for the front tab of an inactive tab control.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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 in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeTextColorIconLabelSelected`
Text color for the label of an icon that is currently selected.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeTextColorBevelButtonStickyActive`
Text color for an active bevel button that is currently on.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeTextColorBevelButtonStickyInactive`
Text color for an inactive bevel button that is currently on.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeTextColorBlack`
Black; this color does not change from theme to theme. You may use this constant instead of specifying a direct RGB value.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeTextColorWhite`
White; this color does not change from theme to theme. You may use this constant instead of specifying a direct RGB value.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

Discussion
You can pass a constant of type `ThemeTextColor` to the function `SetThemeTextColor` (page 101) 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 74) to obtain the actual color in use under the current theme for the specified `ThemeTextColor` constant.

Theme Menu Types

Specify a type of menu.

```
enum {
    kThemeMenuTypePullDown = 0,
    kThemeMenuTypePopUp = 1,
    kThemeMenuTypeHierarchical = 2,
    kThemeMenuTypeInactive = 0x0100
};
typedef UInt16 ThemeMenuType;
```

Constants

`kThemeMenuTypePullDown`
A pull-down menu.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeMenuTypePopUp`
A pop-up menu.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeMenuTypeHierarchical`
A hierarchical menu.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeMenuTypeInactive`
An inactive menu. Add this value to any other menu type if the entire menu is inactive.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

Discussion
You can pass constants of type `ThemeMenuType` in the `inMenuType` parameter of `GetThemeMenuBackgroundRegion` (page 66) and `DrawThemeMenuBackground` (page 37).

Theme Menu States

Specify the state in which theme-compliant menus are drawn.

```
enum {
    kThemeMenuActive = 0,
    kThemeMenuSelected = 1,
    kThemeMenuDisabled = 3
};
typedef UInt16 ThemeMenuState;
```

Constants

`kThemeMenuActive`

Menu is drawn in its active state.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeMenuSelected`

Menu is drawn in its selected state.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

Discussion

You can pass constants of type `ThemeMenuState` in the `inState` parameter of `DrawThemeMenuItem` (page 38) and `DrawThemeMenuTitle` (page 40). The `ThemeMenuState` constants are available with Appearance Manager 1.0.1 and later.

Theme Menu Bar States

Specify whether theme-compliant menu bars are drawn as normal or selected.

```
enum {
    kThemeMenuBarNormal = 0,
    kThemeMenuBarSelected = 1
};
typedef UInt16 ThemeMenuBarState;
```

Constants

`kThemeMenuBarNormal`

Menu bar is drawn in its normal state.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeMenuBarSelected`

Menu bar is drawn in its selected state.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

Discussion

You can pass constants of type `ThemeMenuBarState` in the `inState` parameter of `DrawThemeMenuBarBackground` (page 38). The `ThemeMenuBarState` constants are available with Appearance Manager 1.0.1 and later.

Theme Menu Item Types

Identify types of menu items.

```
enum {
    kThemeMenuItemPlain = 0,
    kThemeMenuItemHierarchical = 1,
    kThemeMenuItemScrollUpArrow = 2,
    kThemeMenuItemScrollDownArrow = 3,
    kThemeMenuItemAtTop = 0x0100,
    kThemeMenuItemAtBottom = 0x0200,
    kThemeMenuItemHierBackground = 0x0400,
    kThemeMenuItemPopUpBackground = 0x0800,
    kThemeMenuItemHasIcon = 0x8000,
    kThemeMenuItemNoBackground = 0x4000
};
typedef UInt16 ThemeMenuItemType;
```

Constants

`kThemeMenuItemPlain`
A plain menu item.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeMenuItemHierarchical`
A hierarchical menu item.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeMenuItemScrollUpArrow`
A scroll-up arrow.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeMenuItemScrollDownArrow`
A scroll-down arrow.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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 in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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 in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeMenuItemHierBackground`
This value may be added to other `ThemeMenuItemType` constants to specify that the item being drawn is located in a hierarchical menu.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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 in Mac OS X v10.0 and later.

  Declared in `Appearance.h`.

`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 in Mac OS X v10.0 and later.

  Declared in `Appearance.h`.

`kThemeMenuItemNoBackground`
: This value may be added to other `ThemeMenuItemType` constants to specify that the menu background should not be drawn along with the menu item. Available in Mac OS X.

  Available in Mac OS X v10.1 and later.

  Declared in `Appearance.h`.

Discussion
Your application may pass a `ThemeMenuItemType` constant to the function `DrawThemeMenuItem` (page 38) to draw a menu item of the specified type, or it may pass a `ThemeMenuItemType` constant to the function `GetThemeMenuItemExtra` (page 68) to retrieve spatial information for the given menu item type under the current theme.

kThemeMenuSquareMenuBar

Indicates that the menu bar should be drawn with square corners.

```
enum {
    kThemeMenuSquareMenuBar = (1 << 0)
};
```

Constants
`kThemeMenuSquareMenuBar`
: Menu bar is drawn with square corners.

  Available in Mac OS X v10.0 and later.

  Declared in `Appearance.h`.

Discussion
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`.

Theme Scroll Bar Arrow Styles

Specify types of scroll bar arrows.

```
enum {
    kThemeScrollBarArrowsSingle = 0,
    kThemeScrollBarArrowsLowerRight = 1
};
typedef UInt16 ThemeScrollBarArrowStyle;
```

Constants

`kThemeScrollBarArrowsSingle`
Specifies the use of a single arrow at each end of a scroll bar.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`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.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

Discussion
You can call the function `GetThemeScrollBarArrowStyle` (page 70) to obtain the type of scroll bar arrows being used in the current theme.

Theme Scroll Box Styles

Specify types of scroll boxes.

```
enum {
    kThemeScrollBarThumbNormal = 0,
    kThemeScrollBarThumbProportional = 1
};
typedef UInt16 ThemeScrollBarThumbStyle;
```

Constants

`kThemeScrollBarThumbNormal`
A classic scroll box.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeScrollBarThumbProportional`
A proportional scroll box.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

Discussion
You can call the function `GetThemeScrollBarThumbStyle` (page 71) to obtain the type of scroll boxes (also known as "scroll indicators" or "thumbs") being used in the current theme.

Theme Size Box Directions

Identify the directions in which a window may be resized.

```
enum {
    kThemeGrowLeft = (1 << 0),
    kThemeGrowRight = (1 << 1),
    kThemeGrowUp = (1 << 2),
    kThemeGrowDown = (1 << 3)
};
typedef UInt16 ThemeGrowDirection;
```

Constants

`kThemeGrowLeft`
If the bit specified by this mask is set, the object can grow to the left.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeGrowRight`
If the bit specified by this mask is set, the object can grow to the right.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeGrowUp`
If the bit specified by this mask is set, the object can grow upward.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeGrowDown`
If the bit specified by this mask is set, the object can grow downward.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

Discussion

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 48) to draw a size box and with the function `GetThemeStandaloneGrowBoxBounds` (page 72) to obtain the bounding rectangle of a size box. The constants may be combined to set more than one direction of growth.

Theme Thumb Directions

Specify the direction in which the indicator points in a slider control.

```
enum {
    kThemeThumbPlain = 0,
    kThemeThumbUpward = 1,
    kThemeThumbDownward = 2
};
typedef UInt8 ThemeThumbDirection;
```

Constants

`kThemeThumbPlain`
A plain indicator; that is, one that does not point in any direction.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeThumbUpward`
For a horizontal slider, an upward-pointing indicator. For a vertical slider, a left-pointing indicator.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeThumbDownward`
For a horizontal slider, a downward-pointing indicator. For a vertical slider, a right-pointing indicator.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

Discussion
You can use constants of type `ThemeThumbDirection` in the `SliderTrackInfo` (page 114) structure. You may use these constants with either horizontal or vertical sliders, and the Appearance Manager interprets the direction of the indicator appropriately.

Theme Tab Directions

Specify the orientation of a tab.

```
enum {
    kThemeTabNorth = 0,
    kThemeTabSouth = 1,
    kThemeTabEast = 2,
    kThemeTabWest = 3
};
typedef UInt16 ThemeTabDirection;
```

Constants
`kThemeTabNorth`
An upward-pointing tab.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeTabSouth`
A downward-pointing tab.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeTabEast`
A right-pointing tab.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeTabWest`
A left-pointing tab.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

Discussion
You can pass constants of type `ThemeTabDirection` to the function `DrawThemeTab` (page 50) to draw theme-compliant tabs that are oriented in various directions. You can also pass a `ThemeTabDirection` constant to the function `GetThemeTabRegion` (page 73) to obtain the region containing a tab oriented in a particular direction.

Theme Tab Styles

Specfiy a type of tab.

```
enum {
    kThemeTabNonFront = 0,
    kThemeTabNonFrontPressed = 1,
    kThemeTabNonFrontInactive = 2,
    kThemeTabFront = 3,
    kThemeTabFrontInactive = 4,
    kThemeTabNonFrontUnavailable = 5,
    kThemeTabFrontUnavailable = 6
};
typedef UInt16 ThemeTabStyle;
```

Constants

`kThemeTabNonFront`
An active tab that is not the frontmost in a tab control.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeTabNonFrontPressed`
A tab that is being clicked on by the user which is not the frontmost tab in a tab control.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeTabFront`
The frontmost tab in an active tab control.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeTabFrontInactive`
The frontmost tab in an inactive tab control.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

Discussion

You can pass a constant of type `ThemeTabStyle` to the function `DrawThemeTab` (page 50) to draw a theme-compliant tab in a specific state. You can also pass a `ThemeTabStyle` constant to the function `GetThemeTabRegion` (page 73) to obtain the region containing a tab in a specific state.

Tab Heights

Specify the height of a tab.

```
enum {
    kThemeSmallTabHeight = 16,
    kThemeLargeTabHeight = 21,
    kThemeTabPaneOverlap = 3,
    kThemeSmallTabHeightMax = 19,
    kThemeLargeTabHeightMax = 24
};
```

Constants

`kThemeSmallTabHeight`
The amount that small tabs protrude from the frame.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeLargeTabHeight`
The amount that large tabs protrude from the frame.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeTabPaneOverlap`
The amount that tabs overlap the frame.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeSmallTabHeightMax`
The small tab height, including the overlap.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeLargeTabHeightMax`
The large tab height, including the overlap.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

Discussion

Use the `kThemeSmallTabHeightMax` and `kThemeLargeTabHeightMax` constants when calculating the rectangles to draw tabs into. This height includes the tab frame overlap. Tabs that are not in the front are only drawn to where they meet the frame, as if the height was just `kThemeLargeTabHeight`, for example. Remember that for east and west tabs, the height referred to is actually the width.

Theme Track Attributes

Specify attributes of a track control.

```
enum {
    kThemeTrackHorizontal = (1 << 0),
    kThemeTrackRightToLeft = (1 << 1),
    kThemeTrackShowThumb = (1 << 2),
    kThemeTrackThumbRgnIsNotGhost = (1 << 3),
    kThemeTrackNoScrollBarArrows = (1 << 4),
    kThemeTrackHasFocus = (1 << 5)
};
typedef UInt16 ThemeTrackAttributes;
```

Constants

`kThemeTrackHorizontal`
If the bit specified by this mask is set, the track is horizontally, not vertically, oriented.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeTrackShowThumb`
If the bit specified by this mask is set, an indicator is drawn for this track.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeTrackThumbRgnIsNotGhost`
If the bit specified by this mask is set, the thumb region is drawn opaque, rather than as a ghost.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeTrackNoScrollBarArrows`
If the bit specified by this mask is set, the track scroll bar is drawn without arrows. This attribute currently has no effect.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeTrackHasFocus`
If the bit specified by this mask is set, the thumb has focus. This attribute currently has effect only on sliders.

Available in Mac OS X v10.2 and later.

Declared in `Appearance.h`.

Discussion

The `ThemeTrackAttributes` enumeration defines masks your application can use in the `ThemeTrackDrawInfo` (page 116) structure to specify various attributes of track controls.

Theme Track States

Specify the state of a track control.

```
enum {
    kThemeTrackActive = 0,
    kThemeTrackDisabled = 1,
    kThemeTrackNothingToScroll = 2,
    kThemeTrackInactive = 3
};
typedef UInt8 ThemeTrackEnableState;
```

Constants

`kThemeTrackActive`

A track in the active state.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeTrackDisabled`

A track in the disabled state.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

Discussion

You can use constants of type `ThemeTrackEnableState` in the `ThemeTrackDrawInfo` (page 116) structure and in the functions `GetThemeScrollBarTrackRect` (page 71) and `HitTestThemeScrollBarArrows` (page 83).

Theme Track Kinds

Identify specific kinds of track-based controls to the Appearance Manager.

```
enum {
    kThemeMediumScrollBar = 0,
    kThemeSmallScrollBar = 1,
    kThemeMediumSlider = 2,
    kThemeMediumProgressBar = 3,
    kThemeMediumIndeterminateBar = 4,
    kThemeRelevanceBar = 5,
    kThemeSmallSlider = 6,
    kThemeLargeProgressBar = 7,
    kThemeLargeIndeterminateBar = 8
};
typedef UInt16 ThemeTrackKind;
```

Constants

`kThemeMediumScrollBar`

A scroll bar.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSmallScrollBar`
A small scroll bar.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeMediumSlider`
A slider bar.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeMediumProgressBar`
A progress bar.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeMediumIndeterminateBar`
An indeterminate progress bar.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

Theme Track Press States

Identify what is pressed in an active scroll bar or slider.

```
enum {
    kThemeLeftOutsideArrowPressed = 0x01,
    kThemeLeftInsideArrowPressed = 0x02,
    kThemeLeftTrackPressed = 0x04,
    kThemeThumbPressed = 0x08,
    kThemeRightTrackPressed = 0x10,
    kThemeRightInsideArrowPressed = 0x20,
    kThemeRightOutsideArrowPressed = 0x40,
    kThemeTopOutsideArrowPressed = 0x01,
    kThemeTopInsideArrowPressed = 0x02,
    kThemeTopTrackPressed = 0x04,
    kThemeBottomTrackPressed = 0x10,
    kThemeBottomInsideArrowPressed = 0x20,
    kThemeBottomOutsideArrowPressed = 0x40
};
typedef UInt8 ThemeTrackPressState;
```

Constants

`kThemeLeftOutsideArrowPressed`
For a horizontal scroll bar containing a single pair of arrows, this constant indicates that the arrow on the left is selected.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeLeftInsideArrowPressed`
For a horizontal scroll bar containing a single pair of arrows, this constant should not be used. For a horizontal scroll bar containing two pairs of arrows with one pair at each end, this constant indicates that the inner arrow at the left end of the scroll bar is selected.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeThumbPressed`
Indicates that the scroll box or indicator is selected.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeRightInsideArrowPressed`
For a horizontal scroll bar containing a single pair of arrows, this constant should not be used. For a horizontal scroll bar containing two pairs of arrows with one pair at each end, this constant indicates that the inner arrow at the right end of the scroll bar is selected.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeRightOutsideArrowPressed`
For a horizontal scroll bar containing a single pair of arrows, this constant indicates that the arrow on the right is selected.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeTopOutsideArrowPressed`
For a vertical scroll bar containing a single pair of arrows, this constant indicates that the arrow on the top is selected.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeTopInsideArrowPressed`
For a vertical scroll bar containing a single pair of arrows, this constant should not be used. For a vertical scroll bar containing two pairs of arrows with one pair at each end, this constant indicates that the inner arrow at the top end of the scroll bar is selected.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeTopTrackPressed`
For a vertical scroll bar or slider, indicates that the end of the track above the scroll box or indicator is selected.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeBottomTrackPressed`
For a vertical scroll bar or slider, indicates that the end of the track beneath the scroll box or indicator is selected.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeBottomInsideArrowPressed`
For a vertical scroll bar containing a single pair of arrows, this constant should not be used. For a vertical scroll bar containing two pairs of arrows with one pair at each end, this constant indicates that the inner arrow at the bottom end of the scroll bar is selected.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeBottomOutsideArrowPressed`
For a vertical scroll bar containing a single pair of arrows, this constant indicates that the arrow on the bottom is selected.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

Discussion
You can use constants of type `ThemeTrackPressState` in structures of type `ScrollBarTrackInfo` (page 114) and `SliderTrackInfo` (page 114) to identify what is pressed in an active scroll bar or slider; the press state is ignored if the control is not active.

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 70) to obtain the type of scroll bar arrows currently being used.

Theme Window Types

Identify windows of specific visual categories.

```
enum {
    kThemeDocumentWindow = 0,
    kThemeDialogWindow = 1,
    kThemeMovableDialogWindow = 2,
    kThemeAlertWindow = 3,
    kThemeMovableAlertWindow = 4,
    kThemePlainDialogWindow = 5,
    kThemeShadowDialogWindow = 6,
    kThemePopupWindow = 7,
    kThemeUtilityWindow = 8,
    kThemeUtilitySideWindow = 9,
    kThemeSheetWindow = 10,
    kThemeDrawerWindow = 11
};
typedef UInt16 ThemeWindowType;
```

Constants

`kThemeDocumentWindow`
A document window.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeDialogWindow`
A modal dialog box.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeMovableDialogWindow`
A movable modal dialog box.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeAlertWindow`
An alert box.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeMovableAlertWindow`
A movable alert box.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`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.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeShadowDialogWindow`
A dialog box with shadowing.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemePopupWindow`
A pop-up window.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeUtilityWindow`
A utility window.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeUtilitySideWindow`
A utility window with a side title bar.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

Theme Window Attributes

Specify interface elements in a window.

```
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),
    kThemeWindowHasDirty = (1 << 9)
};
typedef UInt32 ThemeWindowAttributes;
```

Constants

`kThemeWindowHasGrow`
If the bit specified by this mask is set, the window contains a size box.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeWindowHasHorizontalZoom`
If the bit specified by this mask is set, the window contains a horizontal zoom box.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeWindowHasVerticalZoom`
If the bit specified by this mask is set, the window contains a vertical zoom box.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeWindowHasFullZoom`
If the bit specified by this mask is set, the window contains a full (horizontal and vertical) zoom box.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeWindowHasCloseBox`
If the bit specified by this mask is set, the window contains a close box.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeWindowHasCollapseBox`
If the bit specified by this mask is set, the window contains a collapse box.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeWindowHasTitleText`
If the bit specified by this mask is set, the window contains a title.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeWindowIsCollapsed`
If the bit specified by this mask is set, the window is currently collapsed.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

Discussion
The `ThemeWindowAttributes` enumeration defines masks your application can use to specify the various interface elements that a given window contains.

Theme Title Bar Items

Identify specific types of window title bar items.

```
enum {
    kThemeWidgetCloseBox = 0,
    kThemeWidgetZoomBox = 1,
    kThemeWidgetCollapseBox = 2,
    kThemeWidgetDirtyCloseBox = 6
};
typedef UInt16 ThemeTitleBarWidget;
```

Constants
`kThemeWidgetCloseBox`
Identifies a close box.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeWidgetZoomBox`
Identifies a zoom box.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeWidgetCollapseBox`
Identifies a collapse box.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

Discussion
You may pass constants of type `ThemeTitleBarWidget` to the function `DrawThemeTitleBarWidget` (page 54) 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.

Pop-up Window Tab Positions

Indicate how a pop-up window's tab is positioned.

```
enum {
    kThemePopupTabNormalPosition = 0,
    kThemePopupTabCenterOnWindow = 1,
    kThemePopupTabCenterOnOffset = 2
};
```

Constants
`kThemePopupTabNormalPosition`
Specifies that the left edge of the tab is to be drawn at the position indicated by the `popupTabOffset` field of the `ThemeWindowMetrics` structure.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemePopupTabCenterOnWindow`
Specifies that the tab is to be drawn centered on the window; the `popupTabOffset` field is ignored.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemePopupTabCenterOnOffset`
Specifies that the tab is to be drawn centered at the position indicated by the `popupTabOffset` field.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

Discussion
You can use these constants in the `popupTabPosition` field of the `ThemeWindowMetrics` structure.

Theme Sound Masks

Define masks that are used to specify the classes of sounds that are enabled for a theme.

```
enum {
    kThemeNoSounds = 0,
    kThemeWindowSoundsMask = (1 << 0),
    kThemeMenuSoundsMask = (1 << 1),
    kThemeControlSoundsMask = (1 << 2),
    kThemeFinderSoundsMask = (1 << 3)
};
```

Constants

`kThemeNoSounds`
If no bits are set, no theme sounds are enabled.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeWindowSoundsMask`
If the bit specified by this mask is set, window sounds are enabled.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeMenuSoundsMask`
If the bit specified by this mask is set, menu sounds are enabled.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeControlSoundsMask`
If the bit specified by this mask is set, control sounds are enabled.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeFinderSoundsMask`
If the bit specified by this mask is set, Finder sounds are enabled.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

Discussion

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 122).

Theme Sounds

Identify theme-specific sounds played when an interface object changes state.

```
enum {
    kThemeSoundNone = 0,
    kThemeSoundMenuOpen = 'mnuo',
    kThemeSoundMenuClose = 'mnuc',
    kThemeSoundMenuItemHilite = 'mnui',
    kThemeSoundMenuItemRelease = 'mnus',
    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',
    kThemeSoundPopupWindowOpen = 'pwop',
    kThemeSoundPopupWindowClose = 'pwcl',
    kThemeSoundButtonPress = 'btnp',
    kThemeSoundButtonEnter = 'btne',
    kThemeSoundButtonExit = 'btnx',
    kThemeSoundButtonRelease = 'btnr',
    kThemeSoundDefaultButtonPress = 'dbtp',
    kThemeSoundDefaultButtonEnter = 'dbte',
```

```
    kThemeSoundDefaultButtonExit = 'dbtx',
    kThemeSoundDefaultButtonRelease = 'dbtr',
    kThemeSoundCancelButtonPress = 'cbtp',
    kThemeSoundCancelButtonEnter = 'cbte',
    kThemeSoundCancelButtonExit = 'cbtx',
    kThemeSoundCancelButtonRelease = 'cbtr',
    kThemeSoundCheckboxPress = 'chkp',
    kThemeSoundCheckboxEnter = 'chke',
    kThemeSoundCheckboxExit = 'chkx',
    kThemeSoundCheckboxRelease = 'chkr',
    kThemeSoundRadioPress = 'radp',
    kThemeSoundRadioEnter = 'rade',
    kThemeSoundRadioExit = 'radx',
    kThemeSoundRadioRelease = 'radr',
    kThemeSoundScrollArrowPress = 'sbap',
    kThemeSoundScrollArrowEnter = 'sbae',
    kThemeSoundScrollArrowExit = 'sbax',
    kThemeSoundScrollArrowRelease = 'sbar',
    kThemeSoundScrollEndOfTrack = 'sbte',
    kThemeSoundScrollTrackPress = 'sbtp',
    kThemeSoundSliderEndOfTrack = 'slte',
    kThemeSoundSliderTrackPress = 'sltp',
    kThemeSoundBalloonOpen = 'blno',
    kThemeSoundBalloonClose = 'blnc',
    kThemeSoundBevelPress = 'bevp',
    kThemeSoundBevelEnter = 'beve',
    kThemeSoundBevelExit = 'bevx',
    kThemeSoundBevelRelease = 'bevr',
    kThemeSoundLittleArrowUpPress = 'laup',
    kThemeSoundLittleArrowDnPress = 'ladp',
    kThemeSoundLittleArrowEnter = 'lare',
    kThemeSoundLittleArrowExit = 'larx',
    kThemeSoundLittleArrowUpRelease = 'laur',
    kThemeSoundLittleArrowDnRelease = 'ladr',
    kThemeSoundPopupPress = 'popp',
    kThemeSoundPopupEnter = 'pope',
    kThemeSoundPopupExit = 'popx',
    kThemeSoundPopupRelease = 'popr',
    kThemeSoundDisclosurePress = 'dscp',
    kThemeSoundDisclosureEnter = 'dsce',
    kThemeSoundDisclosureExit = 'dscx',
    kThemeSoundDisclosureRelease = 'dscr',
    kThemeSoundTabPressed = 'tabp',
    kThemeSoundTabEnter = 'tabe',
    kThemeSoundTabExit = 'tabx',
    kThemeSoundTabRelease = 'tabr',
    kThemeSoundDragTargetHilite = 'dthi',
    kThemeSoundDragTargetUnhilite = 'dtuh',
    kThemeSoundDragTargetDrop = 'dtdr',
    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;
```

Constants

`kThemeSoundNone`
Specifies that no sound is played.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeSoundMenuOpen`
Identifies a sound to be played when the user opens a menu.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeSoundMenuClose`
Identifies a sound to be played when the user closes a menu.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeSoundMenuItemHilite`
Identifies a sound to be played when the user highlights a menu item.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeSoundMenuItemRelease`
Identifies a sound to be played when the user selects a menu item.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeSoundWindowClosePress`
Identifies a sound to be played when the user presses the mouse button while the cursor is over a window's close box.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`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.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`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.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeSoundWindowCloseRelease`
Identifies a sound to be played when the user releases the mouse button while the cursor is over a window's close box.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundWindowZoomPress`
Identifies a sound to be played when the user presses the mouse button while the cursor is over a window's zoom box.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundWindowZoomRelease`
Identifies a sound to be played when the user releases the mouse button while the cursor is over a window's zoom box.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundWindowCollapsePress`
Identifies a sound to be played when the user presses the mouse button while the cursor is over a window's collapse box.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundWindowCollapseRelease`
Identifies a sound to be played when the user releases the mouse button while the cursor is over a window's collapse box.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundWindowOpen`
Identifies a sound to be played when the user opens a window.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundWindowClose`
Identifies a sound to be played when the user closes a window.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundWindowZoomIn`
Identifies a sound to be played when the user zooms a window in, that is, to the user state.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundWindowZoomOut`
Identifies a sound to be played when the user zooms a window out, that is, to the standard state.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundWindowCollapseUp`
Identifies a sound to be played when the user collapses a window.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundWindowCollapseDown`
Identifies a sound to be played when the user uncollapses a window.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundUtilWindowOpen`
Identifies a sound to be played when the user opens a utility (floating) window.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundUtilWindowClose`
Identifies a sound to be played when the user closes a utility (floating) window.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundUtilWindowZoomIn`
Identifies a sound to be played when the user zooms a utility (floating) window in, that is, to the user state.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundUtilWindowZoomOut`
Identifies a sound to be played when the user zooms a utility (floating) window out, that is, to the standard state.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundUtilWindowCollapseUp`
Identifies a sound to be played when the user collapses a utility (floating) window.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundUtilWindowCollapseDown`
Identifies a sound to be played when the user uncollapses a utility (floating) window.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundDialogOpen`
Identifies a sound to be played when a dialog box opens.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundDialogClose`
Identifies a sound to be played when a dialog box closes.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundAlertOpen`
Identifies a sound to be played when an alert box opens.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundAlertClose`
Identifies a sound to be played when an alert box closes.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundPopupWindowOpen`
Identifies a sound to be played when a pop-up window opens.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundPopupWindowClose`
Identifies a sound to be played when a pop-up window closes.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundButtonPress`
Identifies a sound to be played when the user presses the mouse button while the cursor is over a push button.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundButtonRelease`
Identifies a sound to be played when the user releases the mouse button while the cursor is over a push button.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundDefaultButtonPress`
Identifies a sound to be played when the user presses the mouse button while the cursor is over a default button.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundDefaultButtonRelease`
Identifies a sound to be played when the user releases the mouse button while the cursor is over a default button.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundCancelButtonPress`
Identifies a sound to be played when the user presses the mouse button while the cursor is over a Cancel button.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundCancelButtonRelease`
Identifies a sound to be played when the user releases the mouse button while the cursor is over a Cancel button.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundCheckboxPress`
Identifies a sound to be played when the user presses the mouse button while the cursor is over a checkbox.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundCheckboxRelease`
Identifies a sound to be played when the user releases the mouse button while the cursor is over a checkbox.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundRadioPress`
Identifies a sound to be played when the user presses the mouse button while the cursor is over a radio button.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

  Available in Mac OS X v10.0 and later.

  Declared in `Appearance.h`.

`kThemeSoundRadioRelease`
: Identifies a sound to be played when the user releases the mouse button while the cursor is over a radio button.

  Available in Mac OS X v10.0 and later.

  Declared in `Appearance.h`.

`kThemeSoundScrollArrowPress`
: Identifies a sound to be played when the user presses the mouse button while the cursor is over a scroll bar arrow.

  Available in Mac OS X v10.0 and later.

  Declared in `Appearance.h`.

`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.

  Available in Mac OS X v10.0 and later.

  Declared in `Appearance.h`.

`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.

  Available in Mac OS X v10.0 and later.

  Declared in `Appearance.h`.

`kThemeSoundScrollArrowRelease`
: Identifies a sound to be played when the user releases the mouse button while the cursor is over a scroll bar arrow.

  Available in Mac OS X v10.0 and later.

  Declared in `Appearance.h`.

`kThemeSoundScrollEndOfTrack`
: Identifies a sound to be played when a scroll box arrives at the end of a scroll bar and can go no further.

  Available in Mac OS X v10.0 and later.

  Declared in `Appearance.h`.

`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).

  Available in Mac OS X v10.0 and later.

  Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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).

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundBalloonOpen`
: Identifies a sound to be played when a help balloon appears.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundBalloonClose`
: Identifies a sound to be played when a help balloon disappears.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundBevelPress`
: Identifies a sound to be played when the user presses the mouse button while the cursor is over a bevel button.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundBevelRelease`
: Identifies a sound to be played when the user releases the mouse button while the cursor is over a bevel button.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

  Available in Mac OS X v10.0 and later.

  Declared in `Appearance.h`.

`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.

  Available in Mac OS X v10.0 and later.

  Declared in `Appearance.h`.

`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.

  Available in Mac OS X v10.0 and later.

  Declared in `Appearance.h`.

`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.

  Available in Mac OS X v10.0 and later.

  Declared in `Appearance.h`.

`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.

  Available in Mac OS X v10.0 and later.

  Declared in `Appearance.h`.

`kThemeSoundPopupPress`
: Identifies a sound to be played when the user presses the mouse button while the cursor is over a pop-up menu button.

  Available in Mac OS X v10.0 and later.

  Declared in `Appearance.h`.

`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.

  Available in Mac OS X v10.0 and later.

  Declared in `Appearance.h`.

`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.

  Available in Mac OS X v10.0 and later.

  Declared in `Appearance.h`.

`kThemeSoundPopupRelease`
Identifies a sound to be played when the user releases the mouse button while the cursor is over a pop-up menu button.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundDisclosurePress`
Identifies a sound to be played when the user presses the mouse button while the cursor is over a disclosure triangle.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundDisclosureRelease`
Identifies a sound to be played when the user releases the mouse button while the cursor is over a disclosure triangle.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundTabPressed`
Identifies a sound to be played when the user presses the mouse button while the cursor is over a tab.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundTabEnter`
Identifies a sound to be played when the user places the cursor over a tab.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundTabRelease`
Identifies a sound to be played when the user releases the mouse button while the cursor is over a tab.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundDragTargetHilite`
Identifies a sound to be played when the user drags an object over a valid drag-and-drop destination.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundDragTargetUnhilite`
Identifies a sound to be played when the user drags an object away from a valid drag-and-drop destination.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundDragTargetDrop`
Identifies a sound to be played when the user drops an object on a valid drag-and-drop destination.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundEmptyTrash`
Identifies a sound to be played when the Finder completes emptying the Trash directory.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundSelectItem`
Identifies a sound to be played when the user presses the mouse button while the cursor is over an item in the Finder.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundNewItem`
Identifies a sound to be played when the user creates a new item.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundReceiveDrop`
Identifies a sound to be played when a Finder object changes parents, such as when the user drops an icon on a folder.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundCopyDone`
Identifies a sound to be played when the Finder completes a copy operation.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundResolveAlias`
Identifies a sound to be played when the Finder resolves an alias.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundLaunchApp`
Identifies a sound to be played when the Finder launches an application.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeSoundDiskInsert`
Identifies a sound to be played when a disk is inserted.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeSoundDiskEject`
Identifies a sound to be played when a disk is ejected.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeSoundFinderDragOnIcon`
Identifies a sound to be played when the user drags an object over an icon.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeSoundFinderDragOffIcon`
Identifies a sound to be played when the user drags an object off of an icon.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

Discussion
Your application can pass constants of type `ThemeSoundKind` to the function `PlayThemeSound` (page 94) to play a theme-specific sound for an interface object when it changes state. Each sound plays asynchronously until complete, then stops automatically.

Theme Drag Sounds

Identify theme-specific sounds played when the user performs a drag.

```
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;
```

Constants
`kThemeDragSoundNone`
Specifies that no drag sound is used.
Available in Mac OS X v10.0 and later.
Declared in `Appearance.h`.

`kThemeDragSoundMoveWindow`
Specifies a sound to be played when the user moves a document window.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeDragSoundGrowWindow`
Specifies a sound to be played when the user resizes a window by dragging the size box.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeDragSoundMoveUtilWindow`
Specifies a sound to be played when the user moves a utility window.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeDragSoundGrowUtilWindow`
Specifies a sound to be played when the user resizes a utility window by dragging the size box.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeDragSoundMoveDialog`
Specifies a sound to be played when the user moves a dialog box.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeDragSoundMoveAlert`
Specifies a sound to be played when the user moves an alert box.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeDragSoundMoveIcon`
Specifies a sound to be played when the user moves an icon.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeDragSoundSliderThumb`
Specifies a sound to be played when the user drags the indicator of a slider control that supports live feedback.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeDragSoundSliderGhost`
Specifies a sound to be played when the user drags the indicator of a slider control that does not support live feedback.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeDragSoundScrollBarThumb`
Specifies a sound to be played when the user drags a scroll box belonging to a scroll bar that supports live feedback.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

`kThemeDragSoundDragging`
Specifies a sound to be played during a Drag Manager drag.

Available in Mac OS X v10.0 and later.

Declared in `Appearance.h`.

Discussion
Your application can pass constants of type `ThemeDragSoundKind` to the function `BeginThemeDragSound` (page 27) 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 59) when the drag has finished. Only one drag sound may occur at a time.

Desktop Picture Alignments

Represent picture alignments that might be reported in the data for the `kThemeDesktopPictureAlignmentTag` theme collection tag.

```
enum {
    kTiledOnScreen = 1,
    kCenterOnScreen = 2,
    kFitToScreen = 3,
    kFillScreen = 4,
    kUseBestGuess = 5
};
```

Constants
`kTiledOnScreen`
The picture draws repeatedly.

Available in Mac OS X v10.2 and later.

Declared in `Appearance.h`.

`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

Available in Mac OS X v10.2 and later.

Declared in `Appearance.h`.

`kFitToScreen`
The picture is reduced if necessary.

Available in Mac OS X v10.2 and later.

Declared in `Appearance.h`.

`kFillScreen`
The picture's aspect ratio is altered if necessary.

Available in Mac OS X v10.2 and later.

Declared in `Appearance.h`.

`kUseBestGuess`
The picture is automatically positioned, based on picture and monitor sizes.

Available in Mac OS X v10.2 and later.

Declared in `Appearance.h`.

appearanceBadBrushIndexErr

Obsolete error codes. Use the result codes listed in the section "Appearance Manager Result Codes" (page 217) instead.

```
enum {
    appearanceBadBrushIndexErr = -30560,
    appearanceProcessRegisteredErr = -30561,
    appearanceProcessNotRegisteredErr = -30562,
    appearanceBadTextColorIndexErr = -30563,
    appearanceThemeHasNoAccents = -30564,
    appearanceBadCursorIndexErr = -30565
};
```

kAEThemeSwitch

Obsolete. Use `kAEAppearanceChanged` , described in "Appearance Manager Apple Events," instead.

```
enum {
    kAEThemeSwitch = kAEAppearanceChanged
};
```

kThemeActiveDialogBackgroundBrush

Obsolete. Use the constants described in "Theme Brushes" instead.

```
enum {
    kThemeActiveDialogBackgroundBrush = kThemeBrushDialogBackgroundActive,
    kThemeInactiveDialogBackgroundBrush =
            kThemeBrushDialogBackgroundInactive,
    kThemeActiveAlertBackgroundBrush = kThemeBrushAlertBackgroundActive,
    kThemeInactiveAlertBackgroundBrush = kThemeBrushAlertBackgroundInactive,
    kThemeActiveModelessDialogBackgroundBrush =
            kThemeBrushModelessDialogBackgroundActive,
    kThemeInactiveModelessDialogBackgroundBrush =
        kThemeBrushModelessDialogBackgroundInactive,
    kThemeActiveUtilityWindowBackgroundBrush =
        kThemeBrushUtilityWindowBackgroundActive,
    kThemeInactiveUtilityWindowBackgroundBrush =
        kThemeBrushUtilityWindowBackgroundInactive,
    kThemeListViewSortColumnBackgroundBrush =
        kThemeBrushListViewSortColumnBackground,
    kThemeListViewBackgroundBrush = kThemeBrushListViewBackground,
    kThemeIconLabelBackgroundBrush = kThemeBrushIconLabelBackground,
    kThemeListViewSeparatorBrush = kThemeBrushListViewSeparator,
    kThemeChasingArrowsBrush = kThemeBrushChasingArrows,
    kThemeDragHiliteBrush = kThemeBrushDragHilite,
    kThemeDocumentWindowBackgroundBrush =
        kThemeBrushDocumentWindowBackground,
    kThemeFinderWindowBackgroundBrush = kThemeBrushFinderWindowBackground
};
```

kThemeActiveScrollBarDelimiterBrush

Obsolete. Use the constants described in "Theme Brushes" instead.

```
enum {
    kThemeActiveScrollBarDelimiterBrush =
        kThemeBrushScrollBarDelimiterActive,
    kThemeInactiveScrollBarDelimiterBrush =
        kThemeBrushScrollBarDelimiterInactive,
    kThemeFocusHighlightBrush     = kThemeBrushFocusHighlight,
    kThemeActivePopupArrowBrush   = kThemeBrushPopupArrowActive,
    kThemePressedPopupArrowBrush  = kThemeBrushPopupArrowPressed,
    kThemeInactivePopupArrowBrush = kThemeBrushPopupArrowInactive,
    kThemeAppleGuideCoachmarkBrush = kThemeBrushAppleGuideCoachmark
};
```

kThemeBrushPassiveAreaFill

Obsolete. Use the `kThemeBrushStaticAreaFill` constant, described in "Theme Brushes," instead.

```
enum {
    kThemeBrushPassiveAreaFill = kThemeBrushStaticAreaFill
};
```

kThemeActiveDialogTextColor

Obsolete. Use the constants described in "Theme Text Colors" instead.

```
enum {
    kThemeActiveDialogTextColor   = kThemeTextColorDialogActive,
    kThemeInactiveDialogTextColor = kThemeTextColorDialogInactive,
    kThemeActiveAlertTextColor    = kThemeTextColorAlertActive,
    kThemeInactiveAlertTextColor  = kThemeTextColorAlertInactive,
    kThemeActiveModelessDialogTextColor =
        kThemeTextColorModelessDialogActive,
    kThemeInactiveModelessDialogTextColor =
        kThemeTextColorModelessDialogInactive,
    kThemeActiveWindowHeaderTextColor = kThemeTextColorWindowHeaderActive,
    kThemeInactiveWindowHeaderTextColor =
        kThemeTextColorWindowHeaderInactive,
    kThemeActivePlacardTextColor  = kThemeTextColorPlacardActive,
    kThemeInactivePlacardTextColor = kThemeTextColorPlacardInactive,
    kThemePressedPlacardTextColor = kThemeTextColorPlacardPressed,
    kThemeActivePushButtonTextColor = kThemeTextColorPushButtonActive,
    kThemeInactivePushButtonTextColor = kThemeTextColorPushButtonInactive,
    kThemePressedPushButtonTextColor = kThemeTextColorPushButtonPressed,
    kThemeActiveBevelButtonTextColor = kThemeTextColorBevelButtonActive,
    kThemeInactiveBevelButtonTextColor = kThemeTextColorBevelButtonInactive,
    kThemePressedBevelButtonTextColor = kThemeTextColorBevelButtonPressed,
    kThemeActivePopupButtonTextColor = kThemeTextColorPopupButtonActive,
    kThemeInactivePopupButtonTextColor = kThemeTextColorPopupButtonInactive,
    kThemePressedPopupButtonTextColor = kThemeTextColorPopupButtonPressed,
    kThemeIconLabelTextColor      = kThemeTextColorIconLabel,
    kThemeListViewTextColor       = kThemeTextColorListView
};
```

kThemeActiveDocumentWindowTitleTextColor

Obsolete. Use the constants described in "Theme Text Colors" instead.

```
enum {
    kThemeActiveDocumentWindowTitleTextColor =
        kThemeTextColorDocumentWindowTitleActive,
    kThemeInactiveDocumentWindowTitleTextColor =
        kThemeTextColorDocumentWindowTitleInactive,
    kThemeActiveMovableModalWindowTitleTextColor =
        kThemeTextColorMovableModalWindowTitleActive,
    kThemeInactiveMovableModalWindowTitleTextColor =
        kThemeTextColorMovableModalWindowTitleInactive,
    kThemeActiveUtilityWindowTitleTextColor =
        kThemeTextColorUtilityWindowTitleActive,
    kThemeInactiveUtilityWindowTitleTextColor =
        kThemeTextColorUtilityWindowTitleInactive,
    kThemeActivePopupWindowTitleColor =
        kThemeTextColorPopupWindowTitleActive,
    kThemeInactivePopupWindowTitleColor =
        kThemeTextColorPopupWindowTitleInactive,
    kThemeActiveRootMenuTextColor = kThemeTextColorRootMenuActive,
    kThemeSelectedRootMenuTextColor = kThemeTextColorRootMenuSelected,
    kThemeDisabledRootMenuTextColor = kThemeTextColorRootMenuDisabled,
    kThemeActiveMenuItemTextColor = kThemeTextColorMenuItemActive,
    kThemeSelectedMenuItemTextColor = kThemeTextColorMenuItemSelected,
    kThemeDisabledMenuItemTextColor = kThemeTextColorMenuItemDisabled,
    kThemeActivePopupLabelTextColor = kThemeTextColorPopupLabelActive,
    kThemeInactivePopupLabelTextColor = kThemeTextColorPopupLabelInactive
};
```

kThemeScrollBar

Obsolete. Use the constants described in "Theme Track Kinds" instead.

```
enum {
    kThemeScrollBar              = kThemeMediumScrollBar,
    kThemeSlider                 = kThemeMediumSlider,
    kThemeProgressBar            = kThemeMediumProgressBar,
    kThemeIndeterminateBar       = kThemeMediumIndeterminateBar
};
```

kThemeMetricCheckBoxGlyphHeight

Obsolete. Use the constants described in "Theme Metrics" instead.

```
enum {
    kThemeMetricCheckBoxGlyphHeight = kThemeMetricCheckBoxHeight,
    kThemeMetricRadioButtonGlyphHeight = kThemeMetricRadioButtonHeight,
    kThemeMetricDisclosureButtonSize = kThemeMetricDisclosureButtonHeight,
    kThemeMetricBestListHeaderHeight = kThemeMetricListHeaderHeight,
    kThemeMetricSmallProgressBarThickness =
        kThemeMetricNormalProgressBarThickness,
    kThemeMetricProgressBarThickness = kThemeMetricLargeProgressBarThickness };
```

kThemeNoAdornment

Obsolete. Use the constants described in "Theme Button Adornments" instead.

```
enum {
    kThemeNoAdornment             = kThemeAdornmentNone,
    kThemeDefaultAdornment        = kThemeAdornmentDefault,
    kThemeFocusAdornment          = kThemeAdornmentFocus,
    kThemeRightToLeftAdornment    = kThemeAdornmentRightToLeft,
    kThemeDrawIndicatorOnly       = kThemeAdornmentDrawIndicatorOnly
};
```

kThemeStateDisabled

Obsolete. Use the constants described in "Theme Drawing States" instead.

```
enum {
    kThemeStateDisabled = 0
};
```

kThemeWidgetABox

Obsolete. Use the constants described in "Theme Title Bar Items" instead.

```
enum {
    kThemeWidgetABox = 3,
    kThemeWidgetBBox = 4,
    kThemeWidgetBOffBox = 5
};
```

Result Codes

The most common result codes returned by Appearance Manager are listed below.

Result Code	Value	Description
`themeInvalidBrushErr`	-30560	Invalid brush color constant Available in Mac OS X v10.0 and later.

Result Code	Value	Description
themeProcessRegisteredErr	-30561	Application already registered as an Appearance Manager client. Available in Mac OS X v10.0 and later.
themeProcessNotRegisteredErr	-30562	Application not registered as Appearance Manager client. Available in Mac OS X v10.0 and later.
themeBadTextColorErr	-30563	Invalid text color constant Available in Mac OS X v10.0 and later.
themeHasNoAccentsErr	-30564	Theme does not support accent colors Available in Mac OS X v10.0 and later.
themeBadCursorIndexErr	-30565	Invalid cursor constant Available in Mac OS X v10.0 and later.
themeScriptFontNotFoundErr	-30566	No font record for specified script. Available in Mac OS X v10.0 and later.
themeMonitorDepthNotSupportedErr	-30567	Theme cannot be supported on all monitors at their current bit depth Available in Mac OS X v10.0 and later.
themeNoAppropriateBrushErr	-30568	Theme brush has no corresponding theme text color Available in Mac OS X v10.2 and later.

Gestalt Constants

You can check for version and feature availability information by using the Appearance Manager selectors defined in the Gestalt Manager. For more information, see *Gestalt Manager Reference*.

Application Manager Reference

Framework: Carbon/Carbon.h

Declared in MacApplication.h

Overview

Note: This document was previously titled *Dock Manager Reference*.

The Application Manager provides a set of functions that Mac OS X applications can use to perform various application-level tasks. For example, you can use the Application Manager to:

- Control the display of system-provided user interface elements such as the menu bar and Dock while your application is in the foreground
- Customize your application's Dock tile by modifying the Dock icon and adding items to the contextual menu displayed for your application
- Display a Spotlight search window
- Display a custom about box for your application
- Retrieve the current application object (HIObject)

Functions by Task

Drawing in the Application Dock Tile

`HIApplicationCreateDockTileContext` (page 227)
Returns a Quartz graphics context for drawing in the application Dock tile.

`BeginCGContextForApplicationDockTile` (page 221)
Returns a Quartz graphics context for drawing in the application Dock tile.

`EndCGContextForApplicationDockTile` (page 223)
Releases the Quartz graphics context for an application Dock tile.

`BeginQDContextForApplicationDockTile` (page 221) Deprecated in Mac OS X v10.5
Returns a QuickDraw graphics port for drawing in the application Dock tile. (Deprecated. Use `BeginCGContextForApplicationDockTile` (page 221) or `HIApplicationCreateDockTileContext` (page 227) instead.)

`EndQDContextForApplicationDockTile` (page 223) Deprecated in Mac OS X v10.5
Releases the QuickDraw graphics port for an application Dock tile. (Deprecated. Use `EndCGContextForApplicationDockTile` (page 223) instead.)

Working With the Dock Menu

`GetApplicationDockTileMenu` (page 224)
Returns the menu containing items added to the contextual menu for your application Dock tile.

`SetApplicationDockTileMenu` (page 231)
Adds items to the contextual menu for your application Dock tile.

Working With the Dock Icon

`SetApplicationDockTileImage` (page 230)
Replaces an application Dock icon.

`OverlayApplicationDockTileImage` (page 229)
Composites an image with your application's Dock icon.

`RestoreApplicationDockTileImage` (page 230)
Restores your application Dock icon to the application icon.

`CreateCGImageFromPixMaps` (page 222)
Creates a Quartz image from an image and a mask.

Getting Scripts and Encodings

`GetApplicationScript` (page 224)
Returns the application script.

`GetApplicationTextEncoding` (page 225)
Returns the application text encoding for Resource Manager resources.

Displaying an About Box

`HIAboutBox` (page 226)
Displays a generic, HI-compliant about box.

Controlling System-Provided User Interface Elements

`SetSystemUIMode` (page 232)
Sets the presentation mode of the calling application.

`GetSystemUIMode` (page 225)
Gets the presentation mode of the calling application.

`HISearchWindowShow` (page 229)
Displays a Spotlight search window.

Getting the Application Object

`HIApplicationGetCurrent` (page 228)
Returns the currently running Carbon application object.

Getting the Focused Window

`HIApplicationGetFocus` (page 228)
Returns either the modeless or effective focused window.

Functions

BeginCGContextForApplicationDockTile

Returns a Quartz graphics context for drawing in the application Dock tile.

```
CGContextRef BeginCGContextForApplicationDockTile (
   void
);
```

Return Value
A graphics context you can use to draw in the application Dock tile with Quartz 2D.

Discussion
This function makes it possible to draw into the application Dock tile at a resolution of 128x128, which is the size of all Dock tiles prior to Mac OS X v10.5. If the user interface scale factor is not 1.0, the drawing will be scaled to the actual size of the tile.

This function locks the application Dock tile to prevent the Dock from drawing in the tile. When you are finished using the context, you must call the function `EndCGContextForApplicationDockTile` to release the context and the lock. Do not use `CGEndContext` or `CFRelease` for this purpose. To ensure that drawing to the context appears onscreen, you should call `CGContextFlush` before releasing the context.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

See Also
`EndCGContextForApplicationDockTile` (page 223)

Declared In
`MacApplication.h`

BeginQDContextForApplicationDockTile

Returns a QuickDraw graphics port for drawing in the application Dock tile. (Deprecated in Mac OS X v10.5. Use `BeginCGContextForApplicationDockTile` (page 221) or `HIApplicationCreateDockTileContext` (page 227) instead.)

```
CGrafPtr BeginQDContextForApplicationDockTile (
   void
);
```

Return Value
A pointer to a graphics port. You can use this port to draw into your application Dock tile with QuickDraw.

Discussion
This function locks the application Dock tile to prevent the Dock from drawing in the tile. When you are finished using the graphics port, you must call the function `EndQDContextForApplicationDockTile` to release the port and the lock. Do not use `DisposePort` for this purpose.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

See Also
`EndQDContextForApplicationDockTile` (page 223)

Declared In
`MacApplication.h`

CreateCGImageFromPixMaps

Creates a Quartz image from an image and a mask.

```
OSStatus CreateCGImageFromPixMaps (
   PixMapHandle inImage,
   PixMapHandle inMask,
   CGImageRef *outImage
);
```

Parameters

inImage
A handle to the image you want to use to create the Quartz image. The image should be the same size as the mask. For use in the Dock, the image should be 128 pixels square. Otherwise, the image can be any size.

inMask
A handle to the mask to use as the alpha channel. The mask should be the same size as the image.

outImage
On return, a Quartz image.

Return Value
A result code.

Discussion
The function `CreateCGImageFromPixMaps` uses the mask as the alpha channel for the resulting image. This allows you to have any level of transparency in the resulting image. You can pass the Quartz image as a parameter to any Quartz 2D drawing function, as well as to Dock tile functions such as the functions `SetApplicationDockTileImage` and `OverlayApplicationDockTileImage`. You can use `CreateCGImageFromPixMaps` to create an image for a badge, and then apply the badge to your application Dock icon.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacApplication.h`

EndCGContextForApplicationDockTile

Releases the Quartz graphics context for an application Dock tile.

```
void EndCGContextForApplicationDockTile (
   CGContextRef inContext
);
```

Parameters

inContext

A Quartz graphics context created by calling `BeginCGContextForApplicationDockTile` or `HIApplicationCreateDockTileContext`. On output, the context is invalid and should no longer be used.

Discussion
This function also releases the lock on the application Dock tile, signaling the Dock that you are done drawing in the tile.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

See Also
`HIApplicationCreateDockTileContext` (page 227)
`BeginCGContextForApplicationDockTile` (page 221)

Declared In
`MacApplication.h`

EndQDContextForApplicationDockTile

Releases the QuickDraw graphics port for an application Dock tile. (Deprecated in Mac OS X v10.5. Use `EndCGContextForApplicationDockTile` (page 223) instead.)

```
void EndQDContextForApplicationDockTile (
   CGrafPtr inContext
);
```

Parameters

inContext

A QuickDraw graphics port acquired by calling `BeginQDContextForApplicationDockTile`. On output, the port is invalid and should no longer be used.

Discussion
This function also releases the lock on the application Dock tile, signaling the Dock that you are done drawing in the tile.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

See Also
`BeginQDContextForApplicationDockTile` (page 221)

Declared In
`MacApplication.h`

GetApplicationDockTileMenu

Returns the menu containing items added to the contextual menu for your application Dock tile.

```
MenuRef GetApplicationDockTileMenu (
   void
);
```

Return Value
The menu containing items added to your application Dock tile menu using the function `SetApplicationDockTileMenu` (page 231), or `NULL` if there are no additional menu items.

Availability
Available in Mac OS X v10.1 and later.
Not available to 64-bit applications.

See Also
`SetApplicationDockTileMenu` (page 231)

Declared In
`MacApplication.h`

GetApplicationScript

Returns the application script.

```
ScriptCode GetApplicationScript (
   void
);
```

Return Value
The application script.

Discussion
Your application needs to get the application script when it uses a function, such as `UseThemeFont`, that takes a script code as a parameter.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacApplication.h`

GetApplicationTextEncoding

Returns the application text encoding for Resource Manager resources.

```
TextEncoding GetApplicationTextEncoding (
   void
);
```

Return Value
The application text encoding.

Discussion
Your application needs to use the application text encoding when it creates a CFString from text stored in Resource Manager resources. Typically the text uses a Mac encoding such as MacRoman or MacJapanese. For more information, see *Programming With the Text Encoding Conversion Manager*.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTCarbonShell

Declared In
`MacApplication.h`

GetSystemUIMode

Gets the presentation mode of the calling application.

```
void GetSystemUIMode (
   SystemUIMode *outMode,
   SystemUIOptions *outOptions
);
```

Parameters

outMode

On output, the caller's presentation mode. Pass `NULL` if you don't need this information. For a list of possible modes, see "Presentation Modes" (page 235). The presentation mode of an application determines which system-provided user interface elements are visible on the screen.

outOptions

On output, the options for the caller's presentation mode. Pass `NULL` if you don't need this information. For a list of possible options, see "Presentation Options" (page 236). Presentation options are used to inhibit or allow certain user interface elements and commands.

Discussion
This function returns information about the presentation mode of the calling application, not the presentation mode of the current login session. The login session mode may be different, since the login session mode is determined by the presentation mode of the frontmost application. If the calling application is not currently the frontmost application, then its presentation mode will not be in use. To track changes in the login session's presentation mode, you may handle the `kEventAppSystemUIModeChanged` Carbon event.

Availability
Available in Mac OS X v10.2 and later.

See Also
SetSystemUIMode (page 232)

Declared In
MacApplication.h

HIAboutBox

Displays a generic, HI-compliant about box.

```
OSStatus HIAboutBox (
   CFDictionaryRef inOptions
);
```

Parameters

inOptions

A dictionary of standard strings, a dictionary with the name of a localized strings file from which to retrieve the strings, or NULL to retrieve the strings from the Info.plist file. See the discussion for details.

Return Value
A result code.

Discussion
When this function is called, it displays a window called an about box that contains your application icon, name, software version, and other optional information. In a Carbon event–based application, the standard application event handler responds to the kHICommandAbout command by calling the function HIAboutBox for you. If your application menu has an About menu item, you will get this behavior for free. You don't need to call this function unless you want to customize the contents of the about box.

In addition to the application name and version, this function is designed to display two additional strings in the about box, a copyright string and a description string. You can customize what this function displays by passing in various options in the inOptions parameter:

- You can pass NULL to display application information defined in the Info.plist file or your bundle resource (not recommended). This function looks the Info.plist file for three keys to get the name, version, and copyright strings: CFBundleName, CFBundleVersion, and CFBundleGetInfoString.
- You can pass a dictionary with replacement values for one or more strings. See "About Box Keys" (page 233) for a list of valid keys in this dictionary. If a replacement string is not passed, the default behavior applies. For example, you could pass some variant of your application name in the dictionary, but not pass a replacement version string or copyright string. The function would display your replacement string, and fall back to looking in the Info.plist file for the other strings.
- You can pass a dictionary with a single entry, the name of a localized strings file that contains replacement values for one or more strings. The dictionary key is kHIAboutBoxStringFileKey, and the value is the name of the strings file without the .strings extension. This function automatically uses that file to find the strings for the about box. This example shows the key-value pairs in a typical strings file:

  ```
  HIAboutBoxName = "AboutBox";
  HIAboutBoxVersion = "v1.0";
  HIAboutBoxCopyright = "© Apple Computer, 2006";
  ```

```
HIAboutBoxDescription = "An Example Application";
```

Again, if a string is not found in that file, this function falls back to looking for a string in the dictionary, and then finally the `Info.plist` file.

Note that the description string can only be specified in an options dictionary or a strings file; this function does not check your `Info.plist` file for a description string.

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In
`MacApplication.h`

HIApplicationCreateDockTileContext

Returns a Quartz graphics context for drawing in the application Dock tile.

```
CGContextRef HIApplicationCreateDockTileContext (
   HISize *outContextSize
);
```

Parameters

outContextSize
On output, the size of the graphics context in which the application should draw.

Return Value
A Quartz graphics context you can use to draw in the application Dock tile. For more information about this context, see the Discussion below.

Discussion
This function makes it possible to draw into the application Dock tile at a resolution other than 128x128, which is the size of all Dock tiles prior to Mac OS X v10.5. In Mac OS X v10.5 and later, dock tiles may use different sizes when the user interface scale factor is not 1.0.

Unlike `BeginCGContextForApplicationDockTile` (page 221), this function returns a context that has no transform applied to it; user space and device space are 1:1. Your application must use the output context size to determine the area in which you should draw in the context.

Because the Dock's tile size can change dynamically, applications that use this function should be prepared to redraw their Dock tile as necessary. A `kEventAppUpdateDockTile` Carbon event is sent when the application needs to redraw its Dock tile.

This function locks the application Dock tile to prevent the Dock from drawing in the tile. When you are finished using the context, you must call the function `EndCGContextForApplicationDockTile` to release the context and the lock. Do not use `CGEndContext` or `CFRelease` for this purpose. To ensure that drawing to the context appears onscreen, you should call `CGContextFlush` before releasing the context.

Availability
Available in Mac OS X v10.5 and later.
Not available to 64-bit applications.

See Also
EndCGContextForApplicationDockTile (page 223)

Declared In
MacApplication.h

HIApplicationGetCurrent

Returns the currently running Carbon application object.

```
HIObjectRef HIApplicationGetCurrent (
   void
);
```

Return Value
The current application object.

Discussion
In Mac OS X v10.5 and later, you can use this function to install your own HIObject delegates on the application object.

The function GetApplicationEventTarget (page 260) returns the event target associated with the application object.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
MacApplication.h

HIApplicationGetFocus

Returns either the modeless or effective focused window.

```
WindowRef HIApplicationGetFocus (
   Boolean inConsideringModalFocus
);
```

Parameters
inConsideringModalFocus
A Boolean value that specifies whether to return the effective focus (true) or the modeless focus (false).

Return Value
The focused window, or NULL if there is no focused window.

Discussion
With the introduction of the modal focus stack in Mac OS X v10.5, an application may have two different focused windows: the modeless focus (the window most recently passed to the function SetUserFocusWindow (page 1934)), and the effective focus (either the modeless focus or, if there is a non-empty modal focus stack, the topmost window in the focus stack). This function returns either window.

Applications can use this function to determine if the modeless focus and effective focus are different windows. An application with a custom HIView can also use this function to determine if the application should show an insertion point. The insertion point should only be visible if the view is inside the effective focus.

Note that the function `GetUserFocusWindow` (page 1841) returns the modeless focus—the same window returned when you pass `false` to `HIApplicationGetFocus`.

Availability
Available in Mac OS X v10.5 and later.
Not available to 64-bit applications.

Declared In
`MacApplication.h`

HISearchWindowShow

Displays a Spotlight search window.

```
OSStatus HISearchWindowShow (
   CFStringRef inSearchString,
   OptionBits inFlags
);
```

Parameters

inSearchString
An initial query string. Pass `NULL` to open the search window with no initial query string.

inFlags
Optional flags. Currently, you should pass `kNilOptions`.

Return Value
A result code.

Discussion
This function displays a window with the standard Spotlight search interface. For more information, see *Spotlight Query Programming Guide*.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`MacApplication.h`

OverlayApplicationDockTileImage

Composites an image with your application's Dock icon.

```
OSStatus OverlayApplicationDockTileImage (
   CGImageRef inImage
);
```

Parameters

inImage
The image to overlay onto your application Dock icon.

Return Value
A result code.

Discussion
You can overlay an image, such as a badge, to indicate the application's status to the user.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacApplication.h`

RestoreApplicationDockTileImage

Restores your application Dock icon to the application icon.

```
OSStatus RestoreApplicationDockTileImage (
   void
);
```

Return Value
A result code.

Discussion
If you've called the functions `SetApplicationDockTileImage` or `OverlayApplicationDockTileImage`, you can use the function `RestoreApplicationDockTileImage` to restore the Dock icon to the original application icon.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacApplication.h`

SetApplicationDockTileImage

Replaces an application Dock icon.

```
OSStatus SetApplicationDockTileImage (
   CGImageRef inImage
);
```

Parameters

inImage
The image to use for your application Dock tile.

Return Value
A result code.

Discussion
When an application starts up, by default the application icon is always used as the application Dock tile. You can use the function `SetApplicationDockTileImage` to replace the application icon with another image. This can be useful to indicate the state of the application to the user. If you set the image, it will not revert back to its original image when your application terminates. You need to manually restore it before quitting using the function `RestoreApplicationDockTileImage`.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacApplication.h`

SetApplicationDockTileMenu

Adds items to the contextual menu for your application Dock tile.

```
OSStatus SetApplicationDockTileMenu (
   MenuRef inMenu
);
```

Parameters

inMenu
A menu containing the additional items, or `NULL` to remove the current menu.

Return Value
A result code.

Discussion
When you position the cursor over an application Dock tile and hold down the mouse button, a contextual menu automatically displays a list of the application's document windows and standard application Dock menu items such as Open at Login and Show in Finder. You can use the function `SetApplicationDockTileMenu` to add menu items to the contextual menu displayed for your application Dock tile. The items in the menu you pass to this function are inserted into your application Dock tile menu, between the list of document windows and the standard items.

This function increments the reference count of the menu you pass to it. Before the menu is displayed, it receives the Carbon events `kEventMenuPopulate`, `kEventMenuOpening`, and `kEventMenuEnableItems`, so any event handlers for these events can update the menu appropriately. You can receive notifications of and handle selections from the menu using `kEventCommandProcess` Carbon event handlers installed in the application event target. You must make sure each menu item has a command ID, as the `kEventCommandProcess` event sent to your application provides the menu item's command ID.

When you use this function to pass a menu to the Dock, the following state of each menu item is preserved:

- Information about whether the item is a text item or a separator
- The item text (if the item is not a separator)
- The item command ID
- The item command key modifiers (but not the command key itself)
- The item mark
- The item indent

- The item style
- The item icon, if the icon was specified with `kMenuSystemIconSelectorType` or `kMenuIconResourceType`
- The item's submenu
- These menu item attributes:
  - `kMenuItemAttrNotPreviousAlternate`
  - `kMenuItemAttrSectionHeader`
  - `kMenuItemAttrDisabled`
  - `kMenuItemAttrIconDisabled`
  - `kMenuItemAttrSubmenuParentChoosable`
  - `kMenuItemAttrDynamic`

No other menu or menu item state is preserved when the menu is displayed by the Dock. For example, you can set a custom font for the menu or a menu item, but the menu as displayed by the Dock will not use that font.

Availability
Available in Mac OS X v10.1 and later.
Not available to 64-bit applications.

See Also
`GetApplicationDockTileMenu` (page 224)

Declared In
`MacApplication.h`

SetSystemUIMode

Sets the presentation mode of the calling application.

```
OSStatus SetSystemUIMode (
   SystemUIMode inMode,
   SystemUIOptions inOptions
);
```

Parameters

inMode

The new presentation mode. Pass one of the constants listed in "Presentation Modes" (page 235). The presentation mode of an application determines which system-provided user interface elements are visible on the screen.

inOptions

A mask that specifies options controlling how the specified presentation mode behaves. Pass one or more of the flags listed in "Presentation Options" (page 236), or zero to indicate that no options are needed. Presentation options are used to inhibit or allow certain user interface elements and commands.

Return Value
A result code.

Discussion
The purpose of this function is to make it easier to implement a dedicated kiosk system, in which the user is not permitted access to certain elements and features in the system user interface. This function gives your application control over the visibility of the Dock and the menu bar, and over various other system-provided user interface features such as process switching, logout, restart, and shutdown.

If your application is frontmost and you call this function to request a new presentation mode, your presentation mode will take effect immediately. If another application becomes frontmost, the presentation mode you requested will no longer be in effect. If your application becomes frontmost again, the presentation mode you previously established will come back into effect.

When the frontmost application uses this function to change its presentation mode, a `kEventAppSystemUIModeChanged` Carbon event is sent to all applications that have registered for the event. This event is also sent when an application is activated; it contains the newly active application's presentation mode.

In addition to using this function, an application may also specify an initial presentation mode when it is launched by using the `LSUIPresentationMode` key in its `Info.plist` file. This key should be of type `Number` and should have the value of one of the presentation mode constants listed in "Presentation Modes" (page 235).

Special Considerations

If your application uses the `LSUIElement` or `LSBackgroundOnly` key in its `Info.plist` file, you should not use this function. The presentation mode of the current login session is determined by the presentation mode of the frontmost application, and applications that use these keys generally do not become the frontmost application.

Availability
Available in Mac OS X v10.2 and later.

See Also
`GetSystemUIMode` (page 225)

Declared In
`MacApplication.h`

Constants

About Box Keys

Constants that specify keys used in an options dictionary passed to the function `HIAboutBox` (page 226).

```
#define kHIAboutBoxNameKey CFSTR("HIAboutBoxName")
#define kHIAboutBoxVersionKey CFSTR("HIAboutBoxVersion")
#define kHIAboutBoxCopyrightKey CFSTR("HIAboutBoxCopyright")
#define kHIAboutBoxDescriptionKey CFSTR("HIAboutBoxDescription")
#define kHIAboutBoxStringFileKey CFSTR("HIAboutBoxStringFile")
```

Constants

`kHIAboutBoxNameKey`
Key for the application name that replaces the name specified by the `CFBundleName` key in the `Info.plist` file.

Available in Mac OS X v10.3 and later.

Declared in `MacApplication.h`.

`kHIAboutBoxVersionKey`
Key for the application software version number that replaces the version number specified by the `CFBundleVersion` key in the `Info.plist` file.

Available in Mac OS X v10.3 and later.

Declared in `MacApplication.h`.

`kHIAboutBoxCopyrightKey`
Key for the application copyright notice that replaces the text specified by the `CFBundleGetInfoString` key in the `Info.plist` file.

Available in Mac OS X v10.3 and later.

Declared in `MacApplication.h`.

`kHIAboutBoxDescriptionKey`
Key for a short description of the application.

Available in Mac OS X v10.3 and later.

Declared in `MacApplication.h`.

`kHIAboutBoxStringFileKey`
Key for the name of a localized strings file that contains about-box strings for the application.

Available in Mac OS X v10.3 and later.

Declared in `MacApplication.h`.

Discussion
The values associated with the keys in an options dictionary are all strings.

HIToolbox Version Number

Constant that specifies the current version number of HIToolbox.

```
const float kHIToolboxVersionNumber;
```

Constants

`kHIToolboxVersionNumber`
The current HIToolbox version number, which is incremented each time that HIToolbox is rebuilt during the course of a Mac OS X release.

Available in Mac OS X v10.3 and later.

Declared in `MacApplication.h`.

Discussion
You can use this constant to check for the presence of bug fixes documented in HIToolbox release notes. For example, to test for the HIToolbox included in Mac OS X 10.4.2, check that `kHIToolboxVersionNumber` is at least 220. See the header file `MacApplication.h` for a list of the version numbers for specific releases.

Presentation Modes

Constants used to control the presentation of user interface elements provided by Mac OS X, such as the menu bar and Dock.

```
enum {
    kUIModeNormal = 0,
    kUIModeContentSuppressed = 1,
    kUIModeContentHidden = 2,
    kUIModeAllSuppressed = 4,
    kUIModeAllHidden = 3,
};
typedef UInt32 SystemUIMode;
```

Constants

`kUIModeNormal`
All standard system UI elements are visible.
Available in Mac OS X v10.2 and later.
Declared in `MacApplication.h`.

`kUIModeContentSuppressed`
System UI elements positioned in the content area of the screen (the area other than the menu bar) are hidden. However, these elements may automatically show themselves in response to mouse movements or other user activity.
Available in Mac OS X v10.2 and later.
Declared in `MacApplication.h`.

`kUIModeContentHidden`
System UI elements positioned in the content area of the screen (the area other than the menu bar) are hidden.
Available in Mac OS X v10.2 and later.
Declared in `MacApplication.h`.

`kUIModeAllSuppressed`
All system UI elements (including the menu bar) are hidden. However, these elements may automatically show themselves in response to mouse movements or other user activity.
Available in Mac OS X v10.3 and later.
Declared in `MacApplication.h`.

`kUIModeAllHidden`
All system UI elements (including the menu bar) are hidden.
Available in Mac OS X v10.2 and later.
Declared in `MacApplication.h`.

Discussion
The presentation mode of an application determines which system-provided user interface elements are visible on the screen. Your application can specify its presentation mode using the function `SetSystemUIMode` (page 232).

Presentation Options

Flags used to control optional behavior of system-provided user interface elements and features.

```
enum {
    kUIOptionAutoShowMenuBar = 1 << 0,
    kUIOptionDisableAppleMenu = 1 << 2,
    kUIOptionDisableProcessSwitch = 1 << 3,
    kUIOptionDisableForceQuit = 1 << 4,
    kUIOptionDisableSessionTerminate = 1 << 5,
    kUIOptionDisableHide = 1 << 6
};
typedef OptionBits SystemUIOptions;
```

Constants

`kUIOptionAutoShowMenuBar`
: This flag specifies that the menu bar automatically shows itself when the user moves the mouse into the screen area that would ordinarily be occupied by the menu bar. Only valid for the presentation mode `kUIModeAllHidden`.

  Available in Mac OS X v10.2 and later.

  Declared in `MacApplication.h`.

`kUIOptionDisableAppleMenu`
: This flag disables all items in the Apple menu. Valid for all presentation modes.

  Available in Mac OS X v10.2 and later.

  Declared in `MacApplication.h`.

`kUIOptionDisableProcessSwitch`
: This flag disables the Command-Tab and Command-Shift-Tab key sequences to switch the active process, and the global window rotation key sequence selected by the user in the Keyboard preference pane. The function `SetFrontProcess` may still be used to explicitly switch the active process. Only valid with presentation modes other than `kUIModeNormal`.

  Available in Mac OS X v10.2 and later.

  Declared in `MacApplication.h`.

`kUIOptionDisableForceQuit`
: This flag disables the Command-Option-Escape key sequence and the Force Quit menu item in the Apple menu to open the Force Quit window. Only valid with presentation modes other than `kUIModeNormal`.

  Available in Mac OS X v10.2 and later.

  Declared in `MacApplication.h`.

`kUIOptionDisableSessionTerminate`
: This flag disables the Power key (if present) and the Restart, Shut Down, and Log Out menu items in the Apple menu. Only valid with modes other than `kUIModeNormal`.

  Available in Mac OS X v10.2 and later.

  Declared in `MacApplication.h`.

`kUIOptionDisableHide`
: This flag disables the Hide menu item in the Application menu. Note that this option does not prevent this application from being hidden if Hide Others is selected in some other application.

  Available in Mac OS X v10.3 and later.

  Declared in `MacApplication.h`.

Discussion
Presentation mode options are used to inhibit or allow certain user interface elements and commands. Your application can specify these options using the function `SetSystemUIMode` (page 232).

Carbon Event Manager Reference

Framework:	Carbon/Carbon.h
Declared in	CarbonEventsCore.h CarbonEvents.h

Overview

The Carbon Event Manager is the preferred API for handling events in Carbon applications. You can use this interface to handle events generated in response to user input as well as to create your own custom events. Because event handling is so fundamental to all applications, this document is relevant for everyone writing Carbon applications. To use this document, you should be familiar with Macintosh terminology and understand the basics of creating and manipulating the Mac OS user interface (windows, controls, menus, and so on).

For more information about HIObjects and the HIView subclass, see *HIView Programming Guide*.

Functions by Task

Creating and Manipulating Event Handlers

`InstallEventHandler` (page 274)
Installs an event handler on a specified event target.

`InstallStandardEventHandler` (page 278)
Installs the standard event handler for the specified target.

`RemoveEventHandler` (page 294)
Removes the specified event handler.

`AddEventTypesToHandler` (page 246)
Adds events to an installed handler.

`RemoveEventTypesFromHandler` (page 295)
Removes events from an installed event handler.

`CallNextEventHandler` (page 247)
Calls the next handler in the handler chain.

`RegisterEventHotKey` (page 290)
Registers a global hot key.

`UnregisterEventHotKey` (page 306)
Unregisters a global hot key.

`RegisterToolboxObjectClass` (page 291) Deprecated in Mac OS X v10.4
Registers events to be associated with a toolbox object. (Deprecated. Use the HIObject function `HIObjectRegisterSubclass` (page 2335) instead.)

`UnregisterToolboxObjectClass` (page 307) Deprecated in Mac OS X v10.4
Unregisters events for a given toolbox object class (Deprecated. Use the HIObject function `HIObjectUnregisterClass` (page 2340) instead.)

Creating and Manipulating Event Timers

`InstallEventLoopTimer` (page 276)
Installs a timer.

`InstallEventLoopIdleTimer` (page 275)
Installs a timer that fires only when there is no user activity.

`RemoveEventLoopTimer` (page 295)
Removes the specified timer.

`SetEventLoopTimerNextFireTime` (page 300)
Sets the next time that the specified timer will fire.

Creating and Manipulating Events

`GetEventClass` (page 264)
Returns the class of an event (for example, window, mouse, or keyboard).

`GetEventKind` (page 265)
Returns the event kind for the specified event.

`GetEventParameter` (page 266)
Obtains a parameter from the specified event.

`SetEventParameter` (page 300)
Sets a parameter associated with a particular event.

`CreateEvent` (page 252)
Creates an event.

`CopyEvent` (page 250)
Copies an event.

`CopyEventAs` (page 250)
Copies an existing event, allowing you to change the class and kind of the event.

`RetainEvent` (page 296)
Increments the reference count of an event.

`ReleaseEvent` (page 292)
Releases, and possibly disposes of, the specified event.

`GetEventRetainCount` (page 268)
Returns the reference count of an event.

`ConvertEventRefToEventRecord` (page 249)
Converts an event reference into an event record.

`IsEventInMask` (page 280)
Determines whether an event reference matches a `WaitNextEvent`-style event mask.

GetEventMonitorTarget (page 266)
Obtains an event monitor target.

Dispatching Events

SendEventToEventTarget (page 299)
Sends an event to the specified event target.

SendEventToEventTargetWithOptions (page 299)
Sends an event to the specified event target with propagation options.

GetControlEventTarget (page 261)
Obtains the event target reference for the specified control.

GetWindowEventTarget (page 273)
Obtains the event target reference for a specified window.

GetMenuEventTarget (page 270)
Obtains an event target reference for the specified menu.

GetApplicationEventTarget (page 260)
Obtains the event target reference for the application.

GetEventDispatcherTarget (page 264)
Obtains the event target reference for the standard toolbox dispatcher.

GetUserFocusEventTarget (page 272)
Obtains the event target reference for the user focus.

ProcessHICommand (page 287)
Sends a command to the command chain.

Managing Secure Event Input

EnableSecureEventInput (page 257)
Enables secure event input mode.

DisableSecureEventInput (page 255)
Disables secure event input mode.

IsSecureEventInputEnabled (page 282)
Determines whether secure event input mode is enabled.

Managing Event Queues

GetCurrentEventQueue (page 263)
Obtains the current event queue.

GetMainEventQueue (page 269)
Obtains the main event queue.

PostEventToQueue (page 286)
Adds an event to the specified event queue.

RemoveEventFromQueue (page 294)
Removes an event from the event queue.

`IsEventInQueue` (page 281)
Determines whether an event is in a particular queue.

`AcquireFirstMatchingEventInQueue` (page 245)
Obtains the first event that matches the specified list of event classes and kinds.

`FlushEventsMatchingListFromQueue` (page 259)
Removes events from the event queue by kind and class.

`FindSpecificEventInQueue` (page 258)
Finds a specific event in the event queue.

`FlushSpecificEventsFromQueue` (page 260)
Removes specified events from the event queue.

`FlushEventQueue` (page 259)
Removes all events from the event queue.

`GetNumEventsInQueue` (page 271)
Returns the number of events in the event queue.

Managing the Event Loop

`RunApplicationEventLoop` (page 297)
Runs the application event loop.

`QuitApplicationEventLoop` (page 288)
Terminates the application event loop.

`GetMainEventLoop` (page 269)
Obtains a reference to the main event loop.

`GetCurrentEventLoop` (page 263)
Obtains a reference to the current event loop.

`GetCFRunLoopFromEventLoop` (page 261)
Obtains a Core Foundation `CFRunLoop` from an Carbon event loop reference.

`RunCurrentEventLoop` (page 298)
Executes the event loop in the current thread.

`QuitEventLoop` (page 289)
Causes a specific event loop to terminate.

`ReceiveNextEvent` (page 289)
Waits for the next event of a specified type.

Manipulating Event Time

`GetCurrentEventTime` (page 263)
Returns the current time since last system startup, in seconds.

`GetEventTime` (page 268)
Returns the time a specific event occurred.

`SetEventTime` (page 301)
Sets the event time for a given event.

GetLastUserEventTime (page 268)
Returns the last time a user input event arrived in the main event queue of the application.

Implementing Modal Windows

RunAppModalLoopForWindow (page 298)
Puts the window in an application-modal state.

QuitAppModalLoopForWindow (page 288)
Quits the application-modal state for a window.

BeginAppModalStateForWindow (page 247)
Puts the window in an application-modal state, but does not process events.

EndAppModalStateForWindow (page 258)
Ends the application-modal state entered using the function BeginAppModalStateForWindow.

Tracking the Mouse

TrackMouseLocation (page 303)
Tracks the mouse, blocking your application when there is no activity.

TrackMouseLocationWithOptions (page 304)
Tracks the mouse with additional options.

TrackMouseRegion (page 305)
Tracks the mouse within a region.

ChangeMouseTrackingRegion (page 248)
(Deprecated. Use HIViewChangeTrackingArea (page 2445) instead.)

ClipMouseTrackingRegion (page 248)
(Deprecated. No replacement function. Use HIView-based mouse tracking areas instead.)

GetMouseTrackingRegionID (page 270)
(Deprecated. Use HIViewGetTrackingAreaID (page 2465) instead.)

MoveMouseTrackingRegion (page 283)
(Deprecated. No replacement function. Use HIView-based tracking areas instead.)

SetMouseTrackingRegionEnabled (page 302)
(Deprecated. No replacement function. Use HIView-based tracking areas instead.)

ClipWindowMouseTrackingRegions (page 249)
(Deprecated. No replacement function. Use HIView-based tracking areas instead.)

MoveWindowMouseTrackingRegions (page 283)
(Deprecated. No replacement function. Use HIView-based tracking areas instead.)

SetWindowMouseTrackingRegionsEnabled (page 303)
(Deprecated. Use HIView-based tracking areas instead.)

ReleaseWindowMouseTrackingRegions (page 293)
(Deprecated. No replacement function. Use HIView-based tracking areas instead.)

HIMouseTrackingGetParameters (page 273)
Obtains information about how mouse tracking loops should behave.

`CreateMouseTrackingRegion` (page 253) Deprecated in Mac OS X v10.4
Creates a mouse tracking region. (Deprecated. Use the HIView function `HIViewNewTrackingArea` (page 2473) instead.)

`GetMouseTrackingRegionRefCon` (page 271) Deprecated in Mac OS X v10.4
Obtains the reference constant for a mouse tracking region. (Deprecated. No replacement function. Use HIView-based mouse tracking areas instead.)

`ReleaseMouseTrackingRegion` (page 293) Deprecated in Mac OS X v10.4
Releases a mouse tracking region. (Deprecated. Use `HIViewDisposeTrackingArea` (page 2450) instead.)

`RetainMouseTrackingRegion` (page 296) Deprecated in Mac OS X v10.4
Retains a mouse tracking region. (Deprecated. No replacement function. Use HIView-based tracking areas instead.)

Working with Hot Keys

`CopySymbolicHotKeys` (page 252)
Obtains information about symbolic hot keys in the Keyboard preferences pane.

`PushSymbolicHotKeyMode` (page 287)
Sets a new mode for enabling and disabling symbolic hot keys.

`PopSymbolicHotKeyMode` (page 286)
Removes a hot key mode request from the hot key mode stack.

`GetSymbolicHotKeyMode` (page 272)
Obtains the current hot key mode.

Callback-Related Functions

`NewEventComparatorUPP` (page 284)
Creates an event comparator UPP.

`InvokeEventComparatorUPP` (page 278)
Calls an event comparator function through a UPP.

`DisposeEventComparatorUPP` (page 256)
Disposes of an event comparator UPP.

`NewEventHandlerUPP` (page 284)
Creates an event handler UPP.

`DisposeEventHandlerUPP` (page 256)
Disposes of an event handler UPP.

`InvokeEventHandlerUPP` (page 279)
Calls an event handler through a UPP.

`NewEventLoopTimerUPP` (page 285)
Creates an event loop timer UPP.

`InvokeEventLoopTimerUPP` (page 280)
Calls an event loop timer through a UPP.

`DisposeEventLoopTimerUPP` (page 257)
Disposes of an event loop timer.

`NewEventLoopIdleTimerUPP` (page 285)
Creates an event loop idle timer UPP.

`InvokeEventLoopIdleTimerUPP` (page 280)
Calls an event loop idle timer through a UPP.

`DisposeEventLoopIdleTimerUPP` (page 257)
Disposes of an event loop idle timer.

Miscellaneous

`CopyServicesMenuCommandKeys` (page 251)
Obtains information about command key shortcuts in an application's Services menu.

`CreateTypeStringWithOSType` (page 255)
Converts an `OSType` string to a Core Foundation string.

`GetCurrentEventKeyModifiers` (page 262)
Obtains the queue-synchronized keyboard modifier state.

`IsMouseCoalescingEnabled` (page 281)
Indicates whether mouse coalescing is enabled.

`SetMouseCoalescingEnabled` (page 302)
Turns mouse coalescing on or off.

`IsUserCancelEventRef` (page 282)
Returns whether the specified event indicates the user wishes to cancel an operation.

Functions

AcquireFirstMatchingEventInQueue

Obtains the first event that matches the specified list of event classes and kinds.

```
EventRef AcquireFirstMatchingEventInQueue (
   EventQueueRef inQueue,
   ItemCount inNumTypes,
   const EventTypeSpec *inList,
   OptionBits inOptions
);
```

Parameters

inQueue
The queue to check.

inNumTypes
The number of event kinds for which to search. You may pass `0` if you also pass `NULL` for `inList`.

inList
The list of event classes and kinds to search for in the queue. You may pass `NULL` if `inNumTypes` is `0`. Doing so effectively matches *any* event in the queue and causes this function to return the first event in the queue.

inOptions
Must be `kEventQueueOptionsNone`.

Return Value
An event reference, or `NULL` if no events match. The reference count for the event has been incremented (that is, it has been retained), so you must release the event reference.

Discussion
This function does not remove the event from the queue. To remove the event, call `RemoveEventFromQueue` (page 294).

This function does not call the run loop, so no timers fire as a result of calling this function. This function does not cause any window flushing to occur, but it does get new events from the window server.

This function should have better performance characteristics than the older `EventAvail` API.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CarbonEventsCore.h`

AddEventTypesToHandler

Adds events to an installed handler.

```
OSStatus AddEventTypesToHandler (
   EventHandlerRef inHandlerRef,
   ItemCount inNumTypes,
   const EventTypeSpec *inList
);
```

Parameters

inHandlerRef
The event handler to add events to.

inNumTypes
The number of events to add.

inList
A pointer to an array of `EventTypeSpec` structures.

Return Value
A result code. See "Carbon Event Manager Result Codes" (page 452).

Discussion
You can use this function to dynamically change which events you want your handler to respond to.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CarbonEventsCore.h`

BeginAppModalStateForWindow

Puts the window in an application-modal state, but does not process events.

```
OSStatus BeginAppModalStateForWindow (
   WindowRef inWindow
);
```

Parameters

inWindow

The window you wish to behave modally. See the Window Manager documentation for a description of the `WindowRef` data type.

Return Value

A result code. See "Carbon Event Manager Result Codes" (page 452).

Discussion

This function is a lower level function than `RunAppModalLoopForWindow` (page 298). You use it if you want to enter an application modal state for a window but need to control the event loop yourself. Once you begin your application modal state, the menu bar will disable and prepare for the modal situation. You can then call low-level functions (such as `ReceiveNextEvent`) to run the event loop and process events.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`CarbonEvents.h`

CallNextEventHandler

Calls the next handler in the handler chain.

```
OSStatus CallNextEventHandler (
   EventHandlerCallRef inCallRef,
   EventRef inEvent
);
```

Parameters

inCallRef

The event handler call reference passed into your event handler.

inEvent

The event you want to pass to the next handler.

Return Value

A result code. See "Carbon Event Manager Result Codes" (page 452).

Discussion

Calls through to the event handlers below you in the event handler stack of the target to which your handler is bound. You might use this to call through to the default toolbox handling in order to post-process the event. You can only call this routine from within an event handler.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code
BSDLLCTest
CarbonSketch
QTCarbonShell

Declared In
`CarbonEventsCore.h`

ChangeMouseTrackingRegion

(Deprecated in Mac OS X v10.4. Use `HIViewChangeTrackingArea` (page 2445) instead.)

```
OSStatus ChangeMouseTrackingRegion (
   MouseTrackingRef inMouseRef,
   RgnHandle inRegion,
   RgnHandle inClip
);
```

Special Considerations

Tracking areas are HIView-based rather than window-based. HIViews support compositing and Quartz, and provide a much easier way to handle user elements in windows. For more details about tracking areas, see the mouse tracking region section in *Carbon Event Manager Programming Guide*. For details about HIViews, see *HIView Programming Guide*.

Availability
Available in Mac OS X v10.2 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`CarbonEvents.h`

ClipMouseTrackingRegion

(Deprecated in Mac OS X v10.4. No replacement function. Use HIView-based mouse tracking areas instead.)

```
OSStatus ClipMouseTrackingRegion (
   MouseTrackingRef inMouseRef,
   RgnHandle inRegion
);
```

Special Considerations

Tracking areas are HIView-based rather than window-based. HIViews support compositing and Quartz, and provide a much easier way to handle user elements in windows. You generally don't need to modify the clipping of a tracking area. For more details about tracking areas, see the mouse tracking region section in *Carbon Event Manager Programming Guide*. For details about HIViews, see *HIView Programming Guide*.

Availability
Available in Mac OS X v10.2 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`CarbonEvents.h`

ClipWindowMouseTrackingRegions

(Deprecated in Mac OS X v10.4. No replacement function. Use HIView-based tracking areas instead.)

```
OSStatus ClipWindowMouseTrackingRegions (
   WindowRef inWindow,
   OSType inSignature,
   RgnHandle inClip
);
```

Special Considerations

Tracking areas are HIView-based rather than window-based. HIViews support compositing and Quartz, and provide a much easier way to handle user elements in windows. You generally don't need to modify the clipping of a tracking area. For more details about tracking areas, see the mouse tracking region section in *Carbon Event Manager Programming Guide*. For details about HIViews, see *HIView Programming Guide*.

Availability
Available in Mac OS X v10.2 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`CarbonEvents.h`

ConvertEventRefToEventRecord

Converts an event reference into an event record.

```
Boolean ConvertEventRefToEventRecord (
   EventRef inEvent,
   EventRecord *outEvent
);
```

Parameters

`inEvent`
The event reference to convert.

`outEvent`
The event record to fill out. See the Event Manager documentation for a description of the `EventRecord` data type.

Return Value
A Boolean value indicating whether the conversion was successful (`true`) or not (`false`).

Discussion
This function helps you when you need an `EventRecord` structure and all you have is a Carbon event reference. If the event can be converted, `outEvent` is filled in and the function returns `true`. If not, the function returns `false` and `outEvent` contains `nullEvent`.

This function can convert the following events:

- kEventMouseDown, kEventMouseUp, kEventMouseMoved, and kEventMouseDragged (kEventClassMouse)
- kEventRawKeyDown, kEventRawKeyUp, and kEventRawKeyRepeat (kEventClassKeyboard)
- kEventWindowUpdate, kEventWindowActivate, kEventWindowDeactivate, and kEventWindowCursorChange (kEventClassWindow)
- kEventAppActivated and kEventAppDeactivate (kEventClassApplication)
- kEventAppleEvent (kEventClassAppleEvents)
- kEventControlTrack (kEventClassControl) is converted to a mouse down event in Mac OS X v10.4 and later

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
CarbonEvents.h

CopyEvent

Copies an event.

```
EventRef CopyEvent (
   EventRef inOther
);
```

Parameters

inOther
The event to copy.

Return Value
A new event reference for the specified event.

Discussion
The CopyEvent function makes an exact duplicate of an existing event reference. The reference count for the duplicate event reference is set to 1.

Availability
Available in Mac OS X v10.0 and later.

Declared In
CarbonEventsCore.h

CopyEventAs

Copies an existing event, allowing you to change the class and kind of the event.

```
EventRef CopyEventAs (
   CFAllocatorRef inAllocator,
   EventRef inOther,
   OSType inEventClass,
   UInt32 inEventKind
);
```

Parameters

inOther

The allocator to use to allocate the event data. Pass `NULL` or `kCFAllocatorDefault` to use the standard allocator.

inOther

The event to copy.

inEventClass

The new event class for the copy of the event.

inEventKind

The new event kind for the copy of the event.

Return Value
A new event reference or `NULL` if the `inOther` was `NULL` or memory for the new event could not be allocated.

Discussion
The `CopyEventAs` is useful during event flow and transformation. For example, this function is used when upgrading a raw mouse down to a window click event, to ensure that the window click event has exactly the same parameters as the original mouse down event.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CarbonEventsCore.h`

CopyServicesMenuCommandKeys

Obtains information about command key shortcuts in an application's Services menu.

```
OSStatus CopyServicesMenuCommandKeys (
   CFArrayRef *outCommandKeyArray
);
```

Parameters

outCommandKeyArray

On return, an array of items in the Services menu that have command key shortcuts associated with them.

Return Value
A result code. See "Carbon Event Manager Result Codes" (page 452).

Discussion
Each array entry is a reference to a CFDictionary, and each dictionary contains information about a single command key shortcut for items in the application's Services menu. Each dictionary contains the following keys: `kHIServicesMenuProviderName`, `kHIServicesMenuItemName`, `kHIServicesMenuCharCode`, and `kHIServicesMenuKeyModifiers`. The array must be released by the caller. The dictionaries do not need to be released because they are released automatically when the array is released.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`CarbonEvents.h`

CopySymbolicHotKeys

Obtains information about symbolic hot keys in the Keyboard preferences pane.

```
OSStatus CopySymbolicHotKeys (
   CFArrayRef *outHotKeyArray
);
```

Parameters

`outHotKeyArray`

An array of dictionaries containing information about the systemwide symbolic hot keys defined in the Keyboard preferences pane, such as the Screen Capture, Universal Access, and Keyboard Navigation keys. The array does not include information about custom, application-specific command keys. You must release the array when you no longer need it. The dictionaries are automatically released when you release the array.

Return Value
A result code. See "Carbon Event Manager Result Codes" (page 452).

Discussion
Each array entry is a reference for a CFDictionary, and each dictionary contains information about a single hot key. There is currently no way to determine which hot key in the Keyboards preference pane corresponds to a specific dictionary. Each dictionary contains the following keys: `kHISymbolicHotKeyCode`, `kHISymbolicHotKeyModifiers`, and `kHISymbolicHotKeyEnabled`. For details, see "Symbolic Hot Key Definitions" (page 376).

The number of hot keys will increase in the future, so do not call this function unnecessarily or in highly performance-sensitive code.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CarbonEvents.h`

CreateEvent

Creates an event.

```
OSStatus CreateEvent (
    CFAllocatorRef inAllocator,
    UInt32 inClassID,
    UInt32 kind,
    EventTime when,
    EventAttributes flags,
    EventRef * outEvent
);
```

Parameters

inAllocator

A reference to the desired memory allocator to use to allocate memory for the event. Pass `NULL` to use the default allocator. See the Base Services documentation for a description of the `CFAllocatorRef` data type.

inClassID

The event class of the event to create.

kind

The event kind of the event to create.

when

The time the event occurred. Pass `0` to specify the current event time (as returned by the `GetCurrentEventTime` (page 263) function).

flags

The event attributes to set. Currently you can pass `kEventAttributeNone` or `kEventAttributeUserEvent`.

outEvent

On return, a reference to the newly created event.

Return Value

A result code. See "Carbon Event Manager Result Codes" (page 452).

Discussion

You can use this function to create your own custom events or to simulate existing events. If you are creating custom events, you must make sure that the event signature (the combination of event class and event kind) does not conflict with any existing events.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CarbonSketch

QTCarbonShell

Declared In

`CarbonEventsCore.h`

CreateMouseTrackingRegion

Creates a mouse tracking region. (Deprecated in Mac OS X v10.4. Use the HIView function `HIViewNewTrackingArea` (page 2473) instead.)

```
OSStatus CreateMouseTrackingRegion (
   WindowRef inWindow,
   RgnHandle inRegion,
   RgnHandle inClip,
   MouseTrackingOptions inOptions,
   MouseTrackingRegionID inID,
   void *inRefCon,
   EventTargetRef inTargetToNotify,
   MouseTrackingRef *outTrackingRef
);
```

Parameters

inWindow

The window to contain the tracking region.

inRegion

The region for which you want to receive mouse entered/exited events.

inClip

The clip region for the `inRegion` region (can be `NULL`).

inOptions

Tracking options that define whether the `inRegion` region is in local or global coordinates.

inID

A signature and ID to uniquely define this tracking region. See `MouseTrackingRegionID` (page 318) for information about the structure of this ID.

inRefCon

A pointer to an application-defined value. You can obtain this value by calling `GetMouseTrackingRegionRefCon` (page 271).

inTargetToNotify

The event target to send the mouse tracking event. If you pass `NULL`, the event target is the owning window specified in `inWindow`.

outTrackingRef

On return, a pointer to the new mouse tracking region.

Return Value

A result code. See "Carbon Event Manager Result Codes" (page 452).

Discussion

`CreateMouseTrackingRegion` allows you to define regions in your window, and the specified event target is notified (using `kEventMouseEntered` or `kEventMouseExited` events) when the mouse cursor interacts with the region. Your application can define any number of regions as long as each has a unique ID. This function is especially useful for creating rollover effects without having to constantly poll the mouse.

If you need to keep track of the state of the mouse (down or up) in a region, you should use `TrackMouseRegion` (page 305), either instead of, or in conjunction with, mouse tracking regions.

Special Considerations

Tracking areas are HIView-based rather than window-based. HIViews support compositing and Quartz, and provide a much easier way to handle user elements in windows. For more details about tracking areas, see the mouse tracking region section in *Carbon Event Manager Programming Guide*. For details about HIViews, see *HIView Programming Guide*.

Availability

Available in Mac OS X v10.2 and later.

Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Related Sample Code
QTCarbonShell

Declared In
`CarbonEvents.h`

CreateTypeStringWithOSType

Converts an `OSType` string to a Core Foundation string.

```
CFStringRef CreateTypeStringWithOSType (
   OSType inType
);
```

Return Value
The Core Foundation string version of the `OSType` string. A return value of `NULL` indicates that an error occurred. See the Base Services documentation for a description of the `CFStringRef` data type.

Discussion
You can use this function to create `CFString` versions of `OSType` data types to pass to the Services Manager. As this is a creation function, you must call `CFRelease` on your Core Foundation string when you no longer need it.

Availability
Available in Mac OS X v10.1 and later.
Not available to 64-bit applications.

Declared In
`CarbonEvents.h`

DisableSecureEventInput

Disables secure event input mode.

```
OSStatus DisableSecureEventInput (
   void
);
```

Return Value
A result code. See "Carbon Event Manager Result Codes" (page 452).

Discussion
When secure event input mode is enabled, keyboard input goes only to the application with keyboard focus and is not echoed to other applications that might be using the event monitor target to watch keyboard input. The `EditText` and `EditUnicodeText` controls automatically enter secure input mode when a password control has focus. If your application implements its own password entry, you should enable secure event input while the user enters text.

This function maintains a count of the number of times that it has been called. Secure event input is not disabled until DisableSecureEventInput has been called the same number of times. Be sure to disable secure event input if your application becomes inactive. If your application crashes, secure event input is automatically disabled if no other application has enabled it.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CarbonEventsCore.h`

DisposeEventComparatorUPP

Disposes of an event comparator UPP.

```
void DisposeEventComparatorUPP (
   EventComparatorUPP userUPP
);
```

Parameters

userUPP
The UPP you want to destroy.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CarbonEventsCore.h`

DisposeEventHandlerUPP

Disposes of an event handler UPP.

```
void DisposeEventHandlerUPP (
   EventHandlerUPP userUPP
);
```

Parameters

userUPP
The event handler UPP you want to destroy.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
BSDLLCTest
HID Explorer

Declared In
`CarbonEventsCore.h`

DisposeEventLoopIdleTimerUPP

Disposes of an event loop idle timer.

```
void DisposeEventLoopIdleTimerUPP (
   EventLoopIdleTimerUPP userUPP
);
```

Availability
Available in Mac OS X v10.2 and later.

Declared In
`CarbonEventsCore.h`

DisposeEventLoopTimerUPP

Disposes of an event loop timer.

```
void DisposeEventLoopTimerUPP (
   EventLoopTimerUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CarbonEventsCore.h`

EnableSecureEventInput

Enables secure event input mode.

```
OSStatus EnableSecureEventInput (
   void
);
```

Return Value
A result code. See "Carbon Event Manager Result Codes" (page 452).

Discussion
When secure event input mode is enabled, keyboard input goes only to the application with keyboard focus and is not echoed to other applications that might be using the event monitor target to watch keyboard input. The `EditText` and `EditUnicodeText` controls automatically enter secure input mode when a password control has focus. If your application implements its own password entry, you should enable secure event input while the user enters text.

This function maintains a count of the number of times that it has been called. Secure event input is not disabled until `DisableSecureEventInput` (page 255) has been called the same number of times. Be sure to disable secure event input if your application becomes inactive. If your application crashes, secure event input is automatically disabled if no other application has enabled it.

Availability
Available in Mac OS X v10.3 and later.

Declared In
CarbonEventsCore.h

EndAppModalStateForWindow

Ends the application-modal state entered using the function BeginAppModalStateForWindow.

```
OSStatus EndAppModalStateForWindow (
   WindowRef inWindow
);
```

Parameters

inWindow

The window you wish to stop acting as application- modal. See the Window Manager documentation for a description of the WindowRef data type.

Return Value
A result code. See "Carbon Event Manager Result Codes" (page 452).

Discussion
This routine ends an app modal state started with BeginAppModalStateForWindow (page 247).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
CarbonEvents.h

FindSpecificEventInQueue

Finds a specific event in the event queue.

```
EventRef FindSpecificEventInQueue (
   EventQueueRef inQueue,
   EventComparatorUPP inComparator,
   void *inCompareData
);
```

Parameters

inQueue

The event queue to search.

inComparator

The comparison function to invoke for each event in the queue. See EventComparatorProcPtr (page 307) for the required format of your comparison function A return value of true from the comparator indicates a match.

inCompareData

The data you wish to pass to your comparison function.

Return Value
An event reference.

Discussion
Returns the first event that matches a comparator function, or `NULL` if no events match.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CarbonEventsCore.h`

FlushEventQueue

Removes all events from the event queue.

```
OSStatus FlushEventQueue (
   EventQueueRef inQueue
);
```

Parameters

inQueue
The event queue to flush.

Return Value
A result code. See "Carbon Event Manager Result Codes" (page 452).

Discussion
Flushes all events from an event queue.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CarbonEventsCore.h`

FlushEventsMatchingListFromQueue

Removes events from the event queue by kind and class.

```
OSStatus FlushEventsMatchingListFromQueue (
   EventQueueRef inQueue,
   ItemCount inNumTypes,
   const EventTypeSpec *inList
);
```

Parameters

inQueue
The event queue to flush events from.

inNumTypes
The number of event kinds to flush.

inList
The list of event classes and kinds to flush from the queue.

Return Value
A result code. See "Carbon Event Manager Result Codes" (page 452).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CarbonEventsCore.h`

FlushSpecificEventsFromQueue

Removes specified events from the event queue.

```
OSStatus FlushSpecificEventsFromQueue (
   EventQueueRef inQueue,
   EventComparatorUPP inComparator,
   void *inCompareData
);
```

Parameters

inQueue
The event queue to flush events from.

inComparator
The comparison function to invoke for each event in the queue. See `EventComparatorProcPtr` (page 307) for the required format of your comparison function. A return value of `true` from the comparator indicates that the event should be flushed.

inCompareData
The data you wish to pass to your comparison function.

Return Value
A result code. See "Carbon Event Manager Result Codes" (page 452).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CarbonEventsCore.h`

GetApplicationEventTarget

Obtains the event target reference for the application.

```
EventTargetRef GetApplicationEventTarget (
   void
);
```

Return Value
An event target reference.

Discussion
Once you obtain this reference, you can send events to the target and install event handlers on it.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
QTCarbonShell
QTMetaData

Declared In
`CarbonEvents.h`

GetCFRunLoopFromEventLoop

Obtains a Core Foundation `CFRunLoop` from an Carbon event loop reference.

```
CFTypeRef GetCFRunLoopFromEventLoop (
   EventLoopRef inEventLoop
);
```

Parameters

`inEventLoop`
The event loop reference to translate.

Return Value
A reference to the CFRunLoop.

Discussion
There isn't necessarily a one-to-one correspondence between Carbon event loops and Core Foundation event loops, so you should use this function instead of simply calling the Core Foundation function `CFRunLoopGetCurrent`.

Availability
Available in Mac OS X v10.1 and later.

Declared In
`CarbonEventsCore.h`

GetControlEventTarget

Obtains the event target reference for the specified control.

```
EventTargetRef GetControlEventTarget (
   ControlRef inControl
);
```

Parameters

`inControl`
The control to return the target for. See the Control Manager documentation for a description of the `ControlRef` data type.

Return Value
An event target reference.

Discussion
Once you obtain this reference, you can send events to the target and install event handlers on it.

Availability
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code
CarbonSketch

Declared In
`CarbonEvents.h`

GetCurrentEventKeyModifiers

Obtains the queue-synchronized keyboard modifier state.

```
UInt32 GetCurrentEventKeyModifiers (
   void
);
```

Return Value
A bit field indicating the queue-synchronized keyboard modifier state. This field is the same as the modifiers field returned in an Event Manager `EventRecord` structure, but it includes only the keyboard modifier flags.

Discussion
The queue-synchronized keyboard modifier state indicates the modifier state according to the event most recently dispatched through an event target. This state may be different from the hardware state obtained using `GetCurrentKeyModifiers` (page 994). For example, say the user invokes a Control-click with the mouse. If the user releases or changes a modifier key before the mouse down event is dispatched, the hardware state reflects the new modifier state, not the one that generated the original mouse event.

The most recently dispatched event may not necessarily be the event that your event handler is handling. For example, if a mouse-down event occurs, and you have a handler for the `kEventWindowHandleContentClick` event that is generated from the mouse-down, then the keyboard modifiers will be those that were attached to the mouse-down. The content-click event itself has a `kEventParamKeyModifiers` parameter, which is copied from the mouse-down event, but `GetCurrentEventKeyModifiers` returns the modifiers from the mouse-down, not from the content-click event, because it was the mouse-down event that was most recently dispatched through the event dispatcher.

Events that are not sent through the event dispatcher target will not update the current event key modifiers. Also, events arriving from outside the application, such as an AppleEvent or an Accessibility event, also will not update the modifiers. If your application modifies its behavior based on modifier state, you should parameterize your core code with the event modifiers, and determine the modifiers based on the origin of the behavior request. For a request that originates directly from user input, you can use `GetCurrentEventKeyModifiers`, but for a request that originates from an AppleEvent or Accessibility event, you would probably use no modifiers. `GetCurrentEventKeyModifiers` gives a more consistent user experience when the user input queue is being remote controlled or manipulated via non-hardware event sources such as speech or AppleEvents; using `GetCurrentEventKeyModifiers` is also much faster than using EventAvail(0, &eventRecord) or GetCurrentKeyModifiers.

`GetCurrentEventKeyModifiers` returns a valid modifier state only if your application is the active application. If your application is not active, then user input events are not flowing through the event dispatcher and the queue-synchronized state is not updated.

Availability
Available in Mac OS X v10.2 and later.

Declared In
CarbonEventsCore.h

GetCurrentEventLoop

Obtains a reference to the current event loop.

```
EventLoopRef GetCurrentEventLoop (
   void
);
```

Return Value
An event loop reference.

Discussion
This function returns the event loop for the current thread. If the current thread is a cooperative thread, the main event loop is returned.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
HID Config Save
HID Explorer

Declared In
CarbonEventsCore.h

GetCurrentEventQueue

Obtains the current event queue.

```
EventQueueRef GetCurrentEventQueue (
   void
);
```

Return Value
An event queue reference.

Discussion
This function obtains the event queue for the current thread. If the current thread is a cooperative thread, the main event queue is returned.

Availability
Available in Mac OS X v10.0 and later.

Declared In
CarbonEventsCore.h

GetCurrentEventTime

Returns the current time since last system startup, in seconds.

```
EventTime GetCurrentEventTime (
   void
);
```

Return Value
EventTime.

Discussion
Returns the current time since last system startup in seconds.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CarbonSketch
QTCarbonShell

Declared In
`CarbonEventsCore.h`

GetEventClass

Returns the class of an event (for example, window, mouse, or keyboard).

```
OSType GetEventClass (
   EventRef inEvent
);
```

Parameters

inEvent
The event in question.

Return Value
The class ID of the event. See "Event Class Constants" (page 322) for more details.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CarbonSketch
HID Calibrator
HID Config Save
HID Explorer
QTCarbonShell

Declared In
`CarbonEventsCore.h`

GetEventDispatcherTarget

Obtains the event target reference for the standard toolbox dispatcher.

```
EventTargetRef GetEventDispatcherTarget (
   void
);
```

Return Value
An event target reference.

Discussion
The standard toolbox dispatcher is the default mechanism for dispatching events to the appropriate event targets. You typically don't need to call this, but some applications may need to pick events off the event queue and call the dispatcher themselves. This allows you to do just that instead of calling `RunApplicationEventLoop` to handle it all.

If desired, you can attach event handlers to the event dispatcher target. Doing so allows you to intercept any events before they can be sent to the appropriate event targets.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CarbonEvents.h`

GetEventKind

Returns the event kind for the specified event.

```
UInt32 GetEventKind (
   EventRef inEvent
);
```

Parameters

`inEvent`
The event in question.

Return Value
The kind of the event.

Discussion
Event kind values overlap in different event classes. For example, `kEventMouseDown` and `kEventAppActivated` both have the same value (`1`). The combination of class and kind determines a unique event signature.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CarbonSketch
HID Calibrator
HID Config Save
HID Explorer
QTCarbonShell

Declared In
`CarbonEventsCore.h`

GetEventMonitorTarget

Obtains an event monitor target.

```
EventTargetRef GetEventMonitorTarget (
   void
);
```

Return Value
An event monitor target.

Discussion
The event monitor target is a special event target used to monitor user input events across all processes. When an event handler is installed on the event monitor target, the Carbon Event Manager examines the EventTypeSpec for user input event types, such as mouse-down, mouse-up, and key-down. It then requests that the WindowServer make copies of any of these events that are sent to any process, and delivers them to the current process. These events are queued into the main thread's event queue and are sent directly to the event handlers installed on the event monitor target during normal event dispatching. Monitored events are not sent through the normal event dispatching path for the current process. Instead, they pass through the event dispatcher target and are sent directly to the event monitor target.Handlers installed on the event monitor target receive events only when the current application is inactive. When the current application is active, all events flow through the event dispatcher target, and no events are sent to the event monitor target. Currently, the event monitor supports the following event kinds: `kEventRawKeyDown`, `kEventRawKeyUp`, `kEventRawKeyRepeat`, `kEventRawKeyModifiersChange`, `kEventMouseDown`, `kEventMouseUp`, `kEventMouseMoved`, `kEventMouseDragged`, `kEventMouseWheelMoved`, `kEventTabletPoint`, and `kEventTabletProximity`.To prevent keyboard events from being passed to other applications, Carbon and Cocoa password-edit-text controls enable a secure input mode while the focus is on the control. Their password-edit-text controls prevent the monitoring event target from being used to capture password keystrokes. For added security, `GetEventMonitorTarget` requires that "Enable access for assistive devices" be checked in the Universal Access preference pane in order to monitor `kEventRawKeyDown`, `kEventRawKeyUp`, and `kEventRawKeyRepeat` events. If this control is not checked, you can still install handlers for these events on the event monitor target, but no events of these types will be sent to your handler. Administrator privileges are required to enable this feature. You can determine whether this control is checked using the `AXAPIEnabled` function in `AXUIElement.h`.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CarbonEvents.h`

GetEventParameter

Obtains a parameter from the specified event.

```
OSStatus GetEventParameter (
   EventRef inEvent,
   EventParamName inName,
   EventParamType inDesiredType,
   EventParamType *outActualType,
   ByteCount inBufferSize,
   ByteCount *outActualSize,
   void *outData
);
```

Parameters

inEvent

The event to get the parameter from.

inName

The symbolic name of the parameter (for example, `kEventParamDirectObject`). The Carbon Event Manager defines a number of constants defining possible parameters.

inDesiredType

The desired type of the parameter (for example, `typeWindowRef`). The Carbon Event Manager automatically uses AppleEvent coercion handlers to convert the data in the event into the desired type, if possible. The Carbon Event Manager defines a number of constants to indicate possible parameter types. Pass `typeWildCard` to request that the data be returned in its original format.

outActualType

The actual type of the parameter (can be `NULL` if you are not interested in receiving this information).

inBufferSize

The size of the output buffer.

outActualSize

The actual size of the data, or `NULL` if you don't want this information.

outData

The pointer to the buffer receiving the parameter data.

Return Value

A result code. See "Carbon Event Manager Result Codes" (page 452).

Discussion

Events often contain additional useful pieces of data, such as the location of a mouse-down event or the window in which an event occurred.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CarbonSketch

HID Calibrator

HID Config Save

HID Explorer

QTCarbonShell

Declared In

`CarbonEventsCore.h`

GetEventRetainCount

Returns the reference count of an event.

```
ItemCount GetEventRetainCount (
   EventRef inEvent
);
```

Return Value
The current reference count for the specified event.

Discussion
When an event is created, its reference count is 1. Calls to `RetainEvent` increment this count; calls to `ReleaseEvent` decrement the count.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CarbonEventsCore.h`

GetEventTime

Returns the time a specific event occurred.

```
EventTime GetEventTime (
   EventRef inEvent
);
```

Parameters

`inEvent`
The event in question.

Return Value
The time the event occurred.

Discussion
Returns the time the event specified occurred, specified as an `EventTime` value, which is a floating point number representing seconds since the last system startup.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTCarbonShell

Declared In
`CarbonEventsCore.h`

GetLastUserEventTime

Returns the last time a user input event arrived in the main event queue of the application.

```
EventTime GetLastUserEventTime (
   void
);
```

Return Value
The time of the last user event.

Discussion
A user input event is something generated by the user, typically a hardware event such as a mouse-click or key-down event.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`CarbonEvents.h`

GetMainEventLoop

Obtains a reference to the main event loop.

```
EventLoopRef GetMainEventLoop (
   void
);
```

Return Value
An event loop reference.

Discussion
The main loop is the event loop for the main application thread.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
BSDLLCTest
QTCarbonShell

Declared In
`CarbonEventsCore.h`

GetMainEventQueue

Obtains the main event queue.

```
EventQueueRef GetMainEventQueue (
   void
);
```

Return Value
An event queue reference.

Discussion
The main queue is the event queue for the main application thread.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTCarbonShell

Declared In
`CarbonEventsCore.h`

GetMenuEventTarget

Obtains an event target reference for the specified menu.

```
EventTargetRef GetMenuEventTarget (
   MenuRef inMenu
);
```

Parameters

inMenu
The menu to return the target for. See the Menu Manager documentation for a description of the `MenuRef` data type.

Return Value
An event target reference.

Discussion
Once you obtain this reference, you can send events to the target and install event handlers on it.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`CarbonEvents.h`

GetMouseTrackingRegionID

(Deprecated in Mac OS X v10.4. Use `HIViewGetTrackingAreaID` (page 2465) instead.)

```
OSStatus GetMouseTrackingRegionID (
   MouseTrackingRef inMouseRef,
   MouseTrackingRegionID *outID
);
```

Special Considerations

Tracking areas are HIView-based rather than window-based. HIViews support compositing and Quartz, and provide a much easier way to handle user elements in windows. For more details about tracking areas, see the mouse tracking region section in *Carbon Event Manager Programming Guide*. For details about HIViews, see *HIView Programming Guide*.

Availability
Available in Mac OS X v10.2 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
CarbonEvents.h

GetMouseTrackingRegionRefCon

Obtains the reference constant for a mouse tracking region. (Deprecated in Mac OS X v10.4. No replacement function. Use HIView-based mouse tracking areas instead.)

```
OSStatus GetMouseTrackingRegionRefCon (
   MouseTrackingRef inMouseRef,
   void **outRefCon
);
```

Parameters

inMouseRef
The mouse tracking region whose reference count you want to obtain.

outRefCon
On return, a handler for the mouse tracking region.

Return Value
A result code. See "Carbon Event Manager Result Codes" (page 452).

Discussion
You use this function to obtain the reference constant you set in the `CreateMouseTrackingRegion` (page 253) function.

Special Considerations

Tracking areas are HIView-based rather than window-based. HIViews support compositing and Quartz, and provide a much easier way to handle user elements in windows. Mouse tracking areas do not support a reference constant. Instead, you can obtain the tracking area ID (using `HIViewGetTrackingAreaID` (page 2465)) and use that as a key to look up extended data in your own tables. For more details about tracking areas, see the mouse tracking region section in *Carbon Event Manager Programming Guide*. For details about HIViews, see *HIView Programming Guide*.

Availability
Available in Mac OS X v10.2 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
CarbonEvents.h

GetNumEventsInQueue

Returns the number of events in the event queue.

```
ItemCount GetNumEventsInQueue (
   EventQueueRef inQueue
);
```

Parameters

inQueue

The event queue to query.

Return Value
The number of items in the queue.

Discussion
Returns the number of events in an event queue.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CarbonEventsCore.h`

GetSymbolicHotKeyMode

Obtains the current hot key mode.

```
OptionBits GetSymbolicHotKeyMode (
   void
);
```

Return Value
The mode request at the top of the hot key mode stack. If there are no mode requests on the stack, this function returns `0` to indicate that hot keys are currently enabled.

Discussion
Unless the "Enable access for assistive devices" checkbox is checked in the Universal Access preference pane, all hot keys are enabled, even if this function returns a nonzero value. This means that hot keys enabled by the caller may be disabled for the current user session if they were disabled by another process.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CarbonEvents.h`

GetUserFocusEventTarget

Obtains the event target reference for the user focus.

```
EventTargetRef GetUserFocusEventTarget (
   void
);
```

Return Value
An event target reference.

Discussion
This event target always references the current user focus. For example, if you install a handler on this target, then your handler will be called whenever an event is sent to the user focus. Keyboard events are always sent to this target.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`CarbonEvents.h`

GetWindowEventTarget

Obtains the event target reference for a specified window.

```
EventTargetRef GetWindowEventTarget (
   WindowRef inWindow
);
```

Parameters

inWindow
The window to return the event target for. See the QuickDraw Manager documentation for a description of the `WindowRef` data type.

Return Value
An event target reference.

Discussion
Once you obtain this reference, you can send events to the target and install an event handler on it.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
QTCarbonShell

Declared In
`CarbonEvents.h`

HIMouseTrackingGetParameters

Obtains information about how mouse tracking loops should behave.

```
OSStatus HIMouseTrackingGetParameters (
   OSType inSelector,
   EventTime *outTime,
   HISize *outDistance
);
```

Parameters

inSelector
The type of information to obtain. Currently, the only supported selector is `kMouseParamsSticky`.

outTime

When sticky mode is select, on return, the maximum time between mouse-down and mouse-up. If the time between events is longer than this value, sticky mode should not be invoked. Pass `NULL` if you don't need this information.

outDistance

When sticky mode is select, on return, the maximum distance between mouse-down and mouse-up. If the distance between events is longer than this value, sticky mode should not be invoked. Pass `NULL` if you don't need this information.

Return Value
A result code. See "Carbon Event Manager Result Codes" (page 452).

Discussion
Mouse tracking loops use different timeouts and wander distances to determine their behavior. This function provides a generic service for requesting this information.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CarbonEvents.h`

InstallEventHandler

Installs an event handler on a specified event target.

```
OSStatus InstallEventHandler (
   EventTargetRef inTarget,
   EventHandlerUPP inHandler,
   ItemCount inNumTypes,
   const EventTypeSpec *inList,
   void *inUserData,
   EventHandlerRef *outRef
);
```

Parameters

inTarget

The event target to register your handler with.

inHandler

A pointer to your event handler function.

inNumTypes

The number of events you are registering for.

inList

A pointer to an array of `EventTypeSpec` entries representing the events you are interested in.

inUserData

The value you pass in this parameter is passed to your event handler function when it is called.

outRef

On return, an event handler reference, which you can use later to remove the handler. You can pass `NULL` if you don't want the reference—when the target is disposed, the handler is disposed as well.

Return Value
A result code. See "Carbon Event Manager Result Codes" (page 452).

Discussion
After being installed, your handler will be called when an event you registered for is sent to the specified event target. Note that CarbonEvents.h defines several macros which you can use for particular event classes. These macros simply combine the appropriate `GetxxxEventTarget` call with `InstallEventHandler`.

- `InstallApplicationEventHandler`
- `InstallWindowEventHandler`
- `InstallControlEventHandler`
- `InstallMenuEventHandler`
- `InstallHIObjectEventHandler` (in Mac OS X v10.2 and later)
- `HIViewInstallEventHandler` (in Mac OS X v10.2 and later)

Be sure to remove the event handler when you no longer need it by calling `RemoveEventHandler` (page 294). Doing so is especially important if the handler calls code that may disappear. For example, if a plugin installs an event handler and is later removed without removing the handler, the system may attempt to call back to the now nonexistent plugin code.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTCarbonShell
QTMetaData

Declared In
`CarbonEventsCore.h`

InstallEventLoopIdleTimer

Installs a timer that fires only when there is no user activity.

```
OSStatus InstallEventLoopIdleTimer (
   EventLoopRef inEventLoop,
   EventTimerInterval inDelay,
   EventTimerInterval inInterval,
   EventLoopIdleTimerUPP inTimerProc,
   void *inTimerData,
   EventLoopTimerRef *outTimer
);
```

Parameters

inEventLoop
The event loop to add the timer.

inFireDelay
The delay before first firing this timer, in seconds. In Mac OS X v10.3 and earlier, this delay must be greater than zero. In Mac OS X v10.4 and later, the delay must be greater than or equal to zero. You cannot pass `kEventDurationForever`.

inInterval
The timer interval, in seconds. Pass `0` or `kEventDurationForever` for a one-shot timer.

inTimerProc
The function to call when the timer fires.

inTimerData
Data to pass to the timer function when called.

outTimer
A reference to the newly installed timer.

Return Value
A result code. See "Carbon Event Manager Result Codes" (page 452).

Discussion
An idle timer is the same as a standard event timer except that it fires only when no user events are being received. That is, if the system receives no user events for the `inFireDelay` delay time, the idle timer fires, and will continue to fire at the rate specified by `inInterval`. If the user begins activity again, the timer stops and resets. For example, you could use an idle timer in a search engine to begin a search 2 seconds after the user stops typing in the search text field.

The callback function for idle timers takes an additional parameter that tells the callback the user status. See `EventLoopIdleTimerProcPtr` (page 309) and "Idle Timer Event Constants" (page 416) for more information.

Be sure to dispose of the timer when you no longer need it by calling `RemoveEventLoopTimer` (page 295). Doing so is especially important if your timer calls code that may no longer exist. For example, if a plugin creates a timer that calls back to it, the timer will attempt to call it even after the plugin is removed.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`CarbonEventsCore.h`

InstallEventLoopTimer

Installs a timer.

```
OSStatus InstallEventLoopTimer (
   EventLoopRef inEventLoop,
   EventTimerInterval inFireDelay,
   EventTimerInterval inInterval,
   EventLoopTimerUPP inTimerProc,
   void *inTimerData,
   EventLoopTimerRef *outTimer
);
```

Parameters

inEventLoop
The event loop to add the timer.

inFireDelay

The delay before first firing this timer, in seconds. In Mac OS X v10.3 and earlier, the delay must be greater than zero. In Mac OS X v10.4, the delay can be greater than or equal to zero.

In Mac OS X and CarbonLib 1.5 and later, you may pass `kEventDurationForever` to stop the timer from firing at all until `SetEventLoopTimerNextFireTime` is used to start it; in earlier versions of CarbonLib, to achieve the same effect, just pass zero and then immediately call `SetEventLoopTimerNextFireTime( timer, (kEventDurationForever )` before returning control to your event loop.

inInterval

The timer interval, in seconds. Pass `0` or (in Mac OS X and CarbonLib 1.5 and later) `kEventDurationForever` for a one-shot timer.

inTimerProc

The function to call when the timer fires.

inTimerData

Data to pass to the timer function when called.

outTimer

A reference to the newly installed timer.

Return Value
A result code. See "Carbon Event Manager Result Codes" (page 452).

Discussion
Installs a timer onto the event loop specified. The timer can either fire once or repeatedly at a specified interval depending on the parameters passed to this function. It executes at task level and should not be confused with Time Manager tasks or any other interrupt-level callback. This means you can call toolbox functions, allocate memory, and draw without worrying about consequences. When a timer fires, it calls the callback you specified when the timer was installed.

Timers in general have two uses: as a timeout mechanism and as a periodic task. An everyday example of using a timer for a timeout might be a light that goes out if no motion is detected in a room for 5 minutes. For this, you might install a timer which will fire in 5 minutes. If motion is detected, you would reset the timer fire time and let the clock start over. If no motion is detected for the full 5 minutes, the timer will fire and you could power off the light. A periodic timer is one that fires at regular intervals (say every second or so). You might use such a timer to blink the insertion point in your editor, and so on.

One advantage of timers is that you can install the timer right from the code that wants the time. For example, the standard editable text control can install a timer to blink the cursor when it's active, meaning that the Control Manager function `IdleControls` is a no-op for that control and doesn't need to be called. When the control is inactive, it removes its timer and doesn't waste CPU time in that state.

Currently, if you do decide to draw when your timer is called, be sure to save and restore the current port so that calling your timer doesn't inadvertently change the port out from under someone.

Be sure to dispose of the timer when you no longer need it by calling `RemoveEventLoopTimer` (page 295). Doing so is especially important if your timer calls code that may no longer exist. For example, if a plugin creates a timer that calls back to it, the timer will attempt to call it even after the plugin is removed.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
BSDLLCTest
HID Config Save

HID Explorer
QTCarbonShell

Declared In
`CarbonEventsCore.h`

InstallStandardEventHandler

Installs the standard event handler for the specified target.

```
OSStatus InstallStandardEventHandler (
   EventTargetRef inTarget
);
```

Parameters

inTarget

The event target for which you want to install the standard handler.

Return Value
A result code. See "Carbon Event Manager Result Codes" (page 452).

Discussion
Currently you can install the standard handler only for window event targets. To install the standard application handler, you must call `RunApplicationEventLoop` (page 297).

Note that events may also have default behaviors or standard definitions which define how an event is handled if you choose not to handle it yourself. Default behavior is the response that occurs whenever you choose not to handle the event, whether or not you have a standard handler installed. Standard definition behavior defines how an event is handled based on that element's standard definition. For example, the standard menu definition provides some default responses for menu events you do not handle. However if you are using your own custom definition, you cannot assume that these default responses will occur.

You can also install the standard handler for a window event target by calling `ChangeWindowAttributes` to set the `kWindowStandardHandlerAttribute` window attribute on the window.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`CarbonEventsCore.h`

InvokeEventComparatorUPP

Calls an event comparator function through a UPP.

```
Boolean InvokeEventComparatorUPP (
   EventRef inEvent,
   void *inCompareData,
   EventComparatorUPP userUPP
);
```

Parameters

inEvent

The event to compare against.

inCompareData

Application-specific data. Typically this is the data you passed when calling `FindSpecificEventInQueue` (page 258) or `FlushSpecificEventsFromQueue` (page 260).

userUPP

A UPP to the comparator function you want to invoke.

Return Value
Returns `true` if the comparator function indicates a match with the specified event, `false` otherwise.

Discussion
You call this function only if you need to invoke your event comparator callback yourself. In most cases you don't need to call this function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CarbonEventsCore.h`

InvokeEventHandlerUPP

Calls an event handler through a UPP.

```
OSStatus InvokeEventHandlerUPP (
   EventHandlerCallRef inHandlerCallRef,
   EventRef inEvent,
   void *inUserData,
   EventHandlerUPP userUPP
);
```

Return Value
A result code. See "Carbon Event Manager Result Codes" (page 452).

Discussion
You use this function only if you need to invoke an event handler yourself. In most cases you don't need to call this function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CarbonEventsCore.h`

InvokeEventLoopIdleTimerUPP

Calls an event loop idle timer through a UPP.

```
void InvokeEventLoopIdleTimerUPP (
   EventLoopTimerRef inTimer,
   EventLoopIdleTimerMessage inState,
   void *inUserData,
   EventLoopIdleTimerUPP userUPP
);
```

Discussion
You use this function only if you need to invoke an idle event timer callback yourself. In most cases you don't need to call this function.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`CarbonEventsCore.h`

InvokeEventLoopTimerUPP

Calls an event loop timer through a UPP.

```
void InvokeEventLoopTimerUPP (
   EventLoopTimerRef inTimer,
   void *inUserData,
   EventLoopTimerUPP userUPP
);
```

Discussion
You use this function only if you need to invoke an event timer callback yourself. In most cases you don't need to call this function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CarbonEventsCore.h`

IsEventInMask

Determines whether an event reference matches a `WaitNextEvent`-style event mask.

```
Boolean IsEventInMask (
   EventRef inEvent,
   EventMask inMask
);
```

Parameters

inEvent

The event reference to check against the event mask.

inMask
The mask to consider. See the Event Manager documentation for a description of the `EventMask` data type.

Return Value
A Boolean whose value is `TRUE` if the event was in the mask; otherwise, `FALSE`.

Discussion
This is a companion function for `ConvertEventRefToEventRecord` (page 249), and is provided as a convenience function to help you if there are places in your application where you want to check an `EventRef` to see if it matches a classic `EventMask` bitfield. If the event matches, the function returns true.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`CarbonEvents.h`

IsEventInQueue

Determines whether an event is in a particular queue.

```
Boolean IsEventInQueue (
   EventQueueRef inQueue,
   EventRef inEvent
);
```

Parameters
inQueue
The queue to check.
inEvent
The event in question.

Return Value
Returns true if the specified event is posted to a queue.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CarbonEventsCore.h`

IsMouseCoalescingEnabled

Indicates whether mouse coalescing is enabled.

```
Boolean IsMouseCoalescingEnabled (
   void
);
```

Return Value
A Boolean whose value is `TRUE` if mouse coalescing is on; otherwise, `FALSE`.

Discussion
If mouse coalescing is enabled, intermediate mouse movement events are merged into the most recent event, so that only one mouse moved or mouse dragged event is in the event queue at any time. For example, when the user moves the mouse across the screen, more mouse moved events are generated than most applications care about. Rather than place all these events in the queue (which would probably slow down the application), the Carbon Event Manager first checks to see if a mouse moved event already exists. If a mouse moved event already exists, that event is updated with the position and delta information from the more recently-generated event.

Availability
Available in Mac OS X v10.1 and later.
Not available to 64-bit applications.

Declared In
`CarbonEvents.h`

IsSecureEventInputEnabled

Determines whether secure event input mode is enabled.

```
Boolean IsSecureEventInputEnabled (
   void
);
```

Return Value
A Boolean whose value is `TRUE` if secure event input mode is enabled; otherwise, `FALSE`.

Discussion
This function determines whether secure event input is enabled by *any* process, not just the current process. Secure event input may be disabled in the current process but enabled in another process, in which case, this function returns `TRUE`.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`CarbonEventsCore.h`

IsUserCancelEventRef

Returns whether the specified event indicates the user wishes to cancel an operation.

```
Boolean IsUserCancelEventRef (
   EventRef event
);
```

Return Value
A Boolean value indicating whether the event is a user cancel event.

Discussion
Tests the event given to see whether the event represents a user cancel event. Currently this is defined to be either the escape key being pressed or command-period being pressed.

Availability
Available in Mac OS X v10.0 and later.

Declared In
CarbonEvents.h

MoveMouseTrackingRegion

(Deprecated in Mac OS X v10.4. No replacement function. Use HIView-based tracking areas instead.)

```
OSStatus MoveMouseTrackingRegion (
   MouseTrackingRef inMouseRef,
   SInt16 deltaH,
   SInt16 deltaV,
   RgnHandle inClip
);
```

Special Considerations

Tracking areas are HIView-based rather than window-based. HIViews support compositing and Quartz, and provide a much easier way to handle user elements in windows. HIView-based mouse tracking areas move automatically when the HIView moves. For more details about tracking areas, see the mouse tracking region section in *Carbon Event Manager Programming Guide*. For details about HIViews, see *HIView Programming Guide*.

Availability
Available in Mac OS X v10.2 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
CarbonEvents.h

MoveWindowMouseTrackingRegions

(Deprecated in Mac OS X v10.4. No replacement function. Use HIView-based tracking areas instead.)

```
OSStatus MoveWindowMouseTrackingRegions (
   WindowRef inWindow,
   OSType inSignature,
   SInt16 deltaH,
   SInt16 deltaV,
   RgnHandle inClip
);
```

Special Considerations

Tracking areas are HIView-based rather than window-based. HIViews support compositing and Quartz, and provide a much easier way to handle user elements in windows. HIView-based mouse tracking areas move automatically when the HIView moves. For more details about tracking areas, see the mouse tracking region section in *Carbon Event Manager Programming Guide*. For details about HIViews, see *HIView Programming Guide*.

Availability
Available in Mac OS X v10.2 and later.

Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
CarbonEvents.h

NewEventComparatorUPP

Creates an event comparator UPP.

```
EventComparatorUPP NewEventComparatorUPP (
   EventComparatorProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your event comparator callback function.

Return Value
The UPP for your callback function.

Discussion
When calling FindSpecificEventInQueue (page 258) or FlushSpecificEventsFromQueue (page 260), you must pass a universal procedure pointer (UPP) to your event comparator instead of a standard procedure pointer.

Availability
Available in Mac OS X v10.0 and later.

Declared In
CarbonEventsCore.h

NewEventHandlerUPP

Creates an event handler UPP.

```
EventHandlerUPP NewEventHandlerUPP (
   EventHandlerProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your event handler.

Return Value
The UPP for your event handler.

Discussion
When registering your event handler with InstallEventHandler (page 274), you must pass a universal procedure pointer (UPP) to your event handler instead of a standard procedure pointer.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
BSDLLCTest
CarbonCocoa_PictureCursor
CarbonSketch
HID Config Save
QTMetaData

Declared In
`CarbonEventsCore.h`

NewEventLoopIdleTimerUPP

Creates an event loop idle timer UPP.

```
EventLoopIdleTimerUPP NewEventLoopIdleTimerUPP (
   EventLoopIdleTimerProcPtr userRoutine
);
```

Parameters

`userRoutine`
A pointer to your idle event timer callback function.

Return Value
The UPP for your event loop idle timer callback function.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`CarbonEventsCore.h`

NewEventLoopTimerUPP

Creates an event loop timer UPP.

```
EventLoopTimerUPP NewEventLoopTimerUPP (
   EventLoopTimerProcPtr userRoutine
);
```

Parameters

`userRoutine`
A pointer to your event timer callback function.

Return Value
The UPP for your event timer callback function.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
BSDLLCTest
HID Config Save
HID Explorer

Declared In
CarbonEventsCore.h

PopSymbolicHotKeyMode

Removes a hot key mode request from the hot key mode stack.

```
void PopSymbolicHotKeyMode (
   void *inToken
);
```

Parameters

inToken

The hot key mode token that was returned by a previous call to PushSymbolicHotKeyMode (page 287).

Discussion
If the request is the topmost request on the stack, the hot key mode changes to the next request on the stack. If there are other mode requests on top of this request on the stack, the mode does not change.

Availability
Available in Mac OS X v10.4 and later.

Declared In
CarbonEvents.h

PostEventToQueue

Adds an event to the specified event queue.

```
OSStatus PostEventToQueue (
   EventQueueRef inQueue,
   EventRef inEvent,
   EventPriority inPriority
);
```

Parameters

inQueue

The event queue to post the event onto.

inEvent

The event to post.

inPriority

The priority of the event. See "Event Priority Constants" (page 326) for a list of possible constants to pass.

Return Value
A result code. See "Carbon Event Manager Result Codes" (page 452).

Discussion
Posts an event to the queue specified and increments its retain count. This automatically wakes up the event loop of the thread the queue belongs to.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTCarbonShell

Declared In
`CarbonEventsCore.h`

ProcessHICommand

Sends a command to the command chain.

```
OSStatus ProcessHICommand (
   const HICommand *inCommand
);
```

Return Value
A result code. See "Carbon Event Manager Result Codes" (page 452)

Discussion
`ProcessHICommand` is a convenience function for sending a "process command" event through the command chain (for example, from menu to user focus to application). The command event is sent initially to either a menu (if the command represents a menu command) or the current user focus. If the function returns `eventNotHandledErr`, the command was not handled by any element in the chain.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`CarbonEvents.h`

PushSymbolicHotKeyMode

Sets a new mode for enabling and disabling symbolic hot keys.

```
void * PushSymbolicHotKeyMode (
   OptionBits inOptions
);
```

Parameters
inOptions
The requested symbolic hot key mode. For details, see "Hot Key Constants" (page 376).

Return Value
A token that is passed to `PopSymbolicHotKeyMode` (page 286) to remove this mode request when it is no longer needed.

Discussion
The Event Manager maintains a stack of hot key modes that have been requested by calls to this function. The most recently pushed mode is the mode that is currently in use.

Disabling hot keys can significantly affect the usability of Mac OS X. For this reason, applications are allowed to disable hot keys only if the "Enable access for assistive devices" checkbox is checked in the Universal Access preference pane. If this checkbox is not checked when this function is called, the requested hot key mode is pushed onto the mode stack and a valid token is returned but the actual hot key mode is unchanged.

If the frontmost application pushes a new hot key mode that disables any hot keys, the new mode is active only while the application remains the frontmost application. If the application is deactivated or exits without re-enabling hot keys, the hot key mode automatically reverts to the previous mode.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`CarbonEvents.h`

QuitApplicationEventLoop

Terminates the application event loop.

```
void QuitApplicationEventLoop (
   void
);
```

Discussion
This function is used to quit the `RunApplicationEventLoop` (page 297) function. Typically, your application doesn't need to call this. If your application has the Quit menu item tagged with the `kHICommandQuit` menu command ID, the toolbox will automatically call this for your application, automatically terminating your event loop. If your application wants to do pre-processing before the event loop exits, it should intercept either the `kHICommandQuit` menu command, or the `kEventAppQuit` event.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
QTCarbonShell

Declared In
`CarbonEvents.h`

QuitAppModalLoopForWindow

Quits the application-modal state for a window.

```
OSStatus QuitAppModalLoopForWindow (
   WindowRef inWindow
);
```

Parameters

inWindow
The window that is leaving the modal state. See the Window Manager documentation for a description of the `WindowRef` data type.

Return Value
A result code. See "Carbon Event Manager Result Codes" (page 452).

Discussion
This function is used to quit a currently running call to `RunAppModalLoopForWindow` (page 298) (that is, it terminates a modal loop). Typically you call this from a handler you have installed on the modal window in question when the user clicks the appropriate button (Ok, Cancel, and so on).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`CarbonEvents.h`

QuitEventLoop

Causes a specific event loop to terminate.

```
OSStatus QuitEventLoop (
   EventLoopRef inEventLoop
);
```

Parameters
inEventLoop
The event loop to terminate.

Return Value
A result code. See "Carbon Event Manager Result Codes" (page 452).

Discussion
Usage of this is similar to `WakeUpProcess`, in that it causes the event loop specified to return immediately (as opposed to timing out). Typically you use this call in conjunction with `RunCurrentEventLoop` (page 298).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CarbonEventsCore.h`

ReceiveNextEvent

Waits for the next event of a specified type.

```
OSStatus ReceiveNextEvent (
   ItemCount inNumTypes,
   const EventTypeSpec *inList,
   EventTimeout inTimeout,
   Boolean inPullEvent,
   EventRef *outEvent
);
```

Parameters
inNumTypes
The number of event types to wait for (0 if any event should cause this function to return).

inList
The list of event types we are waiting for (pass `NULL` if any event should cause this function to return).

inTimeout
The time to wait before returning (passing `kEventDurationForever` is preferred).

inPullEvent
Pass `true` for this parameter to remove the next matching event from the queue.

outEvent
A pointer to the next event that matches the list passed in. If you passed `true` in the `inPullEvent` parameter, the event is owned by you, and you should release it when done.

Return Value
A result indicating whether an event was received, the timeout expired, or the current event loop was quit. See "Carbon Event Manager Result Codes" (page 452) for possible values.

Discussion
This function tries to fetch the next event of a specified type. If no events in the event queue match, this function will run the current event loop until an event that matches arrives, or the timeout expires. Except for timers firing, your application is blocked waiting for events to arrive when inside this function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CarbonEventsCore.h`

RegisterEventHotKey

Registers a global hot key.

```
OSStatus RegisterEventHotKey (
   UInt32 inHotKeyCode,
   UInt32 inHotKeyModifiers,
   EventHotKeyID inHotKeyID,
   EventTargetRef inTarget,
   OptionBits inOptions,
   EventHotKeyRef *outRef
);
```

Parameters

inHotKeyCode
The virtual key code of the hot key you want to register.

inHotKeyModifiers
The keyboard modifiers to look for. In Mac OS X v10.2 and earlier, if you do not specify a modifier key, this function returns `paramErr`. In Mac OS X v10.3 and later, passing `0` does not cause an error.

inHotKeyID
The application-specified hot key ID. You will receive this ID in the `kEventHotKeyPressed` event as the direct object parameter.

inTarget
The target to notify when the hot key is pressed.

inOptions
Currently unused. You must pass `0`.

outRef

On return, a reference to the new hot key. You need this reference if you later wish to unregister it.

Return Value
A result code. See "Carbon Event Manager Result Codes" (page 452).

Discussion
This function registers a global hot key based on the virtual key code and modifiers you pass in. When the user enters the hot-key combination, a `kEventHotKeyPressed` event is sent to the target you specified. Only one such combination can exist for the current application (that is, multiple entities in the same application cannot register for the same hot key combination). The same hot key can, however, be registered by multiple applications. This means that multiple applications can potentially be notified when a particular hot key is requested. This might not necessarily be desirable, but it is how it works at present.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CarbonEvents.h`

RegisterToolboxObjectClass

Registers events to be associated with a toolbox object. (Deprecated in Mac OS X v10.4. Use the HIObject function `HIObjectRegisterSubclass` (page 2335) instead.)

```
OSStatus RegisterToolboxObjectClass (
   CFStringRef inClassID,
   ToolboxObjectClassRef inBaseClass,
   ItemCount inNumEvents,
   const EventTypeSpec *inEventList,
   EventHandlerUPP inEventHandler,
   void *inEventHandlerData,
   ToolboxObjectClassRef *outClassRef
);
```

Parameters

inClassID

The class ID of the toolbox object you want to register. This value should be a Core Foundation string in the form `com.myCorp.myApp.myWidget`.

inBaseClass

The class reference of this toolbox object's base class. Pass `NULL` if there is no base class.

inNumEvents

The number of events to register for this object class.

inEventList

An array of events you want to register for this object class. You define these events just as you would for any other Carbon event handler.

inEventHandler

A universal procedure pointer to the event handler for this object class.

inEventHandlerData

Any application-specific data you want passed to your event handler when it is called.

outClassRef

On return, `outClassRef` contains a reference to the new object class. You use this value in your custom definition specification (such as a `ControlDefSpec` or `WindowDefSpec`) to define your new object class.

Return Value
A result code. See "Carbon Event Manager Result Codes" (page 452).

Discussion
You use this function to register event handlers to implement what were formerly called defproc messages; that is, you can use toolbox objects in place of older custom window, menu, and control definitions.

Special Considerations

HIObject allows you to create subclasses that you can use for creating custom HIViews. HIViews support compositing and Quartz and provide an easier way to handle user elements in windows. Use `HIObjectRegisterSubclass` to create custom HIObjects and HIViews. See *HIView Programming Guide* for more details.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`CarbonEvents.h`

ReleaseEvent

Releases, and possibly disposes of, the specified event.

```
void ReleaseEvent (
   EventRef inEvent
);
```

Parameters

inEvent

The event to release.

Discussion
This function decrements the reference count of an event. If the reference count reaches 0, the event is disposed.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CarbonSketch

QTCarbonShell

Declared In
`CarbonEventsCore.h`

ReleaseMouseTrackingRegion

Releases a mouse tracking region. (Deprecated in Mac OS X v10.4. Use `HIViewDisposeTrackingArea` (page 2450) instead.)

```
OSStatus ReleaseMouseTrackingRegion (
   MouseTrackingRef inMouseRef
);
```

Parameters

inMouseRef

The mouse tracking region to release.

Return Value

A result code. See "Carbon Event Manager Result Codes" (page 452).

Discussion

`ReleaseMouseTrackingRegion` decreases the reference count for the region. If the reference count drops to zero, the mouse tracking region is disposed.

Special Considerations

Tracking areas are HIView-based rather than window-based. HIViews support compositing and Quartz, and provide a much easier way to handle user elements in windows. For more details about tracking areas, see the mouse tracking region section in *Carbon Event Manager Programming Guide*. For details about HIViews, see *HIView Programming Guide*.

Availability

Available in Mac OS X v10.2 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

QTCarbonShell

Declared In

`CarbonEvents.h`

ReleaseWindowMouseTrackingRegions

(Deprecated in Mac OS X v10.4. No replacement function. Use HIView-based tracking areas instead.)

```
OSStatus ReleaseWindowMouseTrackingRegions (
   WindowRef inWindow,
   OSType inSignature
);
```

Special Considerations

Tracking areas are HIView-based rather than window-based. HIViews support compositing and Quartz, and provide a much easier way to handle user elements in windows. If you need to release multiple tracking areas at once, you should keep track of them in your own data structures and release each one. For more details about tracking areas, see the mouse tracking region section in *Carbon Event Manager Programming Guide*. For details about HIViews, see *HIView Programming Guide*.

Availability

Available in Mac OS X v10.2 and later.

Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`CarbonEvents.h`

RemoveEventFromQueue

Removes an event from the event queue.

```
OSStatus RemoveEventFromQueue (
   EventQueueRef inQueue,
   EventRef inEvent
);
```

Parameters

inQueue
The queue to remove the event from.

inEvent
The event to remove.

Return Value
A result code. See "Carbon Event Manager Result Codes" (page 452).

Discussion
Removes the given event from the queue on which it was posted and decrements its retain count.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CarbonEventsCore.h`

RemoveEventHandler

Removes the specified event handler.

```
OSStatus RemoveEventHandler (
   EventHandlerRef inHandlerRef
);
```

Parameters

inHandlerRef
The handler ref to remove (returned in a call to `InstallEventHandler`). After you call this function, the handler reference is considered invalid and can no longer be used.

Return Value
A result code. See "Carbon Event Manager Result Codes" (page 452).

Discussion
Removes an event handler from the event target to which it was bound.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTCarbonShell

Declared In
`CarbonEventsCore.h`

RemoveEventLoopTimer

Removes the specified timer.

```
OSStatus RemoveEventLoopTimer (
   EventLoopTimerRef inTimer
);
```

Parameters

inTimer
The timer to remove.

Return Value
A result code. See "Carbon Event Manager Result Codes" (page 452).

Discussion
Removes a timer that was previously installed by a call to `InstallEventLoopTimer` (page 276) or `InstallEventLoopIdleTimer` (page 275). You call this function when you are done using a timer.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
BSDLLCTest
HID Config Save
HID Explorer
QTCarbonShell

Declared In
`CarbonEventsCore.h`

RemoveEventTypesFromHandler

Removes events from an installed event handler.

```
OSStatus RemoveEventTypesFromHandler (
   EventHandlerRef inHandlerRef,
   ItemCount inNumTypes,
   const EventTypeSpec *inList
);
```

Parameters

inHandlerRef
The event handler to remove the events from.

inNumTypes
The number of events to remove.

inList
A pointer to an array of `EventTypeSpec` structures.

Return Value
A result code. See "Carbon Event Manager Result Codes" (page 452).

Discussion
You can use this function dynamically change which events you want your handler to respond to.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CarbonEventsCore.h`

RetainEvent

Increments the reference count of an event.

```
EventRef RetainEvent (
   EventRef inEvent
);
```

Parameters
inEvent
The event to retain.

Return Value
The event reference you passed in the `inEvent` parameter. A value of `NULL` indicates an error condition.

Discussion
The `RetainEvent` function increments an event's reference count by 1. You can use this function to ensure that an event is never disposed of by another event handler. However, if the event system or some other event handler changes the event, those changes are reflected in your reference. To create a separate, unique copy of an event, use `CopyEvent` (page 250) instead.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CarbonEventsCore.h`

RetainMouseTrackingRegion

Retains a mouse tracking region. (Deprecated in Mac OS X v10.4. No replacement function. Use HIView-based tracking areas instead.)

```
OSStatus RetainMouseTrackingRegion (
   MouseTrackingRef inMouseRef
);
```

Parameters
inMouseRef
The mouse tracking region to retain.

Return Value
A result code. See "Carbon Event Manager Result Codes" (page 452).

Discussion
`RetainMouseTrackingRegion` increases the reference count for the region.

Special Considerations

Tracking areas are HIView-based rather than window-based. HIViews support compositing and Quartz, and provide a much easier way to handle user elements in windows. Mouse tracking areas do not have a retain/release semantic, so there is no direct replacement for `RetainMouseTrackingRegion`. For more details about tracking areas, see the mouse tracking region section in *Carbon Event Manager Programming Guide*. For details about HIViews, see *HIView Programming Guide*.

Availability
Available in Mac OS X v10.2 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`CarbonEvents.h`

RunApplicationEventLoop

Runs the application event loop.

```
void RunApplicationEventLoop (
   void
);
```

Discussion
This function is used as the main event loop for a Carbon Event-based application. Once entered, this function waits for events to arrive and dispatches them to your event handlers automatically.

Note that calling `RunApplicationEventLoop` also installs the standard application handler, which provides standard handler responses for menu and application events.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
BSDLLCTest
HID Config Save
HID Explorer
QTCarbonShell
QTMetaData

Declared In
`CarbonEvents.h`

RunAppModalLoopForWindow

Puts the window in an application-modal state.

```
OSStatus RunAppModalLoopForWindow (
   WindowRef inWindow
);
```

Parameters

inWindow

The window you wish to behave modally. See the Window Manager documentation for a description of the `WindowRef` data type.

Return Value

A result code. See "Carbon Event Manager Result Codes" (page 452).

Discussion

This function is used as a replacement for the Dialog Manager function `ModalDialog` to drive a Carbon Event-based modal dialog. Once called, this function will not exit until `QuitAppModalLoopForWindow` (page 288) is called.

While in the modal state, the standard toolbox dispatcher processes events only for the modal window and any that are above it (that is, closer to the front). This feature allows you to create stacked modal dialogs, if desired.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`CarbonEvents.h`

RunCurrentEventLoop

Executes the event loop in the current thread.

```
OSStatus RunCurrentEventLoop (
   EventTimeout inTimeout
);
```

Parameters

inTimeout

The time to wait until returning (can be `kEventDurationForever`).

Return Value

A result code. See "Carbon Event Manager Result Codes" (page 452).

Discussion

This function "runs" the event loop, returning only if aborted or the timeout specified is reached. The event loop is mostly blocked while in this function, occasionally waking up to fire timers or pick up events. The typical use of this function is to cause the current thread to wait for some operation to complete, most likely on another thread of execution.

Availability

Available in Mac OS X v10.0 and later.

Declared In
CarbonEventsCore.h

SendEventToEventTarget

Sends an event to the specified event target.

```
OSStatus SendEventToEventTarget (
   EventRef inEvent,
   EventTargetRef inTarget
);
```

Parameters

inEvent
The event to send.

inTarget
The target to send it to.

Return Value
A result code. See "Carbon Event Manager Result Codes" (page 452).

Discussion
If you are creating your own events, you can dispatch them immediately to an event target by calling this function.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CarbonSketch
QTCarbonShell

Declared In
CarbonEventsCore.h

SendEventToEventTargetWithOptions

Sends an event to the specified event target with propagation options.

```
OSStatus SendEventToEventTargetWithOptions (
   EventRef inEvent,
   EventTargetRef inTarget,
   OptionBits inOptions
);
```

Parameters

inEvent
The event to send.

inTarget
The target to send it to.

inOptions

Options indicating how the event should be propagated. See "Event Target Propagation Options" (page 326) for a list of possible values.

Return Value
A result code. See "Carbon Event Manager Result Codes" (page 452).

Discussion
This function is identical to `SendEventToEventTarget` (page 299) except that you can specify how the event is propagated using options.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`CarbonEventsCore.h`

SetEventLoopTimerNextFireTime

Sets the next time that the specified timer will fire.

```
OSStatus SetEventLoopTimerNextFireTime (
   EventLoopTimerRef inTimer,
   EventTimerInterval inNextFire
);
```

Parameters

inTimer

The timer whose firing time you want to set.

inNextFire

The interval from the current time to wait until firing the timer again.

Return Value
A result code. See "Carbon Event Manager Result Codes" (page 452).

Discussion
This function is used to "reset" a timer. It controls the next time the timer fires. This will override any interval you might have set. For example, if you have a timer that fires every second, and you call this function setting the next time to 5 seconds from now, the timer will sleep for 5 seconds, then fire. The timer will then resume its one second interval. This function acts as if you removed the timer and reinstalled it with a new first-fire delay.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CarbonEventsCore.h`

SetEventParameter

Sets a parameter associated with a particular event.

```
OSStatus SetEventParameter (
   EventRef inEvent,
   EventParamName inName,
   EventParamType inType,
   ByteCount inSize,
   const void *inDataPtr
);
```

Parameters

inEvent

The event to set the data for.

inName

The symbolic name of the parameter.

inType

The symbolic type of the parameter.

inSize

The size of the parameter data.

inDataPtr

A pointer to the parameter data.

Return Value

A result code. See "Carbon Event Manager Result Codes" (page 452).

Discussion

When creating events, you may want to specify additional event-related information, such as the mouse location or the window in which the event occurred. To set these you call `SetEventParameter`, specifying the type and value for the desired parameter.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CarbonCocoa_PictureCursor

CarbonSketch

HID Calibrator

QTCarbonShell

Declared In

`CarbonEventsCore.h`

SetEventTime

Sets the event time for a given event.

```
OSStatus SetEventTime (
   EventRef inEvent,
   EventTime inTime
);
```

Parameters

inEvent

The event in question.

inTime
The new time.

Return Value
A result code. See "Carbon Event Manager Result Codes" (page 452).

Discussion
This function allows you to set the time of a given event, if you so desire. In general, you would never use this function, except for those special cases where you reuse an event from time to time instead of creating a new event each time.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CarbonEventsCore.h`

SetMouseCoalescingEnabled

Turns mouse coalescing on or off.

```
OSStatus SetMouseCoalescingEnabled (
   Boolean inNewState,
   Boolean *outOldState
);
```

Parameters
inNewState
Pass `true` to turn mouse coalescing on, `false` otherwise.

outOldState
A Boolean value indicating the previous mouse coalescing state (that is, before you called this function to set it). You can use this value if you want to save the previous state for later restoration. If you don't need this state information, pass `NULL`.

Return Value
A result code. See "Carbon Event Manager Result Codes" (page 452).

Discussion
See `IsMouseCoalescingEnabled` (page 281) for a definition of mouse coalescing.

Availability
Available in Mac OS X v10.1 and later.
Not available to 64-bit applications.

Declared In
`CarbonEvents.h`

SetMouseTrackingRegionEnabled

(Deprecated in Mac OS X v10.4. No replacement function. Use HIView-based tracking areas instead.)

```
OSStatus SetMouseTrackingRegionEnabled (
   MouseTrackingRef inMouseRef,
   Boolean inEnabled
);
```

Special Considerations

Tracking areas are HIView-based rather than window-based. HIViews support compositing and Quartz, and provide a much easier way to handle user elements in windows. To disable tracking areas, you can either delete the tracking area or ignore kEventControlTrackingAreaEntered events. For more details about tracking areas, see the mouse tracking region section in *Carbon Event Manager Programming Guide*. For details about HIViews, see *HIView Programming Guide*.

Availability
Available in Mac OS X v10.2 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`CarbonEvents.h`

SetWindowMouseTrackingRegionsEnabled

(Deprecated in Mac OS X v10.4. Use HIView-based tracking areas instead.)

```
OSStatus SetWindowMouseTrackingRegionsEnabled (
   WindowRef inWindow,
   OSType inSignature,
   Boolean inEnabled
);
```

Special Considerations

Tracking areas are HIView-based rather than window-based. HIViews support compositing and Quartz, and provide a much easier way to handle user elements in windows. To disable tracking areas, you can either delete the tracking area or ignore kEventControlTrackingAreaEntered events. For more details about tracking areas, see the mouse tracking region section in *Carbon Event Manager Programming Guide*. For details about HIViews, see *HIView Programming Guide*.

Availability
Available in Mac OS X v10.2 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`CarbonEvents.h`

TrackMouseLocation

Tracks the mouse, blocking your application when there is no activity.

```
OSStatus TrackMouseLocation (
   GrafPtr inPort,
   Point *outPt,
   MouseTrackingResult *outResult
);
```

Parameters

inPort

The graphics port to consider for mouse coordinates. You can pass `NULL` for this parameter to indicate the current port. The mouse location is returned in terms of local coordinates of this port. See the QuickDraw Manager documentation for a description of the `GrafPtr` data type.

outPt

On exit, a pointer to the mouse location from the last mouse event that caused this function to exit.

outResult

On exit, a pointer to a value representing what kind of event was received that cause the function to exit, such as `kMouseTrackingMouseReleased`.

Return Value

A result code. See "Carbon Event Manager Result Codes" (page 452).

Discussion

Once entered, this function waits for certain mouse events (move, mouse down, mouse up). When one of these events occurs, the function returns and tells the caller what happened and where the mouse is currently located. While there is no activity, the current event loop is run, effectively blocking the current thread (save for any timers that fire). This helps to minimize CPU usage when there is nothing going on.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`CarbonEvents.h`

TrackMouseLocationWithOptions

Tracks the mouse with additional options.

```
OSStatus TrackMouseLocationWithOptions (
   GrafPtr inPort,
   OptionBits inOptions,
   EventTimeout inTimeout,
   Point *outPt,
   UInt32 *outModifiers,
   MouseTrackingResult *outResult
);
```

Parameters

inPort

The graphics port (`GrafPort`) to consider for mouse coordinates. You can pass `NULL` for this parameter to indicate the current port. The mouse location is returned in global coordinates. See the QuickDraw Manager documentation for a description of the `GrafPtr` data type.

inOptions

The only option supported by this function at present is the option to have the toolbox leave mouse up events in the queue, rather than pulling them (which is the default). See "Mouse Tracking Option Constant" (page 397) for more information.

inTimeout

The amount of time to wait for an event. If no events arrive within this time, `kMouseTrackingTimedOut` is returned in `outResult`.

outPt

On return, a pointer to the mouse location from the last mouse event that caused this function to exit. If a timeout or key modifiers changed event caused this function to exit, the current mouse position at the time is returned.

outModifiers

On return, a pointer to the most recent state of the keyboard modifiers.

outResult

On return, a pointer to a value indicating the kind of event that caused the function to exit, such as `kMouseTrackingMouseReleased`.

Return Value
A result code. See "Carbon Event Manager Result Codes" (page 452).

Discussion
Once entered, this function waits for certain mouse events (move, mouse down, mouse up). When one of these events occurs, the function returns and tells the caller what happened and where the mouse is currently located. While there is no activity, the current event loop is run, effectively blocking the current thread (save for any timers that fire). This helps to minimize CPU usage when there is nothing going on.

Availability
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code
CarbonSketch

Declared In
`CarbonEvents.h`

TrackMouseRegion

Tracks the mouse within a region.

```
OSStatus TrackMouseRegion (
   GrafPtr inPort,
   RgnHandle inRegion,
   Boolean *ioWasInRgn,
   MouseTrackingResult *outResult
);
```

Parameters

inPort

The graphics port to consider for mouse coordinates. You can pass `NULL` for this parameter to indicate the current port. See the QuickDraw Manager documentation for a description of the `GrafPtr` data type.

inRegion

The region to consider. This should be in the coordinates of the port you passed to `inPort`. See the QuickDraw Manager documentation for a description of the `RgnHandle` data type.

ioWasInRgn

On entering the region, this parameter should be set to `true` if the mouse is currently inside the region passed in `inRegion`, or `false` if the mouse is currently outside the region. On exit, this parameter is updated to reflect the current reality. For example, if the `outResult` parameter returns `kMouseTrackingMouseExited`, `ioWasInRgn` will be set to `false` when this function exits. Because it is updated from within, you should only need to set this yourself before the first call to this function in your tracking loop.

outResult

On exit, a pointer to a value indicating the kind of event that caused the function to exit, such as `kMouseTrackingMouseEntered`.

Return Value
A result code. See "Carbon Event Manager Result Codes" (page 452).

Discussion
This function is largely identical to `TrackMouseLocation` (page 303). The difference between `TrackMouseLocation` and `TrackMouseRegion` is that `TrackMouseRegion` only returns when the mouse enters or exits a specified region that you pass in to the function, as opposed to whenever the mouse moves (it also returns for mouse up/down events). This is useful if you don't need to know intermediate mouse events, but rather just if the mouse enters or leaves an area.

Note that in some cases you may prefer to register one or more special mouse tracking regions and receive events when the mouse enters or exits the region. However, this alternative method does not automatically inform you about mouse up and mouse down actions. See `CreateMouseTrackingRegion` (page 253) for more details.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`CarbonEvents.h`

UnregisterEventHotKey

Unregisters a global hot key.

```
OSStatus UnregisterEventHotKey (
   EventHotKeyRef inHotKey
);
```

Parameters

inHotKey

The event hot key reference of the hot key you want to unregister.

Return Value
A result code. See "Carbon Event Manager Result Codes" (page 452).

Discussion
Unregisters a global hot key that was previously registered with the function `RegisterEventHotKey` (page 290). You do not need to unregister a hot key when your application terminates; the system takes care of that for you. You can use this function if the user changes a hot key for something in your application—you would unregister the previous key and register your new key.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CarbonEvents.h`

UnregisterToolboxObjectClass

Unregisters events for a given toolbox object class (Deprecated in Mac OS X v10.4. Use the HIObject function `HIObjectUnregisterClass` (page 2340) instead.)

```
OSStatus UnregisterToolboxObjectClass (
   ToolboxObjectClassRef inClassRef
);
```

Parameters
inClassRef
A reference to the toolbox object class you want to unregister.

Return Value
A result code. See "Carbon Event Manager Result Codes" (page 452).

Special Considerations

HIObject allows you to create subclasses that you can use for creating custom HIViews. HIViews support compositing and Quartz and provide an easier way to handle user elements in windows. Use `HIObjectUnregisterClass` to unregister custom HIObjects and HIViews. See *HIView Programming Guide* for more details.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`CarbonEvents.h`

Callbacks

EventComparatorProcPtr

Defines the format of your event comparator callback function.

```
typedef Boolean (*EventComparatorProcPtr) (
   EventRef inEvent,
   void * inCompareData
);
```

If you name your function `MyEventComparatorProc`, you would declare it like this:

```
Boolean MyEventComparatorProc (
   EventRef inEvent,
   void * inCompareData
);
```

Parameters

inEvent

The event to compare.

inCompareData

The data you passed to `FindSpecificEventInQueue` (page 258) or `FlushSpecificEventsFromQueue` (page 260).

Return Value

A Boolean value indicating whether the event matches (`true`) or not (`false`).

Discussion

You use this callback function when searching the event queue using functions such as `FindSpecificEventInQueue` (page 258).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CarbonEventsCore.h`

EventHandlerProcPtr

Defines the format of your event handler.

```
typedef OSStatus (*EventHandlerProcPtr) (
   EventHandlerCallRef inHandlerCallRef,
   EventRef inEvent,
   void * inUserData
);
```

If you name your function `MyEventHandlerProc`, you would declare it like this:

```
OSStatus MyEventHandlerProc (
   EventHandlerCallRef inHandlerCallRef,
   EventRef inEvent,
   void * inUserData
);
```

Parameters

inHandlerCallRef

A reference to the current handler call chain. This is passed to your handler so that you can call `CallNextEventHandler` if you need to.

inEvent

The event that triggered this call.

inUserData

The application-specific data you passed in to `InstallEventHandler` (page 274).

Return Value

A result code. See "Carbon Event Manager Result Codes" (page 452). Returning `noErr` indicates you handled the event. Returning `eventNotHandledErr` indicates you did not handle the event and perhaps other handlers in the calling chain should take action.

Discussion

Callback to install on an event target.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CarbonEventsCore.h`

EventLoopIdleTimerProcPtr

Defines the format of your idle timer callback function.

```
typedef void (*EventLoopIdleTimerProcPtr) (
   EventLoopTimerRef inTimer,
   EventLoopIdleTimerMessage inState,
   void * inUserData
);
```

If you name your function `MyEventLoopTimerProc`, you would declare it like this:

```
void MyEventLoopTimerProc (
   EventLoopTimerRef inTimer,
   EventLoopIdleTimerMessage inState,
   void * inUserData
);
```

Parameters

inTimer

The timer that fired.

inState

The state of the idle period. See "Idle Timer Event Constants" (page 416) for a list of possible constants you can receive.

inUserData

The application-specific data you passed into `InstallEventLoopIdleTimer` (page 275).

Discussion

Called when an idle timer fires.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`CarbonEventsCore.h`

EventLoopTimerProcPtr

Defines the format of your event loop timer callback function.

```
typedef void (*EventLoopTimerProcPtr) (
   EventLoopTimerRef inTimer,
   void * inUserData
);
```

If you name your function `MyEventLoopTimerProc`, you would declare it like this:

```
void MyEventLoopTimerProc (
   EventLoopTimerRef inTimer,
   void * inUserData
);
```

Parameters

inTimer
The timer that fired.

inUserData
The data you passed into `InstallEventLoopTimer` (page 276).

Discussion
Called when a timer fires.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CarbonEventsCore.h`

Data Types

EventClassID

Represents an event class ID.

```
typedef UInt32 EventClassID;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CarbonEvents.h`

EventComparatorUPP

Represents a universal procedure pointer to an event comparator callback function.

```
typedef EventComparatorProcPtr EventComparatorUPP
```

EventHandlerCallRef

Indicates the next handler in the event handler calling hierarchy.

```
typedef struct OpaqueEventHandlerCallRef * EventHandlerCallRef;
```

Discussion
This structure is passed to your event handler, which can then choose to pass control to the next handler in the calling hierarchy (such as a standard event handler). Doing so is a convenient way to add pre- or post-processing to the standard event handler. See the `CallNextEventHandler` (page 247) function for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CarbonEventsCore.h`

EventHandlerUPP

Represents a universal procedure pointer for an event handler callback function.

```
typedef EventHandlerProcPtr EventHandlerUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CarbonEventsCore.h`

EventLoopTimerUPP

Represents a universal procedure pointer for an event timer callback function.

```
typedef EventLoopTimerProcPtr EventLoopTimerUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CarbonEventsCore.h`

EventLoopIdleTimerUPP

Represents a universal procedure pointer for an idle event timer callback function.

```
typedef EventLoopIdleTimerProcPtr EventLoopIdleTimerUPP;
```

Availability
Available in Mac OS X v10.2 and later.

Declared In
`CarbonEventsCore.h`

EventHandlerRef

Represents an installed event handler.

```
typedef struct OpaqueEventHandlerRef * EventHandlerRef;
```

Discussion
You receive an event handler reference when you install your handler using `InstallEventHandler` (page 274). You can use this reference when calling functions such as `RemoveEventHandler` (page 294) and `AddEventTypesToHandler` (page 246).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CarbonEventsCore.h`

EventHotKeyID

Represents the ID of a global hot key.

```
struct EventHotKeyID {
    OSType signature;
    UInt32 id;
};
typedef struct EventHotKeyID EventHotKeyID;
```

Discussion
You register a hot key using the `RegisterEventHotKey` (page 290) function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CarbonEvents.h`

EventHotKeyRef

Represents a registered global hot key.

```
typedef struct OpaqueEventHotKeyRef * EventHotKeyRef;
```

Discussion
You register a hot key using the `RegisterEventHotKey` (page 290) function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CarbonEvents.h`

EventLoopIdleTimerMessage

Represents an idle timer message.

```
typedef UInt16 EventLoopIdleTimerMessage;
```

Discussion
Sent to idle timer callback functions to indicate the current idle status. See "Idle Timer Event Constants" (page 416) for a list of possible values.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`CarbonEventsCore.h`

EventLoopRef

Represents an event loop.

```
typedef struct OpaqueEventLoopRef * EventLoopRef;
```

Discussion
The `EventLoopRef` type represents an event loop, which is the conceptual entity that you run to fetch events from hardware and other sources and also fires timers that might be installed with `InstallEventLoopTimer` (page 276) or `InstallEventLoopIdleTimer` (page 275). The term "run" is a bit of a misnomer, as the event loop's goal is to stay as blocked as possible to minimize CPU usage for the current application. The event loop is run implicitly through calls to functions like `ReceiveNextEvent` (page 289), `RunApplicationEventLoop` (page 297), or even the Classic Event Manager function `WaitNextEvent`. It can also be run explicitly through a call to `RunCurrentEventLoop` (page 298). Each preemptive thread can have an event loop. Cooperative threads share the main thread's event loop.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CarbonEventsCore.h`

EventLoopTimerRef

Represents an installed event timer.

```
typedef struct __EventLoopTimer * EventLoopTimerRef;
```

Discussion
The `EventLoopTimerRef` type represents a timer function that is called either once or at regular intervals. See `InstallEventLoopTimer` (page 276) and `InstallEventLoopIdleTimer` (page 275) for more information about event timers.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CarbonEventsCore.h`

EventParamName

Represents an event parameter constant.

```
typedef OSType EventParamName;
```

Discussion
You specify an event parameter name when calling `GetEventParameter` (page 266) or `SetEventParameter` (page 300). Parameter names indicate what kind of event parameter you want to set or obtain (such as `kEventParamDirectObject`). For specific types, see the tables of event parameters and types associated with each class of events (for example, Table 3-8 (page 392)).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CarbonEventsCore.h`

EventParamType

Represents an event parameter type constant.

```
typedef OSType EventParamType;
```

Discussion
You specify an event parameter type when calling `GetEventParameter` (page 266) or `SetEventParameter` (page 300). Event parameter types indicate the data type of the parameter you want to set or obtain (such as `typeBoolean`). For specific types, see the tables of event parameters and types associated with each class of events (for example, Table 3-11 (page 424)).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CarbonEventsCore.h`

EventQueueRef

Represents an event queue.

```
typedef struct OpaqueEventQueueRef * EventQueueRef;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CarbonEventsCore.h`

EventRef

Represents an opaque data structure that identifies individual events.

```
typedef struct OpaqueEventRef * EventRef;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CarbonEventsCore.h`

EventTargetRef

Represents an event target (such as a window or control).

```
typedef struct OpaqueEventTargetRef * EventTargetRef;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CarbonEventsCore.h`

EventTime

Represents a time value in seconds. An absolute `EventTime` value is seconds since boot time.

```
typedef double EventTime;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CarbonEventsCore.h`

EventTimeout

Represents a timeout interval, in seconds.

```
typedef EventTime EventTimeout;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
CarbonEventsCore.h

EventTimerInterval

Specifies the period of an event timer, in seconds.

```
typedef EventTime EventTimerInterval;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
CarbonEventsCore.h

EventType

Represents an event type.

```
typedef UInt32 EventType;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
cssmspi.h

EventTypeSpec

Describes the class and kind of an event.

```
struct EventTypeSpec {
    UInt32 eventClass;
    UInt32 eventKind;
};
typedef struct EventTypeSpec EventTypeSpec;
```

Discussion
This structure is used to specify an event. Typically, you pass a static array of `EventTypeSpec` structures into functions such as `InstallEventHandler` (page 274), as well as functions such as `FlushEventsMatchingListFromQueue` (page 259).

Availability
Available in Mac OS X v10.0 and later.

Declared In
CarbonEventsCore.h

HICommand

Represents a command event; this structure has been superseded by the `HICommandExtended` structure.

```
struct HICommand {
    UInt32 attributes
    UInt32 commandID
    struct {
        MenuRef menuRef;
        MenuItemIndex menuItemIndex;
    } menu;
};
typedef struct HICommand HICommand;
```

Fields

`attributes`
Attributes of the command event.

`commandID`
The command ID of the command event.

`menuRef`
A reference to the menu containing the HICommand.

`menuItemIndex`
The index number of the menu item containing the HICommand.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CarbonEvents.h`

HICommandExtended

Represents an extended command event.

```
struct HICommandExtended {
    UInt32 attributes;
    UInt32 commandID;
    union {
        controlRef control;
        windowRef window;
        struct {
            MenuRef menuRef;
            MenuItemIndex menuItemIndex;
        } menu;
    } source;
};
typedef struct HICommandExtended HICommandExtended;
```

Fields

`attributes`
Attributes of the command event. The value of this field (indicating whether the source of the command event is a control, window, or menu) determines what reference is stored in the union. See "Command Event Source Constants" (page 350) for a list of possible values.

`commandID`
The command ID of the command event.

`controlRef`
The control that produced the command event.

`windowRef`
The window that produced the command event.

`menuRef`
A reference to the menu containing the command event.

`menuItemIndex`
The index number of the menu item containing the command event.

Discussion
The `HICommandExtended` structure was introduced in Mac OS X v10.2 and CarbonLib 1.6. Because the `HICommand` and `HICommandExtended` structures are exactly the same size and have the same fields at the same offsets, you can use an `HICommandExtended` structure at runtime while running on any version of CarbonLib or Mac OS X. The only difference is that the `HICommandExtended` structure has a union that allows you to get type-safe access to the source object. The originator of the command determines whether the structure actually contains a ControlRef, WindowRef, MenuRef, or nothing at all. You can determine what is in the command by checking the `attributes` field.

For example, in Mac OS X v10.2 and later, when a push button is clicked, the Control Manager sends a command event containing the push button's command ID, sets the `kHICommandFromControl` bit in the `attributes` field, and stores the button's ControlRef in the `source.control` field. In Mac OS X v10.0 and v10.1, the same command event is sent, but the `kHICommandFromControl`, `kHICommandFromMenu`, and `kHICommandFromWindow` attributes are not set, and the `source.controlRef`, `source.menu.menuRef` and `source.windowRef` fields are not initialized, respectively. Your code can use an `HICommandExtended` structure when running on Mac OS X v10.0 and v10.1 as long as it first checks the `kHICommandFromControl`, `kHICommandFromMenu`, and `kHICommandFromWindow` attributes before accessing the `source.control`, `menu.control`, and `window.control` fields.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`CarbonEvents.h`

MouseTrackingRef

Represents a mouse tracking region

```
typedef struct OpaqueMouseTrackingRef * MouseTrackingRef;
```

Discussion
Use `CreateMouseTrackingRegion` (page 253) to create a mouse tracking region.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`CarbonEvents.h`

MouseTrackingRegionID

Represents a mouse tracking region identifier.

```
struct MouseTrackingRegionID {
    OSType signature;
    SInt32 id;
};
typedef struct MouseTrackingRegionID MouseTrackingRegionID;
```

Fields

`signature`

A four-character code (such as `'moof'`)that uniquely identifies the application that owns this mouse tracking region.

`id`

An integer that identifies the mouse tracking region in this application.

Discussion

Each application can register multiple mouse tracking regions as long as each region has a unique ID. Use `CreateMouseTrackingRegion` (page 253) to create a mouse tracking region.

Availability

Available in Mac OS X v10.2 and later.

Not available to 64-bit applications.

Declared In

`CarbonEvents.h`

TabletPointRec

Defines a tablet point structure.

```
struct TabletPointRec {
    SInt32 absX;
    SInt32 absY;
    SInt32 absZ;
    UInt16 buttons;
    UInt16 pressure;
    SInt16 tiltX;
    SInt16 tiltY;
    UInt16 rotation;
    SInt16 tangentialPressure;
    UInt16 deviceID;
    SInt16 vendor1;
    SInt16 vendor2;
    SInt16 vendor3;
};
typedef struct TabletPointRec TabletPointRec;
typedef TabletPointRec TabletPointerRec;
```

Fields

`absX`

The x-coordinate of the pointer, in tablet space (at full tablet resolution).

`absY`

The y-coordinate of the pointer, in tablet space (at full tablet resolution).

`absZ`

The z-coordinate of the pointer, in tablet space (at full tablet resolution).

`buttons`
: The buttons that are pressed. This integer is interpreted as a bit field, with bit 0 indicating the first button, bit 1 the second button, and so on. A value of `1` indicates that the button is down.

`pressure`
: The scaled pressure value. The pressure value is in the range 0 to 65535.

`tiltX`
: The scaled tilt x value. The tilt value is in the range -32767 to 32767.

`tiltY`
: The scaled tilt y value. The tilt value is in the range -32767 to 32767.

`rotation`
: The device rotation as a fixed-point value in a 10.6 format.

`tangentialPressure`
: The tangential pressure on the device. This pressure is in the range -32767 to 32767.

`deviceID`
: A unique system-assigned device ID. This ID matches the device ID you receive for the `kEventTabletProximity` event.

`vendor1`
: A vendor-defined value.

`vendor2`
: A vendor-defined value.

`vendor3`
: A vendor-defined value.

Discussion
You receive this structure in the `kEventParamTabletPointRec` parameter for the `kEventTabletPoint` event.

Availability
Available in Mac OS X v10.1 and later.

Declared In
`CarbonEvents.h`

TabletProximityRec

Defines a tablet proximity structure.

```
struct TabletProximityRec {
    UInt16 vendorID;
    UInt16 tabletID;
    UInt16 pointerID;
    UInt16 deviceID;
    UInt16 systemTabletID;
    UInt16 vendorPointerType;
    UInt32 pointerSerialNumber;
    UInt64 uniqueID;
    UInt32 capabilityMask;
    UInt8 pointerType;
    UInt8 enterProximity;
};
typedef struct TabletProximityRec TabletProximityRec;
```

Fields

`vendorID`
A vendor-defined ID. This value is typically the USB vendor ID.

`tabletID`
A vendor-defined ID for the tablet. This value is typically the USB product ID for the tablet.

`pointerID`
A vendor-defined ID for the pointing device (for example, a pen).

`deviceID`
A unique system-assigned device ID. This ID matches the device ID you receive for the `kEventTabletPoint` event.

`systemTabletID`
A system-assigned unique tablet ID.

`vendorPointerType`
A vendor-defined pointer type.

`pointerSerialNumber`
A vendor-defined serial number for the pointing device.

`uniqueID`
A vendor-defined ID for this pointer.

`capabilityMask`
A bit mask representing the capabilities of this device.

`pointerType`
The type of pointing device.

`enterProximity`
The proximity value. A nonzero value indicates that the pointer is entering the tablet proximity; zero indicates that it is leaving.

Discussion

You receive this structure in the `kEventParamTabletProximityRec` parameter for the `kEventTabletProximity` event.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CarbonEvents.h`

ToolboxObjectClassRef

Represents a toolbox object class.

```
typedef struct OpaqueToolboxObjectClassRef * ToolboxObjectClassRef;
```

Discussion
Typically you use toolbox object classes to specify custom user interface elements. See `RegisterToolboxObjectClass` (page 291) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CarbonEvents.h`

Constants

Basic Event Constants

Event Class Constants

Define constants for specifying event classes.

```
typedef UInt32 EventClass;
enum {
    kEventClassMouse = 'mous',
    kEventClassKeyboard = 'keyb',
    kEventClassTextInput = 'text',
    kEventClassApplication = 'appl',
    kEventClassAppleEvent = 'eppc',
    kEventClassMenu = 'menu',
    kEventClassWindow = 'wind',
    kEventClassControl = 'cntl',
    kEventClassCommand = 'cmds',
    kEventClassTablet = 'tblt',
    kEventClassVolume = 'vol ',
    kEventClassAppearance = 'appm',
    kEventClassService = 'serv',
    kEventClassToolbar = 'tbar',
    kEventClassToolbarItem = 'tbit',
    kEventClassToolbarItemView = 'tbiv',
    kEventClassAccessibility = 'acce'.
    kEventClassSystem = 'macs',
    kEventClassInk = 'ink ',
    kEventClassTSMDocumentAccess = 'tdac'
};
```

Constants

`kEventClassMouse`
Events related to the mouse (mouse down/up/moved).
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventClassKeyboard`
Events related to the keyboard.
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventClassTextInput`
Events related to text input (by keyboard or by input method).
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventClassApplication`
Application-level events (launch, quit, and so on).
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventClassAppleEvent`
Apple Events.
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventClassMenu`
Menu-related events.
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventClassWindow`
Window-related events.
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventClassControl`
Control-related events.
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventClassCommand`
Command events (HICommands).
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventClassTablet`
Events related to tablet input.
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventClassVolume`
Events related to File Manager volumes.
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventClassAppearance`
Events related to the Appearance Manager.
Available in Mac OS X v10.1 and later.
Declared in `CarbonEvents.h`.

`kEventClassService`
Events related to the Services Manager.
Available in Mac OS X v10.1 and later.
Declared in `CarbonEvents.h`.

`kEventClassToolbar`
Events related to the toolbar (not the toolbar window class).
Available in Mac OS X v10.2 and later.
Declared in `CarbonEvents.h`.

`kEventClassToolbarItem`
Events related to toolbar items.
Available in Mac OS X v10.2 and later.
Declared in `CarbonEvents.h`.

`kEventClassToolbarItemView`
Events related to toolbar item views.
Available in Mac OS X v10.3 and later.
Declared in `CarbonEvents.h`.

`kEventClassAccessibility`
Events related to application accessibility features.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

`kEventClassSystem`
Events related to the system.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`kEventClassInk`
Events related to ink.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`kEventClassTSMDocumentAccess`
Events related to Text Services Manager document access.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

Discussion
Event classes specify broad categories of events, grouped according to the object they are associated with. Within an event class are specific event types.

Event Attributes

Define constants for special attributes of an event.

```
typedef UInt32 EventAttributes;
enum {
    kEventAttributeNone = 0,
    kEventAttributeUserEvent = (1 << 0),
    kEventAttributeMonitored= 1 << 3
};
```

Constants
`kEventAttributeNone`
No attributes.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEventsCore.h`.

`kEventAttributeUserEvent`
An event generated in response to a user action.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEventsCore.h`.

`kEventAttributeMonitored`
An event that was not originally targeted to this process but has been provided to this process because an event handler for this event type has been installed on the event monitoring target. The event dispatcher sends events with this attribute directly to the event monitor target.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEventsCore.h`.

Discussion
You use these attributes only if you are creating your own events.

Event Priority Constants

Define event priority constants.

```
typedef SInt16 EventPriority;
enum {
    kEventPriorityLow = 0,
    kEventPriorityStandard = 1,
    kEventPriorityHigh = 2
};
```

Constants

`kEventPriorityLow`
Lowest priority. Currently only window update events are posted at this priority.
Available in Mac OS X v10.0 and later.
Declared in `CarbonEventsCore.h`.

`kEventPriorityStandard`
Normal priority of events. Most events are standard priority.
Available in Mac OS X v10.0 and later.
Declared in `CarbonEventsCore.h`.

`kEventPriorityHigh`
Highest priority.
Available in Mac OS X v10.0 and later.
Declared in `CarbonEventsCore.h`.

Discussion
These values define the relative priority of an event, and are used when posting events with `PostEventToQueue` (page 286). In general events are pulled from the queue in order of first posted to last posted. These priorities are a way to alter that behavior when posting events. You can post a standard priority event and then a high priority event, and the high priority event will be pulled from the queue first.

Event Target Propagation Options

Define options for the `SendEventToEventTargetWithOptions` function.

```
enum {
    kEventTargetDontPropagate = (1 << 0),
    kEventTargetSendToAllHandlers = (1 << 1)
};
```

Constants

`kEventTargetDontPropagate`

Do not propagate this event to any other event target. That is, even if the handler returns `eventNotHandledErr`, the event is not propagated up the handler chain. When passed an event sent with this option, `CallNextEventHandler` only calls other event handlers installed on the current event target; it does not propagate the event to other event targets.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEventsCore.h`.

`kEventTargetSendToAllHandlers`

Send this event to all event targets in the handler chain, regardless of any handler's return value. For example, if sent to a control, after returning, the event is sent to the owning window and then to the application. Note that the Carbon Event Manager keeps track of the strongest result code when progressing up the handler chain. That is, if the first handler returns `noErr`, and the second handler returns `eventNotHandledErr`, the result returned is `noErr`.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEventsCore.h`.

Event Queue Constants

Define constants for specifying how events should be handled on the queue.

```
enum {
    kEventLeaveInQueue = false,
    kEventRemoveFromQueue = true
};
```

Constants

`kEventLeaveInQueue`

Leave the event on the queue after examining.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEventsCore.h`.

`kEventRemoveFromQueue`

Remove the event from the queue after examining.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEventsCore.h`.

Discussion

When calling a function such as `ReceiveNextEvent` (page 289), you can specify whether to leave the event on the queue (peeking at it to determine its class, type, and so on), or to pull it before dispatching it to an event handler.

Direct Object Parameter

Define the direct object parameter.

```
enum {
    kEventParamDirectObject = '----'
};
```

Constants

`kEventParamDirectObject`

Type varies depending on event.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEventsCore.h`.

Discussion

The direct object parameter is usable for a wide variety of events. It defines the "object the event acted upon or within." For example, for window events, the direct object parameter returns a reference (that is a `WindowRef`) to the window in which the event occurred.

Event Target Parameter

Define constants for a special event target parameter and its type, that you can set for any created event.

```
enum {
    kEventParamPostTarget = 'ptrg',
    typeEventTargetRef = 'etrg'
};
```

Constants

`kEventParamPostTarget`

Specifies the target the event should be sent to. Instead of sending an event directly to a given target, you can set this parameter and post the event onto the event queue.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

`typeEventTargetRef`

The parameter type for `kEventParamPostTarget`.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

Object Reference Parameters and Types

Define constants for parameters that specify various objects and their types.

```
enum {
    kEventParamWindowRef = 'wind',
    kEventParamGrafPort = 'graf',
    kEventParamDragRef = 'drag',
    kEventParamMenuRef = 'menu',
    kEventParamEventRef = 'evnt',
    kEventParamControlRef = 'ctrl',
    kEventParamRgnHandle = 'rgnh',
    kEventParamEnabled = 'enab',
    kEventParamDimensions = 'dims',
    kEventParamBounds = 'boun',
    kEventParamAvailableBounds = 'avlb',
    kEventParamAEEventID = keyAEEventID,
    kEventParamAEEventClass = keyAEEventClass,
    kEventParamCGContextRef = 'cntx',
    kEventParamDeviceDepth = 'devd',
    kEventParamDeviceColor = 'devc',
    kEventParamMutableArray = 'marr',
    kEventParamResult = 'ansr',
    kEventParamMinimumSize = 'mnsz',
    kEventParamMaximumSize = 'mxsz',
    kEventParamAttributes = 'attr',
    kEventParamReason = 'why?',
    kEventParamTransactionID = 'trns',
    kEventParamGDevice = 'gdev',
    kEventParamIndex = 'indx',
    kEventParamUserData = 'usrd',
    kEventParamShape = 'shap',
    typeWindowRef = 'wind',
    typeGrafPtr = 'graf',
    typeGWorldPtr = 'gwld',
    typeDragRef = 'drag',
    typeMenuRef = 'menu',
    typeControlRef = 'ctrl',
    typeCollection = 'cltn',
    typeQDRgnHandle = 'rgnh',
    typeOSStatus = 'osst',
    typeCFIndex = 'cfix',
    typeCGContextRef = 'cntx',
    typeHIPoint = 'hipt',
    typeHISize = 'hisz',
    typeHIRect = 'hirc',
    typeHIShapeRef = 'shap',
    typeVoidPtr = 'void',
    typeGDHandle = 'gdev'
};
```

Constants

`kEventParamWindowRef`

A window reference. (typeWindowRef)

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventParamGrafPort`

typeGrafPtr

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventParamDragRef`
typeDragRef
Available in Mac OS X v10.0 and later.
Declared in `CarbonEventsCore.h`.

`kEventParamMenuRef`
typeMenuRef
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamEventRef`
typeEventRef
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamControlRef`
typeControlRef
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamRgnHandle`
typeQDRgnHandle
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamEnabled`
typeBoolean
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamDimensions`
typeQDPoint
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamBounds`
typeQDRectangle
Available in Mac OS X v10.3 and later.
Declared in `CarbonEvents.h`.

`kEventParamAvailableBounds`
typeQDRectangle
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamAEEventID`
typeType
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamAEEventClass`
typeType
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamCGContextRef`
typeCGContextRef
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamDeviceDepth`
typeShortInteger
Available in Mac OS X v10.1 and later.
Declared in `CarbonEvents.h`.

`kEventParamDeviceColor`
typeBoolean
Available in Mac OS X v10.1 and later.
Declared in `CarbonEvents.h`.

`kEventParamMutableArray`
typeCFMutableArrayRef
Available in Mac OS X v10.2 and later.
Declared in `CarbonEvents.h`.

`kEventParamResult`
Any type, depending on the event
Available in Mac OS X v10.2 and later.
Declared in `CarbonEvents.h`.

`kEventParamMinimumSize`
typeHISize
Available in Mac OS X v10.2 and later.
Declared in `CarbonEvents.h`.

`kEventParamMaximumSize`
typeHISize
Available in Mac OS X v10.2 and later.
Declared in `CarbonEvents.h`.

`kEventParamAttributes`
typeUInt32
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamReason`
typeUInt32
Available in Mac OS X v10.3 and later.
Declared in `CarbonEvents.h`.

`kEventParamTransactionID`
typeUInt32
Available in Mac OS X v10.3 and later.
Declared in `CarbonEvents.h`.

`kEventParamGDevice`
typeGDHandle
Available in Mac OS X v10.3 and later.
Declared in `CarbonEvents.h`.

`kEventParamIndex`
typeCFIndex
Available in Mac OS X v10.3 and later.
Declared in `CarbonEvents.h`.

`kEventParamUserData`
typeVoidPtr
Available in Mac OS X v10.3 and later.
Declared in `CarbonEvents.h`.

`kEventParamShape`
typeHIShapeRef
Available in Mac OS X v10.4 and later.
Declared in `CarbonEvents.h`.

`typeWindowRef`
WindowRef
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`typeGrafPtr`
CGrafPtr
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`typeGWorldPtr`
GWorldPtr
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`typeDragRef`
DragRef
Available in Mac OS X v10.0 and later.
Declared in `CarbonEventsCore.h`.

`typeMenuRef`
MenuRef
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`typeControlRef`
ControlRef
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`typeCollection`
Collection
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`typeQDRgnHandle`
RgnHandle
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`typeOSStatus`
OSStatus
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`typeCFIndex`
CFIndex
Available in Mac OS X v10.2 and later.
Declared in `CarbonEvents.h`.

`typeCGContextRef`
CGContextRef
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`typeHIPoint`
HIPoint
Available in Mac OS X v10.1 and later.
Declared in `CarbonEvents.h`.

`typeHISize`
HISize
Available in Mac OS X v10.2 and later.
Declared in `CarbonEvents.h`.

`typeHIRect`
HIRect
Available in Mac OS X v10.2 and later.
Declared in `CarbonEvents.h`.

`typeHIShapeRef`
HIShapeRef
Available in Mac OS X v10.4 and later.
Declared in `CarbonEvents.h`.

`typeVoidPtr`
Void

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

`typeGDHandle`
GDHandle

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

Discussion
You specify these parameters to obtain references to various objects such as windows, controls, graphics ports, and so on. See the various event kinds to determine the parameters available for each event. For example, Table 3-8 (page 392) in "Mouse Events" (page 390) lists the various parameters used in mouse events.

Core Foundation Object Types

Define type constants for Carbon event parameters that refer to Core Foundation objects.

```
enum {
    typeCFAttributedStringRef = 'cfas',
    typeCFMutableAttributedStringRef = 'cfaa',
    typeCFStringRef = 'cfst',
    typeCFMutableStringRef = 'cfms',
    typeCFArrayRef = 'cfar',
    typeCFMutableArrayRef = 'cfma',
    typeCFDictionaryRef = 'cfdc',
    typeCFMutableDictionaryRef = 'cfmd',
    typeCFNumberRef = 'cfnb',
    typeCFBooleanRef = 'cftf',
    typeCFTypeRef = 'cfty'
};
```

Constants
`typeCFAttributedStringRef`
A Core Foundation attributed string.

Available in Mac OS X v10.5 and later.

Declared in `AEDataModel.h`.

`typeCFMutableAttributedStringRef`
A Core Foundation mutable attributed string.

Available in Mac OS X v10.5 and later.

Declared in `AEDataModel.h`.

`typeCFStringRef`
A Core Foundation string.

Available in Mac OS X v10.1 and later.

Declared in `AEDataModel.h`.

`typeCFMutableStringRef`
A Core Foundation mutable string.

Available in Mac OS X v10.2 and later.

Declared in `AEDataModel.h`.

`typeCFArrayRef`
A Core Foundation array.
Available in Mac OS X v10.3 and later.
Declared in `AEDataModel.h`.

`typeCFMutableArrayRef`
A Core Foundation mutable array.
Available in Mac OS X v10.1 and later.
Declared in `AEDataModel.h`.

`typeCFDictionaryRef`
A Core Foundation dictionary.
Available in Mac OS X v10.3 and later.
Declared in `AEDataModel.h`.

`typeCFMutableDictionaryRef`
A Core Foundation mutable dictionary.
Available in Mac OS X v10.3 and later.
Declared in `AEDataModel.h`.

`typeCFNumberRef`
A Core Foundation number.
Available in Mac OS X v10.5 and later.
Declared in `AEDataModel.h`.

`typeCFBooleanRef`
A Core Foundation Boolean value.
Available in Mac OS X v10.5 and later.
Declared in `AEDataModel.h`.

`typeCFTypeRef`
A Core Foundation type.
Available in Mac OS X v10.2 and later.
Declared in `AEDataModel.h`.

Declared In
`AEDataModel.h`

Apple Event Constants

AppleEvent Constant

Define a constant related to events from `kEventClassAppleEvent`.

```
enum {
    kEventAppleEvent = 1
};
```

Constants

`kEventAppleEvent`

An AppleEvent event was received.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

Discussion

The standard application handler automatically calls the Apple Event Manager function `AEProcessAppleEvent` to handle Apple events.

Table 3-1 shows the parameter associated with AppleEvent events.

Table 3-1 Parameter names and types for AppleEvent kinds

Event kind	Parameter name	Parameter type
`kEventAppleEvent`	`kEventParamAEEventID`	`typeType`

Deprecated AppleEvent Event Constants

Define constants for older names for AppleEvent event constants.

```
enum {
    kEventClassEPPC = kEventClassAppleEvent,
    kEventHighLevelEvent = kEventAppleEvent
};
```

Constants

`kEventClassEPPC`

Equivalent to `kEventClassAppleEvent`.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventHighLevelEvent`

Equivalent to `kEventAppleEvent`.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

Appearance Manager Event Constants

Appearance Manager Events

Define a constant related to events from `kEventClassAppearance`.

```
enum {
    kEventAppearanceScrollBarVariantChanged = 1
};
```

Constants

`kEventAppearanceScrollBarVariantChanged`

The scroll bar variant has changed.

Available in Mac OS X v10.1 and later.

Declared in `CarbonEvents.h`.

Appearance Manager Event Parameter

Define a constant for the parameter to Appearance Manager events.

```
enum {
    kEventParamNewScrollBarVariant = 'nsbv'
};
```

Constants

`kEventParamNewScrollBarVariant`

typeShortInteger

Available in Mac OS X v10.1 and later.

Declared in `CarbonEvents.h`.

Application Event Constants

Application Event Constants

Define constants related to events from `kEventClassApplication`.

```
enum {
    kEventAppActivated = 1,
    kEventAppDeactivated = 2,
    kEventAppQuit = 3,
    kEventAppLaunchNotification = 4,
    kEventAppLaunched = 5,
    kEventAppTerminated = 6,
    kEventAppFrontSwitched = 7,
    kEventAppFocusMenuBar = 8,
    kEventAppFocusNextDocumentWindow = 9,
    kEventAppFocusNextFloatingWindow = 10,
    kEventAppFocusToolbar = 11,
    kEventAppFocusDrawer = 12,
    kEventAppGetDockTileMenu = 20,
    kEventAppIsEventInInstantMouser = 104,
    kEventAppHidden = 107,
    kEventAppShown = 108,
    kEventAppSystemUIModeChanged = 109,
    kEventAppAvailableWindowBoundsChanged = 110,
    kEventAppActiveWindowChanged = 111
};
```

Constants

`kEventAppActivated`
The application was activated (resumed, in old parlance).
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventAppDeactivated`
The application was deactivated (suspended, in old parlance).
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventAppQuit`
The application is quitting.
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventAppLaunchNotification`
Response to asynchronous application launch.
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventAppLaunched`
Some other application was launched. (CarbonLib 1.3 or later)
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventAppTerminated`
Some other application was terminated. (CarbonLib 1.3 or later)
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventAppFrontSwitched`
The frontmost application has changed. (CarbonLib 1.3 or later)

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventAppFocusMenuBar`
Request to switch the keyboard focus to the menu bar. The Carbon Event Manager handles this event by default.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

`kEventAppFocusNextDocumentWindow`
Request to shift keyboard focus to the next or previous document window (depending on the state of the Shift key). If there are no more document windows in the current process, focus should shift to the document window in the next (or previous) process.

If something other than a document window currently has keyboard focus, you should shift focus to the frontmost document window without changing the ordering of the windows.

If the document window does not have a focused area, you should set the focus to the main control within the window.

The Carbon Event Manager handles this event by default; if you handle this event, you should only check if the user focus is somewhere other than a document window, and if so, set the focus on the active document window. If the focus is already on a document window, your handler should always return `eventNotHandledErr` so that the default handler can rotate to the next window across all processes.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

`kEventAppFocusNextFloatingWindow`
Request to shift keyboard focus to the next or previous floating window (depending on the state of the Shift key).

If something other than a floating window currently has keyboard focus, you should shift focus to the frontmost floating window without changing the ordering of the windows.

If the floating window does not have a focused area, you should set the focus to the main control within the window.

The default behavior for this event is to send a `kEventCommandProcess` event containing `kHICommandRotateFloatingWindowsForward` or `kHICommandRotateFloatingWindowsBackward`.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

`kEventAppFocusToolbar`
Request to shift keyboard focus to the toolbar.

The default behavior for this event is to move the keyboard focus to the first item in the toolbar (assuming you are using the standard toolbar).

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

`kEventAppFocusDrawer`

Request to shift keyboard focus to the drawer in the focused window.

The default behavior for this event is to move the focus to the first control in the drawer in the focused window if a drawer is present. If multiple drawers are present, focus is moved in clockwise order from one drawer to the next, starting with the top drawer, if any. If the modifiers parameter contains the shift key, focus is moved in reverse (counterclockwise) order.

Available in Mac OS X v10.4 and later.

Declared in `CarbonEvents.h`.

`kEventAppGetDockTileMenu`

Request to display a pop-up menu by the application's dock tile. You should return the menu reference of the menu to display in the `kEventParamMenuRef` parameter. The sender of this event releases this menu after the Dock displays it, so if you supply a permanently allocated menu reference, you should call the Menu Manager function `RetainMenu` on it before returning from your handler.

The default behavior for this event is to return the menu (if any) supplied by the `SetApplicationDockTileMenu` function (described in the Dock Manager Reference). Note that for most functions, it's easier to set a menu using `SetApplicationDockTileMenu` rather than installing a handler for this event.

Available in Mac OS X v10.1 and later.

Declared in `CarbonEvents.h`.

`kEventAppIsEventInInstantMouser`

The given event's global mouse location is over an "instant mousing" area. An instant mousing area is an area where a mouse down should not generate ink but should be interpreted as a click.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`kEventAppHidden`

The application was hidden.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

`kEventAppShown`

The application was shown.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

`kEventAppSystemUIModeChanged`

The system user interface mode of the frontmost application has changed.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

`kEventAppAvailableWindowBoundsChanged`

The available window positioning bounds have changed. This event is currently sent when the Dock has changed position or size and when the display configuration has changed. A separate copy of this event is sent to each affected GDevice.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`kEventAppActiveWindowChanged`

The active window in the current process has changed. The Window Manager uses `ActiveNonFloatingWindow` to track the active window. When `SelectWindow` is called on a window, that window is made the new active window. At that time, the Window Manager also posts a `kEventAppActiveWindowChanged` event to the main event queue.

If more than one window is activated sequentially before the event loop is run, a single `kEventAppActiveWindowChanged` event is left in the event queue. Its `PreviousActiveWindow` parameter will be the window that was originally active, and its `CurrentActiveWindow` parameter will be the window that was finally active.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

Discussion
You can pass any of these constants when registering an event handler. You can also pass these constants to the `CreateEvent` (page 252) function to specify the type of application event you want to create.

Table 3-2 shows the event parameters associated with application events.

Table 3-2 Parameter names and types for application event kinds

Event kind	Parameter name	Parameter type
`kEventAppActivated`	`kEventParamWindowRef`	`typeWindowRef`
`kEventAppDeactivated`	*None*	
`kEventAppQuit`	*None*	
`kEventAppLaunchNotification`	`kEventParamProcessID`	`typeProcessSerialNumber`
	`kEventParamLaunchRefCon`	`typeUInt32`
	`kEventParamLaunchErr`	`typeOSStatus`
`kEventAppLaunched`	`kEventParamProcessID`	`typeProcessSerialNumber`
`kEventAppTerminated`	`kEventParamProcessID`	`typeProcessSerialNumber`
`kEventAppFrontSwitched`	`kEventParamProcessID`	`typeProcessSerialNumber`
`kEventAppFocusMenuBar`	`kEventParamKeyModifiers`	`typeUInt32`
`kEventAppFocusNextDocumentWindow`	`kEventParamKeyModifiers`	`typeUInt32`
`kEventAppFocusNextFloatingWindow`	`kEventParamKeyModifiers`	`typeUInt32`
`kEventAppFocusToolbar`	`kEventParamKeyModifiers`	`typeUInt32`
`kEventAppGetDockTileMenu`	`kEventParamMenuRef`	`typeMenuRef`
`kEventAppHidden`	*None*	
`kEventAppShown`	*None*	
`kEventAppSystemUIModeChanged`	`kEventParamSystemUIMode`	`typeUInt32`

Application Event Parameters

Define constants for parameters to application events.

```
enum {
    kEventParamProcessID = 'psn ',
    kEventParamLaunchRefCon = 'lref',
    kEventParamLaunchErr = 'err ',
    kEventParamSystemUIMode = 'uimd'
};
```

Constants

`kEventParamProcessID`
typeProcessSerialNumber
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamLaunchRefCon`
typeWildcard
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamLaunchErr`
typeOSStatus
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamSystemUIMode`
typeUInt32
Available in Mac OS X v10.2 and later.
Declared in `CarbonEvents.h`.

Command Events

Command Event Constants

Define constants related to events from `kEventClassCommand`.

```
enum {
    kEventProcessCommand = 1,
    kEventCommandProcess = 1,
    kEventCommandUpdateStatus = 2
};
```

Constants

`kEventProcessCommand`

A command has been invoked and the application should handle it. This event is sent when the user chooses a menu item or when a control with a command is pressed. Some senders of this event will also include the modifier keys that were pressed by the user when the command was invoked, but this parameter is optional.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventCommandProcess`

Same as `kEventProcessCommand`.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventCommandUpdateStatus`

Sent when updates related to the command event may be required. When you receive this event, you should update the necessary user interface elements in your application to reflect the current status of the command. For example, if the command has the `kHICommandFromMenu` bit set, you should update the menu item state, text, and so on, to reflect the current state of your application.

Note that the standard handler for `kEventMenuEnableItems` automatically sends this event to your menu commands. As this can cause a performance hit if you have many menu items, you can choose to bypass these updates by installing a no-op handler for `kEventMenuEnableItems` that simply returns `noErr`.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

Discussion

You pass this constant to the `CreateEvent` (page 252) function to indicate the type of command event you want to create. Future releases of the Carbon Event Manager will provide additional command event types.

Table 3-3 shows the parameters associated with command events.

Table 3-3 Parameter names and types for command event kinds

Event kind	Parameter name	Parameter type
`kEventCommandProcess`	`kEventParamDirectObject`	`typeHICommand`
	`kEventParamKeyModifiers` (Optional)	`typeUInt32`
	`kEventParamMenuContext` (Optional)	`typeUInt32`
`kEventCommand-UpdateStatus`	`kEventParamDirectObject`	`typeHICommand`
	`kEventParamMenuContext` (Optional)	`typeUInt32`

Standard Command ID Constants

Define command IDs for common menu commands and controls.

```
enum {
    kHICommandOK = 'ok  ',
    kHICommandCancel = 'not!',
    kHICommandQuit = 'quit',
    kHICommandUndo = 'undo',
    kHICommandRedo = 'redo',
    kHICommandCut = 'cut ',
    kHICommandCopy = 'copy',
    kHICommandPaste = 'past',
    kHICommandClear = 'clea',
    kHICommandSelectAll = 'sall',
    kHICommandHide = 'hide',
    kHICommandHideOthers = 'hido',
    kHICommandShowAll = 'shal',
    kHICommandPreferences = 'pref',
    kHICommandZoomWindow = 'zoom',
    kHICommandMinimizeWindow = 'mini',
    kHICommandMinimizeAll = 'mina',
    kHICommandMaximizeWindow = 'maxi',
    kHICommandMaximizeAll = 'maxa',
    kHICommandArrangeInFront = 'frnt',
    kHICommandBringAllToFront = 'bfrt',
    kHICommandWindowListSeparator = 'wldv',
    kHICommandWindowListTerminator = 'wlst',
    kHICommandSelectWindow = 'swin',
    kHICommandRotateWindowsForward = 'rotw',
    kHICommandRotateWindowsBackward = 'rotb',
    kHICommandRotateFloatingWindowsForward = 'rtfw',
    kHICommandRotateFloatingWindowsBackward = 'rtfb',
    kHICommandAbout = 'abou',
    kHICommandNew = 'new ',
    kHICommandOpen = 'open',
    kHICommandClose = 'clos',
    kHICommandSave = 'save',
    kHICommandSaveAs = 'svas',
    kHICommandRevert = 'rvrt',
    kHICommandPrint = 'prnt',
    kHICommandPageSetup = 'page',
    kHICommandAppHelp = 'ahlp',
    kHICommandShowCharacterPalette = 'chrp',
    kHICommandShowSpellingPanel = 'shsp',
    kHICommandCheckSpelling = 'cksp',
    kHICommandChangeSpelling = 'chsp',
    kHICommandCheckSpellingAsYouType = 'chsp',
    kHICommandIgnoreSpelling = 'igsp',
    kHICommandLearnWord = 'lrwd'
};
```

Constants

`kHICommandOK`

The OK button in a dialog or alert.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kHICommandCancel`
The Cancel button in a dialog or alert.
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kHICommandQuit`
The application should quit.
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kHICommandUndo`
The last editing operation should be undone.
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kHICommandRedo`
The last editing operation should be redone.
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kHICommandCut`
The selected items should be cut.
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kHICommandCopy`
The selected items should be copied.
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kHICommandPaste`
The contents of the clipboard should be pasted.
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kHICommandClear`
The selected items should be deleted.
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kHICommandSelectAll`
All items in the active window should be selected.
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kHICommandHide`
The application should be hidden. The Menu Manager responds to this command automatically; your application does not need to handle it.
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kHICommandHideOthers`
Other applications should be hidden. The Menu Manager responds to this command automatically; your application does not need to handle it.

Available in Mac OS X v10.1 and later.

Declared in `CarbonEvents.h`.

`kHICommandShowAll`
All applications should become visible. The Menu Manager responds to this command automatically; your application does not need to handle it.

Available in Mac OS X v10.1 and later.

Declared in `CarbonEvents.h`.

`kHICommandPreferences`
The Preferences menu item has been selected.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kHICommandZoom`
The active window should be zoomed in or out. The default application handler responds to this event automatically. Your application does not need to handle this event, but you may want to install a Carbon event handler for `kEventWindowGetIdealSize` to return the ideal size for your document windows.

`kHICommandMinimizeWindow`
The active window should be minimized. The default application handler will respond to this event automatically; your application does not need to handle it.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kHICommandMinimizeAll`
All collapsable windows should be minimized. The default application handler responds to this event automatically; your application does not need to handle it.

Available in Mac OS X v10.1 and later.

Declared in `CarbonEvents.h`.

`kHICommandMaximizeWindow`
The active window should be maximized. Sent only on Mac OS 9. The default application handler will respond to this event automatically; your application does not need to handle it.

Available in Mac OS X v10.1 and later.

Declared in `CarbonEvents.h`.

`kHICommandMaximizeAll`
All collapsible windows should be maximized. This event is not sent or handled on Mac OS X.

Available in Mac OS X v10.1 and later.

Declared in `CarbonEvents.h`.

`kHICommandArrangeInFront`
All document-class windows should be arranged in a stack. The default application handler responds to this event automatically; your application does not need to handle it.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kHICommandBringAllToFront`

All windows of this application should be brought in front of windows from other applications. Sent only on Mac OS X. The default application handler responds to this event automatically; your application does not need to handle it.

Available in Mac OS X v10.1 and later.

Declared in `CarbonEvents.h`.

`kHICommandWindowListSeparator`

A placeholder to mark the separator item dividing the Zoom/Minimize/Maximize/Arrange menu items in the standard Window menu from the menu items listing the visible windows. If you need to add your own menu items to the standard Window menu before the window list section, you can use `GetIndMenuItemWithCommandID` (described in the Menu Manager Reference in the User Experience section of the Carbon documentation) to look for the menu item with this command ID and insert your menu items before the item with this ID.

Available in Mac OS X v10.1 and later.

Declared in `CarbonEvents.h`.

`kHICommandWindowListTerminator`

Used as a placeholder to mark the end of the window list section of the standard Window menu. If you need to add your own menu items to the standard Window menu after the window list section, you can use `GetIndMenuItemWithCommandID` (described in the Menu Manager Reference in the User Experience section of the Carbon documentation) to look for the menu item with this command ID and insert your items after the item with this ID.

Available in Mac OS X v10.1 and later.

Declared in `CarbonEvents.h`.

`kHICommandSelectWindow`

A window in the standard Window menu has been selected and should be activated. In Mac OS X v10.3, this command is also sent by the toolbox whenever it needs to activate a window in your application. For example, it is used when a window is selected from the application's Dock menu, and when a window that uses the standard window event handler is clicked. The default application handler responds to this event automatically; your application does not need to handle it.

Available in Mac OS X v10.1 and later.

Declared in `CarbonEvents.h`.

`kHICommandRotateWindowsForward`

The Rotate Windows hot key (cmd-~ by default) has been pressed. Non-floating windows should be rotated so that the window after the active window is activated. The default application handler responds to this event automatically; your application does not need to handle it.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

`kHICommandRotateWindowsBackward`

The Rotate Windows hot key (cmd-~ by default) has been pressed. Non-floating windows should be rotated so that the window before the active window is activated. The default application handler responds to this event automatically; your application does not need to handle it.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

`kHICommandRotateFloatingWindowsForward`
: The floating window focus hot key (ctl-F6 by default) has been pressed, and floating windows should be rotated so that the window after the focused window is activated. The default application handler responds to this event automatically; your application does not need to handle it.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

`kHICommandRotateFloatingWindowsBackward`
: The floating window focus hot key (ctl-F6 by default) has been pressed, and floating windows should be rotated so that the window before the focused window is activated. The default application handler responds to this event automatically; your application does not need to handle it.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

`kHICommandAbout`
: The About menu item has been selected. In Mac OS X v10.3 and later, `RunApplicationEventLoop` installs a handler for this command ID on the application target that handles this event automatically by calling `HIAboutBox`. Your application can install its own handler if you want to display a customized about box.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kHICommandNew`
: A new document or item should be created.

Available in Mac OS X v10.1 and later.

Declared in `CarbonEvents.h`.

`kHICommandOpen`
: The user wants to open an existing document.

Available in Mac OS X v10.1 and later.

Declared in `CarbonEvents.h`.

`kHICommandClose`
: The active window should be closed. This command is typically be generated by a Close menu item. In Mac OS X v10.3 and later, the default application handler responds to this command by sending a `kEventWindowClose` event; on earlier systems, only the standard window event handler responded to this event.

Available in Mac OS X v10.1 and later.

Declared in `CarbonEvents.h`.

`kHICommandSave`
: The active document should be saved.

Available in Mac OS X v10.1 and later.

Declared in `CarbonEvents.h`.

`kHICommandSaveAs`
: The user wants to save the active document under a new name.

Available in Mac OS X v10.1 and later.

Declared in `CarbonEvents.h`.

`kHICommandRevert`
The contents of the active document should be reverted to the last saved version.

Available in Mac OS X v10.1 and later.

Declared in `CarbonEvents.h`.

`kHICommandPrint`
The active window should be printed.

Available in Mac OS X v10.1 and later.

Declared in `CarbonEvents.h`.

`kHICommandPageSetup`
The user wants to configure the current page margins, formatting, and print options.

Available in Mac OS X v10.1 and later.

Declared in `CarbonEvents.h`.

`kHICommandAppHelp`
The application's help book should be displayed. The Help Manager installs a handler for this command ID on the Help menu returned by `HMGetHelpMenu` and responds to this event automatically. Your application does not need to handle it. In Mac OS X v10.4, the Help Manager installs a handler for this event on the application event target rather than on the Help menu.

Available in Mac OS X v10.1 and later.

Declared in `CarbonEvents.h`.

`kHICommandShowCharacterPalette`
The character palette needs to be shown. Events with this command ID are only generated in Mac OS X v10.3 and later. The toolbox will respond to this event automatically; your application does not need to handle it.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`kHICommandShowSpellingPanel`
Display the spelling panel if it is not already visible. Events with this command ID are only generated in Mac OS X v10.4 and later. If spell checking has been enabled in the Multilingual Text Engine (MLTE) or an HITextView, this command is handled automatically.

Available in Mac OS X v10.4 and later.

Declared in `CarbonEvents.h`.

`kHICommandCheckSpelling`
Spell check the document now. Events with this command ID are only generated in Mac OS X v10.4 and later. If spell checking has been enabled in MLTE or an HITextView, this command is handled automatically.

Available in Mac OS X v10.4 and later.

Declared in `CarbonEvents.h`.

`kHICommandChangeSpelling`
Change the spelling. Events with this command ID are only generated in Mac OS X v10.4 and later. If spell checking has been enabled in MLTE or an HITextView, this command is handled automatically.

Available in Mac OS X v10.4 and later.

Declared in `CarbonEvents.h`.

`kHICommandCheckSpellingAsYouType`
Begin interactive spell checking. Events with this command ID are only generated in Mac OS X v10.4 and later. If spell checking has been enabled in MLTE or an HITextView, this command is handled automatically.

Available in Mac OS X v10.4 and later.

Declared in `CarbonEvents.h`.

`kHICommandIgnoreSpelling`
Ignore this word while spell checking this text view. Events with this command ID are only generated in Mac OS X v10.4 and later. If spell checking has been enabled in MLTE or an HITextView, this command is handled automatically.

Available in Mac OS X v10.4 and later.

Declared in `CarbonEvents.h`.

`kHICommandLearnWord`
Learn this spelling for all documents. Events with this command ID are generated only in Mac OS X v10.4 and later. If spell checking has been enabled in MLTE or an HITextView, this command is handled automatically.

Available in Mac OS X v10.4 and later.

Declared in `CarbonEvents.h`.

Discussion
You should use these values for standard menu and control commands rather than defining your own.

Command Event Source Constants

Define constants for the user interface element that produced an HICommand event.

```
enum {
    kHICommandFromMenu = (1L << 0),
    kHICommandFromControl = (1L << 1),
    kHICommandFromWindow = (1L << 2)
};
```

Constants
`kHICommandFromMenu`
This bit is set for commands generated from menu items in all versions of CarbonLib and Mac OS X.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kHICommandFromControl`
The command event originated from a control. This bit was introduced in Mac OS X v10.2 and CarbonLib 1.6; it is never set in earlier versions of Mac OS X or CarbonLib.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

`kHICommandFromWindow`
The command event originated from a window. This bit was introduced in Mac OS X v10.2 and CarbonLib 1.6; it is never set in earlier versions of Mac OS X or CarbonLib.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

Control Events

Control Event Constants

Define constants related to events from `kEventClassControl`.

```
enum {
    kEventControlInitialize = 1000,
    kEventControlDispose = 1001,
    kEventControlGetOptimalBounds = 1003,
    kEventControlDefInitialize = kEventControlInitialize,
    kEventControlDefDispose = kEventControlDispose,
    kEventControlHit = 1,
    kEventControlSimulateHit = 2,
    kEventControlHitTest = 3,
    kEventControlDraw = 4,
    kEventControlApplyBackground = 5,
    kEventControlApplyTextColor = 6,
    kEventControlSetFocusPart = 7,
    kEventControlGetFocusPart = 8,
    kEventControlActivate = 9,
    kEventControlDeactivate = 10,
    kEventControlSetCursor = 11,
    kEventControlContextualMenuClick = 12,
    kEventControlClick = 13,
    kEventControlGetNextFocusCandidate = 14,
    kEventControlGetAutoToggleValue = 15,
    kEventControlInterceptSubviewClick = 16,
    kEventControlGetClickActivation = 17,
    kEventControlDragEnter = 18,
    kEventControlDragWithin = 19,
    kEventControlDragLeave = 20,
    kEventControlDragReceive = 21,
    kEventControlTrack = 51,
    kEventControlGetScrollToHereStartPoint = 52,
    kEventControlGetIndicatorDragConstraint = 53,
    kEventControlIndicatorMoved = 54,
    kEventControlGhostingFinished = 55,
    kEventControlGetActionProcPart = 56,
    kEventControlGetPartRegion = 101,
    kEventControlGetPartBounds = 102,
    kEventControlSetData = 103,
    kEventControlGetData = 104,
    kEventControlGetSizeConstraints = 105,
    kEventControlValueFieldChanged = 151,
    kEventControlAddedSubControl = 152,
    kEventControlRemovingSubControl = 153,
    kEventControlBoundsChanged = 154,
    kEventControlTitleChanged = 158,
    kEventControlOwningWindowChanged = 159,
    kEventControlHiliteChanged = 160,
    kEventControlEnabledStateChanged = 161,
    kEventControlArbitraryMessage = 201
};
```

Constants

`kEventControlInitialize`

Sent when a control is created. Allows the control to initialize private data.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventControlDispose`
Sent when a control is disposed. Allows the control to dispose of private data.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventControlGetOptimalBounds`
Allows the control to report its best size and its text baseline based on its current settings. You should set the `kEventParamControlOptimalBounds` parameter to an appropriate rectangle. You should also set the `kEventParamControlOptimalBaselineOffset` parameter to be the offset from the top of your optimal bounds of a text baseline, if any. (Mac OS X only)

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventControlDefInitialize`
Same as `kEventControlInitialize`. You can use this event when creating custom control definitions.

Available in Mac OS X v10.0 through Mac OS X v10.2.

Declared in `CarbonEvents.h`.

`kEventControlDefDispose`
Same as `kEventControlDispose`. You can use this event when creating custom control definitions.

Available in Mac OS X v10.0 through Mac OS X v10.2.

Declared in `CarbonEvents.h`.

`kEventControlHit`
Sent by the Control Manager functions `TrackControl` and `HandleControlClick` after handling a click in a control. If you do not handle this event, and the control has a command ID associated with it, then the Control Manager sends a `kEventCommandProcess` event to the control.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventControlSimulateHit`
Sent when your control should simulate a click in response to some other action, such as a return key for a default button. The default behavior is to use the Control Manager function `HiliteControl` to highlight and unhighlight the part specified in the `kEventParamControlPart` parameter (simulating the hit) and then call the control's action callback function. (Mac OS X only)

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventControlHitTest`
Sent when someone wants to find out what part of your control is at a given point in local coordinates. You should set the `kEventParamControlPart` parameter to the appropriate part. (Mac OS X only)

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventControlDraw`
Sent when your control should draw itself. The event can optionally contain parameters indicating which port to draw into and which part to constrain drawing to. (Mac OS X only)

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventControlApplyBackground`
Sent when your control should apply its background color/pattern to the port specified so the subcontrol can properly erase. The port is optional; if it does not exist you should apply the background to the current port. Note that if you don't handle this event, the event is propagated to the control's parent. (Mac OS X only)

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CarbonEvents.h`.

`kEventControlApplyTextColor`
Sent when your control should apply a color/pattern to the specified port and context so a subcontrol can draw text which looks appropriate for your control's background. The port is optional; if it does not exist, you should apply the text color to the current port. The context is also optional. (Mac OS X only)

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventControlSetFocusPart`
Sent when your control is gaining, losing, or changing the focus. Set the focus to the part indicated by the `kEventParamControlPart` parameter. (Mac OS X only)

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventControlGetFocusPart`
Sent when your the Control Manager wants to know what part of your control is currently focused. Set the `kEventParamControlPart` parameter to your currently focused part. If you don't handle this event, the Control Manager sets the part parameter to the last part that was focused (or no part if the control lost focus). (Mac OS X only)

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventControlActivate`
Sent when your control becomes active as a result of a call to `ActivateControl`. (Mac OS X only)

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventControlDeactivate`
Sent when your control becomes inactive as a result of a call to `DeactivateControl`. (Mac OS X only)

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventControlSetCursor`
Sent when your control is asked to change the cursor as a result of a call to the Control Manager function `HandleControlSetCursor`. (Mac OS X only)

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventControlContextualMenuClick`

Sent when your control is asked to display a contextual menu as a result of a call to the Control Manager function `HandleControlContextualMenuClick`. (Mac OS X only)

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventControlClick`

A mouse down occurred in a control. The standard window handler sets the keyboard focus to the control if it takes focus on clicks, and calls the Control Manager function `HandleControlClick`.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventControlGetNextFocusCandidate`

Sent to allow a control to customize the focus order of its subcontrols. The current subcontrol with focus is stored in the `kEventParamStartControl` parameter. The desired focus direction is indicated by the `kControlFocusNextPart` or `kControlFocusPrevPart` constants, passed to you in the `kEventParamControlPart` parameter. The handler should return the next subcontrol in the `kEventParamNextControl` parameter. If the `kEventParamStartControl` parameter is `NULL`, return the first subcontrol in the specified focus direction. If no next subcontrol exists in the desired focus direction, return `NULL` or omit the `kEventParamNextControl` parameter.

The default behavior is to return the "most appropriate" peer control, which currently means the previous control in the ordering scheme.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

`kEventControlGetAutoToggleValue`

Sent when the system wants to auto-toggle a control. You can specify the value to use based on the current value of your control.

If the control has the `kControlAutoToggles` feature bit set, then the default behavior is as follows:

- If the control does not behave like a radio button (the `kControlHasRadioBehavior` feature bit is not set), and its value is 1, then the `kEventParamControlValue` parameter is set to 0.
- If the control's value is anything other than 1, the `kEventParamControlValue` parameter is set to 0.

Otherwise, there is no default behavior.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

`kEventControlInterceptSubviewClick`

Sent when the `HIViewGetViewForMouseClick` function is called (typically by the Control Manager before it descends into subviews). A view can use this event to intercept mouse clicks that would normally be destined for one of its subviews. For example, the Toolbar control uses this event to intercept command-clicks so that it can handle dragging of its children. If the command key is down, the user wants to drag, so the handler returns `noErr` to indicate that this view (the Toolbar) should receive the click, not the child that was actually under the mouse.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

`kEventControlGetClickActivation`

Sent when a mouse click occurs in a background (inactive) control. This event is essentially the control version of `kEventWindowGetClickActivation`. The only differences are that the mouse location is view-relative and no window part parameter is passed to you.

This event is sent only when the standard window handler is installed. The default behavior is to activate the view and absorb the mouse click (that is, the click is not passed on to the view).

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

`kEventControlDragEnter`

Sent when a drag item enters a view's bounds. If you want to respond to the drag, your drag entered handler must return `noErr` If you return `eventNotHandledErr` then you will not receive further drag events, nor will you be able to receive the drag item.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

`kEventControlDragWithin`

Sent when a drag item has moved while in the view's bounds (but not within any of its subviews). If the drag subsequently enters a subview, all additional drag events are directed to that subview.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

`kEventControlDragLeave`

Sent when a drag item leaves your view. You can use this event to unhighlight your view, and so on. (Available in Mac OS X v10.2 and later.)

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

`kEventControlDragReceive`

Sent when a drag item is dropped within your view.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

`kEventControlTrack`

Sent to allow your control to completely replace the normal tracking that is part of a call to the Control Manager functions `TrackControl` or `HandleControlClick`. Set the `kEventParamControlPart` to the part hit during tracking.

This event is sent only to controls that return a non-zero control part code from `kEventControlHitTest`. If you are implementing a custom HIView and you need to receive this event, you must also handle `kEventControlHitTest`. The hit-test handler must place a valid control part code into the `kEventParamControlPart` parameter and return `noErr`.

The default behavior is to implement indicator tracking (if the mouse is down in an indicator part, such as for a scroll bar) or one-part tracking (if the mouse is down in a button or similar part). If the tracking is successful, the Control Manager passes back the part that was hit.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventControlGetScrollToHereStartPoint`
Sent so your control can support "Scroll To Here" behavior during tracking. Set the `kEventParamMouseLocation` parameter to the mouse location in local coordinates which represents where a click would have needed to be to cause your indicator to be dragged to the incoming mouse location. (Mac OS X only)

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventControlGetIndicatorDragConstraint`
Sent so your control can constrain the movement of its indicator during tracking. Set the `kEventParamControlIndicatorDragConstraint` parameter to the appropriate constraint. (Mac OS X only)

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventControlIndicatorMoved`
Sent during live-tracking of the indicator so your control can update its value based on the new indicator position. During non-live tracking, this event lets you redraw the indicator ghost at the appropriate place. (Mac OS X only)

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventControlGhostingFinished`
Sent at the end of non-live indicator tracking so your control can update its value based on the final ghost location. (Mac OS X only)

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventControlGetActionProcPart`
Sent during tracking so your control can alter the part that is passed to its action callback based on modifier keys, etc. Set the `kEventParamControlPart` to the part you want to have sent. (Mac OS X only)

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventControlGetPartRegion`
Sent when a client wants to get a particular region of your control. See the `GetControlRegion` function in the Control Manager. The `kEventParamControlRegion` contains a region for you to modify.

If the requested part is `kControlStructureMetaPart`, the default behavior is to pass back a region equal to the control's bounds. Otherwise, there is no default behavior.

(Mac OS X only)

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventControlGetPartBounds`
Sent when a client wants to get a particular rectangle of your control when it may be more efficient than asking for a region. Set the `kEventParamControlPartBounds` parameter to the appropriate rectangle. (Mac OS X only)

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventControlSetData`
Sent when a client wants to change an arbitrary setting of your control. See the `SetControlData` function in the Control Manager. (Mac OS X only)

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventControlGetData`
Sent when a client wants to get an arbitrary setting of your control. See the Control Manager function `GetControlData`. (Mac OS X only)

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventControlGetSizeConstraints`
Sent when the `HIViewGetSizeContraints` function is called. You use this to let your custom view indicate its maximum and minimum size. A parent view can use this information to help it lay out subviews.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

`kEventControlValueFieldChanged`
Sent when your control's value, minimum, maximum, or view size has changed. Useful so other entities can watch for your control's value to change. If the window does not have compositing enabled, the default behavior is to redraw the control (but not its subcontrols). (Mac OS X only)

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventControlAddedSubControl`
Sent when a control is embedded within your control.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventControlRemovingSubControl`
Sent when one of your child controls will be removed from your control.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventControlBoundsChanged`
Sent when your control's bounding rectangle has changed. Note that the `kEventParamOriginalBounds` and `kEventParamPreviousBounds` parameters for this event contain the same value.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventControlTitleChanged`
Sent when your control's title changes.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

`kEventControlOwningWindowChanged`
Sent when one your control's owning window has changed. Useful to update any dependencies that your control has on its owning window. (Mac OS X only)

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventControlHiliteChanged`
Sent when a control's highlight state changes.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

`kEventControlEnabledStateChanged`
Sent when a control's enabled state changes (that is, when a control is enabled or disabled).

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

`kEventControlArbitraryMessage`
Sent when someone tries to send an old-style CDEF message to your control. In most cases, you should implement Carbon event replacements for CDEF messages instead. If you do handle this event, but do not explicitly handle a particular CDEF message, you should propagate this event up the handler chain (either explicitly by calling `CallNextEventHandler` (page 247) or implicitly by returning `eventNotHandledErr`), as some default behavior may be implemented for compatibility purposes. (Mac OS X only)

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CarbonEvents.h`.

Discussion
You can specify any of these events when installing an event handler. You can also pass these constants to the `CreateControlEvent` function to specify the type of control event you want to create.

Note that many control events are not sent as a request for you to take action; rather they provide a way for the application to override default behavior. Because this is the case, most control events do not have a standard handler associated with them. Their default behavior occurs whether or not you have the standard window handler installed.

Table 3-4 shows the parameters available for control events.

Table 3-4 Parameter names and types for common control event kinds

Event kind	Parameter name	Parameter type
`kEventControl-Initialize`	`kEventParamDirectObject`	`typeControlRef`
	`kEventParamInitCollection`	`typeCollection`
	`kEventParamControlFeatures`	`typeUInt32`
`kEventControlDispose`	`kEventParamDirectObject`	`typeControlRef`
`kEventControlGet-OptimalBounds`	`kEventParamDirectObject`	`typeControlRef`

	kEventParamControlOptimalBounds	typeQDRectangle
	kEventParamControlOptimalBaseline-Offset (Optional)	typeShortInteger
kEventControlHit	kEventParamDirectObject	typeControlRef
	kEventParamControlPart	typeControlPartCode
	kEventParamKeyModifiers	typeUInt32
kEventControl-SimulateHit	kEventParamDirectObject	typeControlRef
	kEventParamControlPart	typeControlPartCode
	kEventParamKeyModifiers	typeUInt32
kEventControlHitTest	kEventParamDirectObject	typeControlRef
	kEventParamMouseLocation	typeQDPoint
	kEventParamControlPart	typeControlPartCode
kEventControlDraw	kEventParamDirectObject	typeControlRef
	kEventParamControlPart (Optional)	typeControlPartCode
	kEventParamGrafPort (Optional)	typeGrafPtr
	kEventParamRgnHandle (Optional)	typeQDRgnHandle
	kEventParamCGContextRef (Optional)	typeCGContextRef
kEventControlApply-Background	kEventParamDirectObject	typeControlRef
	kEventParamControlSubControl	typeControlRef
	kEventParamControlDrawDepth	typeShortInteger
	kEventParamGrafPort (Optional)	typeGrafPtr
kEventControlApply-TextColor	kEventParamDirectObject	typeControlRef
	kEventParamControlSubControl	typeControlRef
	kEventParamControlDrawDepth	typeShortInteger
	kEventParamControlDrawInColor	typeBoolean
	kEventParamGrafPort (Optional)	typeGrafPtr
	kEventParamCGContextRef (Optional)	typeCGContextRef

kEventControlGetNext-FocusCandidate	kEventParamNextControl	typeControlRef
	kEventParamControlPart	typeControlPartCode
kEventControlGet-AutoToggleValue	kEventParamDirectObject	typeControlRef
	kEventParamControlPart	typeControlPartCode
	kEventParamControlValue	typeLongInteger
kEventControl-InterceptSubview-Click	kEventParamEventRef	typeEventRef
kEventControlGet-ClickActivation	kEventParamClickActivation	typeClickActivationResult
kEventControl-DragEnter	kEventParamDirectObject	typeControlRef
	kEventParamDragRef	typeDragRef
kEventControl-DragWithin	kEventParamDirectObject	typeControlRef
	kEventParamDragRef	typeDragRef
kEventControl-DragLeave	kEventParamDirectObject	typeControlRef
	kEventParamDragRef	typeDragRef
kEventControl-DragReceive	kEventParamDirectObject	typeControlRef
	kEventParamDragRef	typeDragRef
kEventControl-SetFocusPart	kEventParamDirectObject	typeControlRef
	kEventParamStartControl	typeControlRef
	kEventParamControlPart	typeControlPartCode
	kEventParamControlFocusEverything (Optional)	typeBoolean
kEventControl-GetFocusPart	kEventParamDirectObject	typeControlRef
	kEventParamControlPart	typeControlPartCode

kEventControl-Activate	kEventParamDirectObject	typeControlRef
kEventControl-Deactivate	kEventParamDirectObject	typeControlRef
kEventControl-SetCursor	kEventParamDirectObject	typeControlRef
	kEventParamMouseLocation	typeQDPoint
	kEventParamKeyModifiers	typeUInt32
kEventControl-ContextualMenuClick	kEventParamDirectObject	typeControlRef
	kEventParamMouseLocation	typeQDPoint
kEventControlTrack	kEventParamDirectObject	typeControlRef
	kEventParamMouseLocation	typeQDPoint
	kEventParamKeyModifiers	typeUInt32
	kEventParamControlAction	typeControlActionUPP
	kEventParamControlPart	typeControlPartCode
kEventControlGet-ScrollToHereStart-Point	kEventParamDirectObject	typeControlRef
	kEventParamMouseLocation	typeQDPoint
	kEventParamKeyModifiers	typeUInt32
kEventControlGet-IndicatorDrag-Constraint	kEventParamDirectObject	typeControlRef
	kEventParamMouseLocation	typeQDPoint
	kEventParamKeyModifiers	typeUInt32
	kEventParamControlIndicator Drag-Constraint	typeIndicatorDrag-Constraint
kEventControl-IndicatorMoved	kEventParamDirectObject	typeControlRef
	kEventParamControlIndicator Region	typeQDRgnHandle
	kEventParamControlIsGhosting	typeBoolean

kEventControl-GhostingFinished	kEventParamDirectObject	typeControlRef
	kEventParamControlIndicator Offset	typeQDPoint
kEventControlGet-ActionProcPart	kEventParamDirectObject	typeControlRef
	kEventParamKeyModifiers	typeUInt32
	kEventParamControlPart	typeControlPartCode
kEventControlGet-PartRegion	kEventParamDirectObject	typeControlRef
	kEventParamControlPart	typeControlPartCode
	kEventParamControlRegion	typeQDRgnHandle
kEventControlGet-PartBounds	kEventParamDirectObject	typeControlRef
	kEventParamControlPart	typeControlPartCode
	kEventParamControlBounds	typeQDRectangle
kEventControlSetData	kEventParamDirectObject	typeControlRef
	kEventParamControlPart	typeControlPartCode
	kEventParamControlDataTag	typeEnumeration
	kEventParamControlDataBuffer	typePtr
	kEventParamControlDataBufferSize	typeLongInteger
kEventControlGetData	kEventParamDirectObject	typeControlRef
	kEventParamControlPart	typeControlPartCode
	kEventParamControlDataTag	typeEnumeration
	kEventParamControlDataBuffer	typePtr
	kEventParamControlDataBuffer Size	typeLongInteger
kEventControlGet-SizeConstraints	kEventParamDirectObject	typeControlRef
	kEventParamMinimumSize	typeHISize
	kEventParamMaximumSize	typeHISize
kEventControlValue-FieldChanged	kEventParamDirectObject	typeControlRef

kEventControlAdded-SubControl	kEventParamDirectObject	typeControlRef
	kEventParamControlSubControl	typeControlRef
kEventControl-RemovingSubControl	kEventParamDirectObject	typeControlRef
	kEventParamControlSubControl	typeControlRef
kEventControlBounds-Changed	kEventParamDirectObject	typeControlRef
	kEventParamAttributes	typeUInt32
	kEventParamOriginalBounds	typeQDRectangle
	kEventParamPreviousBounds	typeQDRectangle
	kEventParamCurrentBounds	typeQDRectangle
kEventControlOwning-WindowChanged	kEventParamDirectObject	typeControlRef
	kEventParamAttributes	typeUInt32
	kEventParamControl-OriginalOwningWindow	typeWindowRef
	kEventParamControl-CurrentOwningWindow	typeWindowRef
kEventControlHilite-StateChanged	kEventParamDirectObject	typeControlRef
	kEventParamPreviousPart	typeControlPartCode
	kEventParamCurrentPart	typeControlPartCode
kEventControlEnabled-StateChanged	kEventParamDirectObject	typeControlRef
kEventControl-ArbitraryMessage	kEventParamDirectObject	typeControlRef
	kEventParamControlMessage	typeShortInteger
	kEventParamControlParam	typeLongInteger
	kEventParamControlResult	typeLongInteger

Control Bounds Constants

Define control bounds change-event attributes.

```
enum {
    kControlBoundsChangeSizeChanged = (1 << 2),
    kControlBoundsChangePositionChanged = (1 << 3)
};
```

Constants

`kControlBoundsChangeSizeChanged`

The dimensions of the control (width and height) changed.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kControlBoundsChangePositionChanged`

The position of the control changed (that is, the top-left corner moved).

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

Discussion

When the system sends out a `kEventControlBoundsChanged` event, it also sends along a parameter containing attributes of the event. These attributes can be used to determine what aspect of the control changed (position, size, or both).

Control Event Parameters

Define parameters related to control events.

```
enum {
    kEventParamControlPart = 'cprt',
    kEventParamInitCollection = 'icol',
    kEventParamControlMessage = 'cmsg',
    kEventParamControlParam = 'cprm',
    kEventParamControlResult = 'crsl',
    kEventParamControlRegion = 'crgn',
    kEventParamControlAction = 'caup',
    kEventParamControlIndicatorDragConstraint = 'cidc',
    kEventParamControlIndicatorRegion = 'cirn',
    kEventParamControlIsGhosting = 'cgst',
    kEventParamControlIndicatorOffset = 'ciof',
    kEventParamControlClickActivationResult = 'ccar',
    kEventParamControlSubControl = 'csub',
    kEventParamControlOptimalBounds = 'cobn',
    kEventParamControlOptimalBaselineOffset = 'cobo',
    kEventParamControlDataTag = 'cdtg',
    kEventParamControlDataBuffer = 'cdbf',
    kEventParamControlDataBufferSize = 'cdbs',
    kEventParamControlDrawDepth = 'cddp',
    kEventParamControlDrawInColor = 'cdic',
    kEventParamControlFeatures = 'cftr',
    kEventParamControlPartBounds = 'cpbd',
    kEventParamControlOriginalOwningWindow = 'coow',
    kEventParamControlCurrentOwningWindow = 'ccow',
    kEventParamControlFocusEverything = 'cfev',
    kEventParamNextControl = 'cnxc',
    kEventParamStartControl = 'cstc',
    kEventParamControlSubview = 'csvw',
    kEventParamControlPreviousPart = 'copc',
    kEventParamControlCurrentPart = 'cnpc',
    kEventParamControlInvalRgn = 'civr',
    kEventParamControlValue = 'cval',
    kEventParamControlHit = 'chit',
    kEventParamControlPartAutoRepeats = 'caur',
    kEventParamControlFrameMetrics = 'cfmt',
    kEventParamControlWouldAcceptDrop = 'cldg',
    kEventParamControlPrefersShape = 'cpsh',
    typeControlActionUPP = 'caup',
    typeIndicatorDragConstraint = 'cidc',
    typeControlPartCode = 'cprt',
    typeControlFrameMetrics = 'cins'
};
```

Constants

`kEventParamControlPart`

typeControlPartCode

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventParamInitCollection`

typeCollection

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventParamControlMessage`
typeShortInteger
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamControlParam`
typeLongInteger
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamControlResult`
typeLongInteger
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamControlRegion`
typeQDRgnHandle
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamControlAction`
typeControlActionUPP
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamControlIndicatorDragConstraint`
typeIndicatorDragConstraint
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamControlIndicatorRegion`
typeQDRgnHandle
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamControlIsGhosting`
typeBoolean
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamControlIndicatorOffset`
typeQDPoint
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamControlClickActivationResult`
typeClickActivationResult
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamControlSubControl`
typeControlRef
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamControlOptimalBounds`
typeQDRectangle
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamControlOptimalBaselineOffset`
typeShortInteger
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamControlDataTag`
typeEnumeration
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamControlDataBuffer`
typePtr
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamControlDataBufferSize`
typeLongInteger
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamControlDrawDepth`
typeShortInteger
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamControlDrawInColor`
typeBoolean
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamControlFeatures`
typeUInt32
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamControlPartBounds`
typeQDRectangle
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamControlOriginalOwningWindow`
typeWindowRef
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamControlCurrentOwningWindow`
typeWindowRef
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamControlFocusEverything`
typeBoolean
Available in Mac OS X v10.2 and later.
Declared in `CarbonEvents.h`.

`kEventParamNextControl`
typeControlRef
Available in Mac OS X v10.2 and later.
Declared in `CarbonEvents.h`.

`kEventParamStartControl`
typeControlRef
Available in Mac OS X v10.2 and later.
Declared in `CarbonEvents.h`.

`kEventParamControlSubview`
typeControlRef
Available in Mac OS X v10.2 and later.
Declared in `CarbonEvents.h`.

`kEventParamControlPreviousPart`
typeControlPartCode
Available in Mac OS X v10.2 and later.
Declared in `CarbonEvents.h`.

`kEventParamControlCurrentPart`
typeControlPartCode
Available in Mac OS X v10.2 and later.
Declared in `CarbonEvents.h`.

`kEventParamControlInvalRgn`
typeQDRgnHandle
Available in Mac OS X v10.2 and later.
Declared in `CarbonEvents.h`.

`kEventParamControlValue`
typeLongInteger
Available in Mac OS X v10.2 and later.
Declared in `CarbonEvents.h`.

`kEventParamControlHit`
typeBoolean
Available in Mac OS X v10.3 and later.
Declared in `CarbonEvents.h`.

`kEventParamControlPartAutoRepeats`
typeBoolean
Available in Mac OS X v10.3 and later.
Declared in `CarbonEvents.h`.

`kEventParamControlFrameMetrics`
typeControlFrameMetrics
Available in Mac OS X v10.3 and later.
Declared in `CarbonEvents.h`.

`kEventParamControlWouldAcceptDrop`
typeBoolean
Available in Mac OS X v10.3 and later.
Declared in `CarbonEvents.h`.

`kEventParamControlPrefersShape`
typeBoolean
Available in Mac OS X v10.4 and later.
Declared in `CarbonEvents.h`.

`typeControlActionUPP`
ControlActionUPP
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`typeIndicatorDragConstraint`
IndicatorDragConstraint
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`typeControlPartCode`
ControlPartCode
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`typeControlFrameMetrics`
HIViewFrameMetrics
Available in Mac OS X v10.3 and later.
Declared in `CarbonEvents.h`.

Ink Events

Ink Event Constants

Define constants related to events from `kEventClassInk`.

```
enum {
    kEventInkPoint = 10,
    kEventInkGesture = 11,
    kEventInkText = 12
};
```

Constants

`kEventInkPoint`

A mouse event will be handled as an ink point and used for recognition. The Ink Manager has determined that the mouse event in `kEventParamEventRef` should be used for recognition. If the application handles the event and returns `noErr`, the Ink Manager does nothing further with the mouse event. If the application returns any other value (including `eventNotHandledErr`), the Ink Manager processes the point normally.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`kEventInkGesture`

The Ink Manager recognizes the current ink phrase as one of the known system gestures. Applications can install a handler for these events to provide targeted gestures and support for context-dependent (tentative) gestures. Applications should return `noErr` if they handled the gesture. If the gesture was context dependent and does not apply to the current situation, applications should return `eventNotHandledErr`.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`kEventInkText`

The Ink Manager recognizes a word. The `kEventParamInkTextRef` parameter contains the ink text reference with all the information about the word. For more information, see `Ink.h`.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

Discussion

Table 3-6 shows the parameters associated with ink events

Table 3-5 Parameter names and types for ink event kinds

Event kind	Parameter name	Parameter type
`kEventInkPoint`	`kEventParamEventRef`	`typeEventRef`
`kEventInkGesture`	`kEventParamInkGesture`	`typeHIRect`
	`kEventParamInkGestureBounds`	`typeHIRect`
	`kEventParamInkGestureHotspot`	`typeHIPoint`
`kEventInkText`	`kEventParamInkTextRef`	`typePtr`
	`kEventParamInkKeyboardShortcut`	`typeBoolean`

Ink Event Parameters

Define constants for parameters to ink events.

```
enum {
    kEventParamInkTextRef = 'iwrd',
    kEventParamInkKeyboardShortcut = 'ikbd',
    kEventParamInkGestureKind = 'gknd',
    kEventParamInkGestureBounds = 'gbnd',
    kEventParamInkGestureHotspot = 'ghot'
};
```

Constants

`kEventParamInkTextRef`
The ink text reference containing the data for the word the Ink Manager recognized. (`typePtr`)

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`kEventParamInkKeyboardShortcut`
A Boolean whose value indicates whether the word the Ink Manager recognized is a keyboard shortcut. That is, the Command or Control key was pressed and the top-choice alternate text is a single character. (`typeBoolean`)

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`kEventParamInkGestureKind`
Kind of gesture. (`typeUInt32`)

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`kEventParamInkGestureBounds`
Bounds of the gesture in global coordinates. (`typeHIRect`)

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`kEventParamInkGestureHotspot`
Hotspot, in global coordinates, for the gesture. (`typeHIPoint`)

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

Keyboard Events

Keyboard Event Constants

Define constants related to events from `kEventClassKeyboard`.

```
enum {
    kEventRawKeyDown = 1,
    kEventRawKeyRepeat = 2,
    kEventRawKeyUp = 3,
    kEventRawKeyModifiersChanged = 4,
    kEventHotKeyPressed = 5,
    kEventHotKeyReleased = 6
};
```

Constants

`kEventRawKeyDown`
A key was pressed.
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventRawKeyRepeat`
Sent periodically as a key is held down by the user.
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventRawKeyUp`
A key was released.
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventRawKeyModifiersChanged`
The keyboard modifiers have changed.
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventHotKeyPressed`
A registered hot key was pressed.
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventHotKeyReleased`
A registered hot key was released.
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

Discussion
These events are the lowest-level keyboard events.

Table 3-6 shows the parameters associated with keyboard events.

Table 3-6 Parameter names and types for keyboard event kinds

Event kind	Parameter name	Parameter type
`kEventRawKeyDown`	`kEventParamKeyMacCharCodes`	`typeChar`
	`kEventParamKeyCode`	`typeUInt32`
	`kEventParamKeyModifiers`	`typeUInt32`

	kEventParamKeyboardType	typeUInt32
kEventRawKeyRepeat	kEventParamKeyMacCharCodes	typeChar
	kEventParamKeyCode	typeUInt32
	kEventParamKeyModifiers	typeUInt32
	kEventParamKeyboardType	typeUInt32
kEventRawKeyUp	kEventParamKeyMacCharCodes	typeChar
	kEventParamKeyCode	typeUInt32
	kEventParamKeyModifiers	typeUInt32
	kEventParamKeyboardType	typeUInt32
kEventRawKey-ModifiersChanged	kEventParamKeyModifiers	typeUInt32
kEventHotKeyPressed	kEventParamDirectObject	typeEventHotKeyID
kEventHotKeyReleased	kEventParamDirectObject	typeEventHotKeyID

Key Modifier Event Masks

Define values used to determine whether additional modifier keys are down for a keyboard or mouse event.

```
enum {
    kEventKeyModifierNumLockMask = 1L << kEventKeyModifierNumLockBit,
    kEventKeyModifierFnMask = 1L << kEventKeyModifierFnBit
};
```

Constants

`kEventKeyModifierNumLockMask`

A bit mask containing `kEventKeyModifierNumLockBit`. (Mac OS X only)

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventKeyModifierFnMask`

A bit mask containing `kEventKeyModifierFnBit`. (Mac OS X only)

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

Key Modifier Event Bits

Define key modifier change event bits.

```
enum {
    kEventKeyModifierNumLockBit = 16,
    kEventKeyModifierFnBit = 17
};
```

Constants

`kEventKeyModifierNumLockBit`

This keyboard event was generated either on the numeric keypad or in the numeric section of an iBook or PowerBook keyboard with the Num Lock key pressed. This state bit does not provide an indication of the Num Lock key on non-portable keyboards. This bit is set on Mac OS X only.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventKeyModifierFnBit`

The Fn key was pressed when this keyboard event was generated. This bit is set on Mac OS X only.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

Discussion

Note that bits 8 through 15 (`cmdKeyBit` to `rightControlKeyBit`) are compatible with the Classic Event Manager modifiers.

Keyboard Event Parameters and Types

Define constants for parameters to raw keyboard events.

```
enum {
    kEventParamKeyCode = 'kcod',
    kEventParamKeyMacCharCodes = 'kchr',
    kEventParamKeyModifiers = 'kmod',
    kEventParamKeyUnicodes = 'kuni',
    kEventParamKeyboardType = 'kbdt',
    typeEventHotKeyID = 'hkid'
};
```

Constants

`kEventParamKeyCode`

typeUInt32

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventParamKeyMacCharCodes`

typeChar

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventParamKeyModifiers`

typeUInt32

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventParamKeyUnicodes`
typeUnicodeText
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`typeEventHotKeyID`
EventHotKeyID
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

Symbolic Hot Key Definitions

Define CFDictionaryRef keys returned by `CopySymbolicHotKeys`.

```
#define kHISymbolicHotKeyCode CFSTR("kHISymbolicHotKeyCode")
#define kHISymbolicHotKeyModifiers CFSTR("kHISymbolicHotKeyModifiers")
#define kHISymbolicHotKeyEnabled CFSTR("kHISymbolicHotKeyEnabled")
```

Constants

`kHISymbolicHotKeyCode`
The virtual key code of the hot key, represented as a CFNumber.
Available in Mac OS X v10.3 and later.
Declared in `CarbonEvents.h`.

`kHISymbolicHotKeyModifiers`
The hot key's keyboard modifiers, represented as a CFNumber.
Available in Mac OS X v10.3 and later.
Declared in `CarbonEvents.h`.

`kHISymbolicHotKeyEnabled`
The enable state of the hot key, represented as a CFBoolean.
Available in Mac OS X v10.3 and later.
Declared in `CarbonEvents.h`.

Hot Key Constants

Define hot key states used by `PushSymbolicHotKeyMode`.

```
enum {
    kHIHotKeyModeAllEnabled = 0,
    kHIHotKeyModeAllDisabled = (1 << 0),
    kHIHotKeyModeAllDisabledExceptUniversalAccess = (1 <<  1)
};
```

Constants

`kHIHotKeyModeAllEnabled`
All hot keys are enabled.
Available in Mac OS X v10.4 and later.
Declared in `CarbonEvents.h`.

`kHIHotKeyModeAllDisabled`
All hot keys are disabled.

Available in Mac OS X v10.4 and later.

Declared in `CarbonEvents.h`.

`kHIHotKeyModeAllDisabledExceptUniversalAccess`
All hot keys are disabled except for the Universal Access hot keys (that is, zooming, white-on-black, and enhanced contrast).

Available in Mac OS X v10.4 and later.

Declared in `CarbonEvents.h`.

Menu Events

Menu Event Constants

Define constants related to events from `kEventClassMenu`.

```
enum {
    kEventMenuBeginTracking = 1,
    kEventMenuEndTracking = 2,
    kEventMenuChangeTrackingMode = 3,
    kEventMenuOpening = 4,
    kEventMenuClosed = 5,
    kEventMenuTargetItem = 6,
    kEventMenuMatchKey = 7,
    kEventMenuEnableItems = 8,
    kEventMenuPopulate = 9,
    kEventMenuMeasureItemWidth = 100,
    kEventMenuMeasureItemHeight = 101,
    kEventMenuDrawItem = 102,
    kEventMenuDrawItemContent = 103,
    kEventMenuDispose = 1001,
    kEventMenuCalculateSize = 1004,
    kEventMenuCreateFrameView = 1005,     kEventMenuGetFrameBounds = 1006,
kEventMenuBecomeScrollable = 1007,     kEventMenuCeaseToBeScrollable = 1008,
    kEventMenuBarShown = 2000,     kEventMenuBarHidden = 2001
};
```

Constants

`kEventMenuBeginTracking`
The user has begun tracking the menubar or a pop-up menu. The direct object parameter is a valid menu reference if tracking a pop-up menu, or `NULL` if tracking the menubar. The `kEventParamCurrentMenuTrackingMode` parameter indicates whether the user is tracking the menus using the mouse or the keyboard. The handler may return `userCanceledErr` to stop menu tracking.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventMenuEndTracking`

The user has finished tracking the menubar or a pop-up menu. In Mac OS X v10.3 and later, when a menu tracking session ends, the Menu Manager sends `kEventMenuEndTracking` to every menu that was opened during the session, in addition to the root menu. This is done to allow menus with dynamic content to remove that content at the end of menu tracking; for example, a menu containing many IconRefs might want to load the IconRefs dynamically in response to the `kEventMenuMenuPopulate` event and remove them in response to the `kEventMenuEndTracking` event to avoid the memory overhead of keeping the IconRef data in memory while the menu is not being displayed.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventMenuChangeTrackingMode`

The user has switched from selecting a menu with the mouse to selecting with the keyboard, or from selecting with the keyboard to selecting with the mouse.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventMenuOpening`

A menu is opening. This event is sent each time that the menu is opened (that is, more than once during a given tracking session if the user opens the menu multiple times). It is sent before the menu is actually drawn, so you can update the menu contents (including making changes that will alter the menu size) and the new contents will be drawn correctly. The `kEventParamMenuFirstOpen` parameter indicates whether this is the first time this menu has been opened during this menu tracking session. The handler may return `userCanceledErr` to prevent this menu from opening. Note that for most applications, you should handle the `kEventMenuPopulate` event instead.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventMenuClosed`

A menu has been closed. Sent after the menu is hidden.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventMenuTargetItem`

The mouse is moving over a particular menu item. This event is sent for both enabled and disabled items.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventMenuMatchKey`

A menu is about to be examined for items that match a command key event. A handler for this event may perform its own command key matching and override the Menu Manager's default matching algorithms. Returning `noErr` from your handler indicates that you have found a match. The handler for this event should not examine submenus of this menu for a match; a separate event will be sent for each submenu.

When called from `IsMenuKeyEvent`, the `kEventParamEventRef` parameter contains the event reference that was passed to `IsMenuKeyEvent`, for your handler to examine; when called from `MenuKey` or `MenuEvent`, the `EventRef` parameter contains an event created from the information passed to `MenuKey` or `MenuEvent`. Note that in the `MenuKey` case, no virtual key code (`kEventParamKeyCode`) or key modifiers (`kEventParamKeyModifiers`) will be available.

The `kEventParamMenuEventOptions` parameter contains a copy of the options that were passed to `IsMenuKeyEvent`, or `0` if called from `MenuKey` or `MenuEvent`. The only option that your handler will need to obey is `kMenuEventIncludeDisabledItems`.

If your handler finds a match, it should set the `kEventParamMenuItemIndex` parameter to contain the item index of the matching item, and return `noErr`. If it does not find a match, it should return `menuItemNotFoundErr`. Any other return value will cause the Menu Manager to use its default command key matching algorithm for this menu.

This event is sent after `kEventMenuEnableItems`.

In CarbonLib and Mac OS X through version 10.3, the Menu Manager sends a `kEventMenuEnableItems` event to the menu before sending `kEventMenuMatchKey`. In Mac OS X 10.4 and later, the Menu Manager no longer sends `kEventMenuEnableItems` (or the resulting `kEventCommandUpdateStatus` events) to the menu; a handler for `kEventMenuMatchKey` is expected to determine on its own whether a matching menu item is enabled.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventMenuEnableItems`

A request that the items in the menu be properly enabled or disabled according to the current state of the application. This event is sent from inside `MenuKey`, `MenuEvent`, and `IsMenuKeyEvent` before those functions examine the menu for an item that matches a keyboard event. It is also sent during menu tracking before a menu is first made visible; it is sent right after `kEventMenuOpening`, once per menu per menu tracking session.

If you install an event handler for `kEventProcessCommand`, you should also install a handler for `kEventMenuEnableItems`. This is necessary because the Carbon event system will attempt to match command keys against the available menus before returning the keyboard event to your application via `WaitNextEvent`. If you have menu command event handlers installed for your menu items, your handlers will be called without you ever receiving the keyboard event or calling `MenuKey`/`MenuEvent`/`IsMenuKeyEvent` yourself. Therefore, you have no opportunity to enable your menu items properly other than from a `kEventMenuEnableItems` handler.

It is not necessary to handle this event if you do not install `kEventProcessCommand` handlers for your menu items; in that case, the command key event will be returned from `WaitNextEvent` or `ReceiveNextEvent` as normal, and you can set up your menus before calling `MenuKey`/`MenuEvent`/`IsMenuKeyEvent`.

The `kEventParamEnableMenuForKeyEvent` parameter indicates whether this menu should be enabled for key event matching (`true`) or because the menu itself is about to become visible (`false`). If true, only the item enable state, command key, command key modifiers, and (optionally) the command key glyph need to be correct. If false, the entire menu item contents must be correct. This may be useful if you have custom menu content that is expensive to prepare.

Note that the standard application handler for `kEventMenuEnableItems` automatically sends `kEventCommandUpdateStatus` to your menu commands. As this can cause a performance hit if you have many menu items, you can choose to bypass these updates by installing a no-op handler for `kEventMenuEnableItems` that simply returns `noErr`.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventMenuPopulate`

Sent when an application should dynamically create a menu. You should use this event instead of `kEventMenuOpening` as the Menu Manager sends it before it searches a menu for a matching command key sequence; therefore you can use this event to dynamically add menu items that have command-key equivalents, making them selectable even if the menu itself is never displayed. The `kEventMenuPopulate` event is sent just once during a menu tracking session, even if the menu is opened and closed multiple times; the `kEventWindowOpening` event is sent each time the menu opens.

To determine whether this event was triggered by a command-key sequence, you should examine the `kEventParamMenuContext` parameter for the `kMenuContextKeyMatching` flag. If the event corresponds to the actual display of a menu, the `kEventContextMenuBarTracking` or `kEventContextPopUpTracking` flags are set.

Note that in Mac OS X v10.2 and later, the Menu Manager sends this event before it searches a menu for a matching command ID. To determine if this event was triggered by a command ID, check for the `kMenuContextCommandIDSearch` flag in the `kEventParamMenuContext` parameter. If that flag is set, the command ID that triggered this event is contained in the `kEventParamMenuCommand` parameter.)

Available in Mac OS X v10.1 and later.

Declared in `CarbonEvents.h`.

`kEventMenuMeasureItemWidth`

Sent by the standard window definition if a menu item has the `kMenuItemAttrCustomDraw` attribute set. You should install your handler directly on the menu. Your handler should return a customized width for the menu item in the `kEventParamMenuItemWidth` parameter.

The default behavior (if you are using the standard menu definition) is to return the standard width for the item.

Available in Mac OS X v10.1 and later.

Declared in `CarbonEvents.h`.

`kEventMenuMeasureItemHeight`

Sent by the standard window definition if a menu item has the `kMenuItemAttrCustomDraw` attribute set. You should install your handler directly on the menu. Your handler should return a customized height for the menu item in the `kEventParamMenuItemHeight` parameter.

The default behavior (if you are using the standard menu definition) is to return the standard height for the item.

Available in Mac OS X v10.1 and later.

Declared in `CarbonEvents.h`.

`kEventMenuDrawItem`

Sent by the standard window definition if a menu item has the `kMenuItemAttrCustomDraw` attribute set. You should install your handler directly on the menu. Use your handler to override the drawing of the menu item and background.

The default behavior (if you are using the standard menu definition) is to call the Appearance Manager function `DrawThemeMenuItem` to draw the menu item's background and content.

If you have the standard event handler installed, the event also contains additional parameters indicating the bounds of the various portions of the menu item content, as well as the baseline of the menu item text. Using these parameters, you can call `CallNextEventHandler` to let the system handlers draw the standard menu content and then your handler can draw custom content on top.

Available in Mac OS X v10.1 and later.

Declared in `CarbonEvents.h`.

`kEventMenuDrawItemContent`

Sent by the standard window definition if a menu item has the `kMenuItemAttrCustomDraw` attribute set. You should install your handler directly on the menu. You use your handler to override the drawing of one or more parts of the menu item's content (that is the mark character, icon, text, and command key information).

When you receive this event, the background and highlighting (if applicable) has already been drawn using the standard system appearance.

The default behavior (if you are using the standard menu definition) is to draw the standard menu item content.

If you have the standard event handler installed, the event also contains additional parameters indicating the bounds of the various portions of the menu item content, as well as the baseline of the menu item text. Using these parameters, you can call `CallNextEventHandler` to let the system handlers draw the standard menu content and then your handler can draw custom content on top.

Available in Mac OS X v10.1 and later.

Declared in `CarbonEvents.h`.

`kEventMenuDispose`

Sent when a menu is being disposed.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventMenuCalculateSize`

Sent by `CalcMenuSize` to request that the menu calculate its size. The Menu Manager provides a default handler for all menus that calls the menu's MDEF or menu content view to determine the menu size. Applications typically do not need to handle this event. A custom menu definition or menu content view should use `kMenuSizeMsg` or `kEventControlGetOptimalBounds` to calculate its size. Note that if the menu uses an MDEF, the MDEF sets the menu's width and height in response to `kMenuSizeMsg`. The default handler for this event saves the old width and height before calling the MDEF and restores them afterward. `CalcMenuSize` sets the final menu width and height based on the dimensions returned from this event; applications may override this event to customize the width or height of a menu by modifying the `kEventParamDimensions` parameter. This event is sent only to the menu and is not propagated past the menu.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`kEventMenuCreateFrameView`

Requests that a menu content view create the HIView that will be used to draw the menu window frame. The HIMenuView class provides a default handler for this event that creates an instance of the standard menu window frame view. This event is sent only to the menu content view and is not propagated past the view.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`kEventMenuGetFrameBounds`

Requests that a menu content view calculate the bounding rect, in global coordinates, of the menu window frame that should contain the menu. This event is sent by the Menu Manager before displaying pull-down, popup, and hierarchical menus. It provides an opportunity for the menu content view to determine the position of the menu frame based on the position of the menu title, parent menu item, or popup menu location. The HIMenuView class provides a default handler for this event that calculates an appropriate location based on the bounds of the menu, the available screen space, and the frame metrics of the menu window content view. This event is sent only to the menu content view and is not propagated past the view.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`kEventMenuBecomeScrollable`

Requests that a menu content view prepare to be scrollable, which it does by installing the appropriate event handlers, timers, and the like. This event is sent by the Menu Manager when a menu becomes the most recently opened menu in the menu hierarchy. It is an indication that this menu content view is now a candidate for scrolling. The Menu Manager provides a default handler for this event that installs event handlers to provide automatic scrolling behavior for HIView-based menus. If a menu content view does not wish to use the Menu Manager's default scrolling support, it can override this event and return `noErr` to prevent the event from being propagated to the Menu Manager's default handler. This event is sent only to the menu content view and is not propagated past the view.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`kEventMenuCeaseToBeScrollable`
: Requests that a menu content view cease to be scrollable. This event is sent by the Menu Manager when a menu ceases to be the most recently opened menu. This occurs when the menu is closed, or when a submenu of the most recently opened menu is opened. It is an indication that this menu content view is no longer a candidate for scrolling. The Menu Manager provides a default handler for this event that removes event handlers installed in response to `kEventMenuBecomeScrollable`. This event is sent only to the menu content view and is not propagated past the view.

  Available in Mac OS X v10.3 and later.

  Declared in `CarbonEvents.h`.

`kEventMenuBarShown`
: Sent to all handlers registered for this event when the front process shows its menubar. This event is sent only to the application target.

  Available in Mac OS X v10.3 and later.

  Declared in `CarbonEvents.h`.

`kEventMenuBarHidden`
: Sent to all handlers registered for this event when the front process hides its menubar. This event is sent only to the application target.

  Available in Mac OS X v10.3 and later.

  Declared in `CarbonEvents.h`.

Discussion
Some menu events are sent or handled by the standard menu definition, which is a collection of handlers that define the default menu behavior. If you have specified a custom menu definition, you will not get the behavior provided by the standard definition.

Table 3-7 shows the parameters associated with menu events.

Table 3-7 Parameter names and types for menu event kinds

Event kind	Parameter name	Parameter type
`kEventMenuBeginTracking`	`kEventParamDirectObject`	`typeMenuRef`
	`kEventParamCurrentMenuTrackingMode`	`typeMenuTrackingMode`
	`kEventParamMenuContext` (Mac OS X v10.1 and later and CarbonLib 1.5 and later.)	`typeUInt32`
`kEventMenuEndTracking`	`kEventParamDirectObject`	`typeMenuRef`
	`kEventParamMenuContext` (Mac OS X v10.1 and later and CarbonLib 1.5 and later.)	`typeUInt32`
	`kEventParamMenuDismissed` (Mac OS X v10.3 and later.)	`typeUInt32`
`kEventMenuChangeTracking Mode`	`kEventParamDirectObject`	`typeMenuRef`
	`kEventParamCurrentMenuTrackingMode`	`typeMenuTrackingMode`

	kEventParamNewMenuTrackingMode	typeMenuTrackingMode
	kEventParamMenuContext	typeUInt32
kEventMenuOpening	kEventParamDirectObject	typeMenuRef
	kEventParamMenuFirstOpen	typeBoolean
	kEventParamMenuContext (Mac OS X v 10.0 and later and CarbonLib 1.5 and later.)	typeUInt32
kEventMenuClosed	kEventParamDirectObject	typeMenuRef
	kEventParamMenuContext (Mac OS X v10.1 and later and CarbonLib 1.5 an later.)	typeUInt32
kEventMenuTargetItem	kEventParamDirectObject	typeMenuRef
	kEventParamMenuItemIndex	typeMenuItemIndex
	kEventParamMenuCommand	typeMenuCommand
	kEventParamMenuContext (Mac OS X v10.1 and later and CarbonLib 1.5 and later.)	typeUInt32
kEventMenuMatchKey	kEventParamDirectObject	typeMenuRef
	kEventParamEventRef	typeEventRef
	kEventParamMenuEventOptions	typeMenuEventOptions
	kEventParamMenuContext (Mac OS X v10.1 and later, and CarbonLib 1.5 and later.)	typeUInt32
	kEventParamMenuItemIndex	typeMenuItemIndex
kEventMenuEnable-Items	kEventParamDirectObject	typeMenuRef
	kEventParamEnableMenuForKeyEvent	typeBoolean
	kEventParamMenuContext (Mac OS X v10.0 and later, and CarbonLib 1.1 and later.)	typeUInt32
kEventMenuPopulate	kEventParamDirectObject	typeMenuRef
	kEventParamMenuContext	typeUInt32
	kEventParamMenuCommand (Mac OS X v10.2 and later, and CarbonLib 1.6 and later.)	typeMenuCommand
kEventMenuMeasure-ItemWidth	kEventParamDirectObject	typeMenuRef
	kEventParamMenuItemIndex	typeMenuItemIndex

	kEventParamMenuItemWidth	typeShortInteger
kEventMenuMeasure-ItemHeight	kEventParamDirectObject	typeMenuRef
	kEventParamMenuItemIndex	typeMenuItemIndex
	kEventParamMenuItemHeight	typeShortInteger
kEventMenuDrawItem	kEventParamDirectObject	typeMenuRef
	kEventParamCurrentBounds	typeQDRectangle
	kEventParamMenuItemIndex	typeMenuItemIndex
	kEventParamMenuItemBounds	typeQDRectangle
	kEventParamMenuVirtualTop	typeLongInteger
	kEventParamMenuVirtualBottom	typeLongInteger
	kEventParamMenuDrawState	typeThemeMenuState
	kEventParamMenuItemType	typeThemeMenuItemType
	kEventParamCGContextRef	typeCGContextRef
	kEventParamMenuMarkBounds	typeQDRectangle
	kEventParamMenuIconBounds	typeQDRectangle
	kEventParamMenuTextBounds	typeQDRectangle
	kEventParamMenuTextBaseline	typeShortInteger
	kEventParamMenuKeyCommandKeyBounds	typeQDRectangle
kEventMenuDraw-ItemContent	kEventParamDirectObject	typeMenuRef
	kEventParamMenuItemIndex	typeMenuItemIndex
	kEventParamMenuItemBounds	typeQDRectangle
	kEventParamDeviceDepth	typeShortInteger
	kEventParamDeviceColor	typeBoolean
	kEventParamCGContextRef	typeCGContextRef
	kEventParamMenuMarkBounds	typeQDRectangle
	kEventParamMenuIconBounds	typeQDRectangle
	kEventParamMenuTextBounds	typeQDRectangle

	kEventParamMenuTextBaseline	typeShortInteger
	kEventParamMenuKeyCommandKeyBounds	typeQDRectangle
kEventMenuDispose	kEventParamDirectObject	typeMenuRef
kEventMenuCalculate-Size	kEventParamDirectObject	typeMenuRef
	kEventParamControlRef	typeControlRef
	kEventParamGDevice	typeGDHandle
	kEventParamAvailableBounds	typeQDRectangle
	kEventParamDimensions	typeHISize
kEventMenuCreate-FrameView	kEventParamEventRef	typeEventRef
	kEventParamMenuType	typeThemeMenuType
	kEventParamMenuFrameView	typeControlRef
kEventMenuGet-FrameBounds	kEventParamMenuType	typeThemeMenuType
	kEventParamMenuIsPopup	typeBoolean
	kEventParamMenuFrameView	typeControlRef
	kEventParamMenuDirection	typeMenuDirection
	kEventParamMenuItemBounds	typeHIRect
	kEventParamGDevice	typeGDHandle
	kEventParamAvailableBounds	typeHIRect
	kEventParamParentMenu	typeMenuRef
	kEventParamParentMenuItem	typeMenuItemIndex
	kEventParamMenuPopupItem	typeMenuItemIndex
	kEventParamBounds	typeHIRect
	kEventParamOrigin	typeHIPoint
kEventMenuBecome-Scrollable	*None*	
kEventMenuCease-ToBeScrollable	*None*	
kEventMenuBarShown	*None*	

kEventMenuBarHidden	*None*	

Menu Context Constants

Define constants that describe the usage or context of a menu event.

```
enum {
    kMenuContextMenuBar = 1 << 0,
    kMenuContextPullDown = 1 << 8,
    kMenuContextPopUp = 1 << 9,
    kMenuContextSubmenu = 1 << 10,
    kMenuContextMenuBarTracking = 1 << 16,
    kMenuContextPopUpTracking = 1 << 17,
    kMenuContextKeyMatching = 1 << 18,
    kMenuContextMenuEnabling = 1 << 19,
    kMenuContextCommandIDSearch = 1 << 20
};
```

Constants

`kMenuContextMenuBar`
The menu associated with this event is in the menu bar or is a submenu of a menu in the menubar.

Available in Mac OS X v10.1 and later.

Declared in `CarbonEvents.h`.

`kMenuContextPulldown`
The menu associated with this event is a pulldown menu located in the menu bar.

`kMenuContextPopUp`
The menu associated with this event is a popup menu displayed by the Menu Manager function `PopUpMenuSelect`.

Available in Mac OS X v10.1 and later.

Declared in `CarbonEvents.h`.

`kMenuContextSubmenu`
The menu associated with this event is a submenu of a pulldown or popup menu.

Available in Mac OS X v10.1 and later.

Declared in `CarbonEvents.h`.

`kMenuContextMenuBarTracking`
This event is being sent while a menu is being tracked in the menu bar.

Available in Mac OS X v10.1 and later.

Declared in `CarbonEvents.h`.

`kMenuContextPopUpTracking`
This event is being sent while a popup menu is being tracked.

Available in Mac OS X v10.1 and later.

Declared in `CarbonEvents.h`.

`kMenuContextKeyMatching`
This event is being sent while trying to match a command-key equivalent to a menu item.

Available in Mac OS X v10.1 and later.

Declared in `CarbonEvents.h`.

`kMenuContextMenuEnabling`
Sent at idle time to update the enabled state of the menus.

Available in Mac OS X v10.1 and later.

Declared in `CarbonEvents.h`.

`kMenuContextCommandIDSearch`
Sent while trying to match a command ID using the Menu Manager function `CountMenuItemsWithCommandID` or `GetIndMenuItemWithCommandID`.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

Discussion
The bits corresponding to these constants are set in the `kEventParamMenuContext` parameter of the menu event.

Menu Event Parameters

Define constants for parameters to menu events.

```
enum {
    kEventParamCurrentMenuTrackingMode = 'cmtm',
    kEventParamNewMenuTrackingMode = 'nmtm',
    kEventParamMenuFirstOpen = '1sto',
    kEventParamMenuItemIndex = 'item',
    kEventParamMenuCommand = 'mcmd',
    kEventParamEnableMenuForKeyEvent = 'fork',
    kEventParamMenuEventOptions = 'meop',
    kEventParamMenuContext = 'mctx',
    kEventParamMenuItemBounds = 'mitb',
    kEventParamMenuMarkBounds = 'mmkb',
    kEventParamMenuIconBounds = 'micb',
    kEventParamMenuTextBounds = 'mtxb',
    kEventParamMenuTextBaseline = 'mtbl',
    kEventParamMenuCommandKeyBounds = 'mcmb',
    kEventParamMenuVirtualTop = 'mvrt',
    kEventParamMenuVirtualBottom = 'mvrb',
    kEventParamMenuDrawState = 'mdrs',
    kEventParamMenuItemType = 'mitp',
    kEventParamMenuItemWidth = 'mitw',
    kEventParamMenuItemHeight = 'mith',
    typeMenuItemIndex = 'midx',
    typeMenuCommand = 'mcmd',
    typeMenuTrackingMode = 'mtmd',
    typeMenuEventOptions = 'meop',
    typeThemeMenuState = 'tmns',
    typeThemeMenuItemType = 'tmit'
};
```

Constants
`kEventParamCurrentMenuTrackingMode`
typeMenuTrackingMode

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventParamNewMenuTrackingMode`
typeMenuTrackingMode
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamMenuFirstOpen`
typeBoolean
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamMenuItemIndex`
typeMenuItemIndex
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamMenuCommand`
typeMenuCommand
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamEnableMenuForKeyEvent`
typeBoolean
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamMenuEventOptions`
typeMenuEventOptions
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`typeMenuItemIndex`
MenuItemIndex
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`typeMenuCommand`
MenuCommand
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`typeMenuTrackingMode`
MenuTrackingMode
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`typeMenuEventOptions`
MenuEventOptions
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

Services Menu Command Keys

Define CFDictionaryRef keys returned by `CopyServicesMenuCommandKeys`.

```
#define kHIServicesMenuProviderName CFSTR("kHIServicesMenuProviderName")
#define kHIServicesMenuItemName CFSTR("kHIServicesMenuItemName")
#define kHIServicesMenuCharCode CFSTR("kHIServicesMenuCharCode")
#define kHIServicesMenuKeyModifiers CFSTR("kHIServicesMenuKeyModifiers")
```

Constants

`kHIServicesMenuProviderName`
The name of the service provider.
Available in Mac OS X v10.4 and later.
Declared in `CarbonEvents.h`.

`kHIServicesMenuItemName`
The name of the menu item.
Available in Mac OS X v10.4 and later.
Declared in `CarbonEvents.h`.

`kHIServicesMenuCharCode`
The character code of the menu item shortcut.
Available in Mac OS X v10.4 and later.
Declared in `CarbonEvents.h`.

`kHIServicesMenuKeyModifiers`
The keyboard modifiers of the menu item shortcut in Menu Manager modifiers format.
Available in Mac OS X v10.4 and later.
Declared in `CarbonEvents.h`.

Mouse Events

Mouse Events

Define constants related to events from `kEventClassMouse`.

```
enum {
    kEventMouseDown = 1,
    kEventMouseUp = 2,
    kEventMouseMoved = 5,
    kEventMouseDragged = 6,
    kEventMouseEntered = 8,
    kEventMouseExited = 9,
    kEventMouseWheelMoved = 10
};
```

Constants

`kEventMouseDown`
A mouse button was pressed. Note that if you install a handler for this event on a window, you must allow the event to propagate (either by calling `CallNextEventHandler` or returning `eventNotHandledErr`) so that the window can be activated.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventMouseUp`
A mouse button was released.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventMouseMoved`
The mouse moved.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventMouseDragged`
The mouse moved, and a button was down.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventMouseEntered`
The mouse entered a tracking region. Used with mouse tracking regions. See `CreateMouseTrackingRegion` (page 253) for more information.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

`kEventMouseExited`
The mouse left a tracking region. Used with mouse tracking regions. See `CreateMouseTrackingRegion` (page 253) for more information.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

`kEventMouseWheelMoved`
The mouse wheel moved.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

Discussion

Table 3-8 shows the parameters related to mouse events.

Table 3-8 Parameter names and types for mouse event kinds

Event kind	Parameter name	Parameter type
kEventMouseDown	kEventParamMouseLocation	typeQDPoint
	kEventParamKeyModifiers	typeUInt32
	kEventParamMouseButton	typeMouseButton
	kEventParamClickCount	typeUInt32
	kEventParamWindowRef	typeWindowRef
	kEventParamWindowMouseLocation	typeHIPoint
	kEventParamWindowPartCode	typeWindowPartCode
	kEventParamMouseChord	typeUInt32
	kEventParamTabletEventType	typeUInt32
	kEventParamTabletPointRec	typeTabletPointRec
	kEventParamTabletProximityRec	typeTabletProximityRec
kEventMouseUp	kEventParamMouseLocation	typeQDPoint
	kEventParamKeyModifiers	typeUInt32
	kEventParamMouseButton	typeMouseButton
	kEventParamWindowRef	typeWindowRef
	kEventParamWindowMouseLocation	typeHIPoint
	kEventParamWindowPartCode	typeWindowPartCode
	kEventParamClickCount	typeUInt32
	kEventParamMouseChord	typeUInt32
	kEventParamTabletEventType	typeUInt32
	kEventParamTabletPointRec	typeTabletPointRec
	kEventParamTabletProximityRec	typeTabletProximityRec
kEventMouseMoved	kEventParamMouseLocation	typeQDPoint
	kEventParamKeyModifiers	typeUInt32
	kEventParamMouseDelta	typeHIPoint
	kEventParamWindowRef	typeWindowRef

	kEventParamWindowMouseLocation	typeHIPoint
	kEventParamWindowPartCode	typeWindowPartCode
	kEventParamTabletEventType	typeUInt32
	kEventParamTabletPointRec	typeTabletPointRec
	kEventParamTabletProximityRec	typeTabletProximityRec
kEventMouseDragged	kEventParamMouseLocation	typeQDPoint
	kEventParamKeyModifiers	typeUInt32
	kEventParamMouseButton	typeMouseButton
	kEventParamMouseChord	typeUInt32
	kEventParamMouseDelta	typeHIPoint
	kEventParamWindowRef	typeWindowRef
	kEventParamWindowMouseLocation	typeHIPoint
	kEventParamWindowPartCode	typeWindowPartCode
	kEventParamTabletEventType	typeUInt32
	kEventParamTabletPointRec	typeTabletPointRec
	kEventParamTabletProximityRec	typeTabletProximityRec
kEventMouseEntered	kEventParamMouseTrackingRef	typeMouseTrackingRef
	kEventParamMouseLocation	typeQDPoint
	kEventParamKeyModifiers	typeUInt32
	kEventParamWindowRef	typeWindowRef
	kEventParamWindowMouseLocation	typeHIPoint
kEventMouseExited	kEventParamMouseTrackingRef	typeMouseTrackingRef
	kEventParamMouseLocation	typeQDPoint
	kEventParamKeyModifiers	typeUInt32
	kEventParamWindowRef	typeWindowRef
	kEventParamWindowMouseLocation	typeHIPoint
kEventMouse-WheelMoved	kEventParamMouseLocation	typeHIPoint
	kEventParamKeyModifiers	typeUInt32

	kEventParamWindowRef	typeWindowRef
	kEventParamWindowMouseLocation	typeHIPoint
	kEventParamWindowPartCode	typeWindowPartCode
	kEventParamMouseWheelAxis	typeMouseWheelAxis
	kEventParamMouseWheelDelta	typeSInt32

Mouse Button Constants

Define mouse button constants.

```
typedef UInt16 EventMouseButton;
enum {
    kEventMouseButtonPrimary = 1,
    kEventMouseButtonSecondary = 2,
    kEventMouseButtonTertiary = 3
};
```

Constants

`kEventMouseButtonPrimary`

The primary mouse button (default for one-button mice, typically the left button for multi-button mice).

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventMouseButtonSecondary`

The “right-click” mouse button.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventMouseButtonTertiary`

The tertiary mouse button.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

Mouse Wheel Constants

Define mouse scroll-wheel-axis constants.

```
typedef UInt16 EventMouseWheelAxis;
enum {
    kEventMouseWheelAxisX = 0,
    kEventMouseWheelAxisY = 1
};
```

Constants

`kEventMouseWheelAxisX`
The x-axis (left-right movement).

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventMouseWheelAxisY`
The y-axis (up-down movement).

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

Mouse Tracking Region Options

Define constants used by the `CreateMouseTrackingRegion` function.

```
typedef UInt32 MouseTrackingOptions;
enum {
    kMouseTrackingOptionsLocalClip = 0,
    kMouseTrackingOptionsGlobalClip = 1,
    kMouseTrackingOptionsStandard = kMouseTrackingOptionsLocalClip
};
```

Constants

`kMouseTrackingOptionsLocalClip`
The region passed to `CreateMouseTrackingRegion` (page 253) is defined in local coordinates, and that the region is clipped to the owning window's content region.

Available in Mac OS X v10.2 and later.

Not available to 64-bit applications.

Declared in `CarbonEvents.h`.

`kMouseTrackingOptionsGlobalClip`
The region passed to `CreateMouseTrackingRegion` (page 253) is defined in global coordinates and that the region is clipped to the owning window's structure region.

Available in Mac OS X v10.2 and later.

Not available to 64-bit applications.

Declared in `CarbonEvents.h`.

`kMouseTrackingOptionsStandard`
Same as `kMouseTrackingOptionsLocalClip`.

Available in Mac OS X v10.2 and later.

Not available to 64-bit applications.

Declared in `CarbonEvents.h`.

Alternate Mouse Tracking Result Constants

Define constants for alternate mouse tracking results.

```
enum {
    kMouseTrackingMousePressed = kMouseTrackingMouseDown,
    kMouseTrackingMouseReleased = kMouseTrackingMouseUp
};
```

Constants

`kMouseTrackingMousePressed`

The user pressed any mouse button.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kMouseTrackingMouseReleased`

The user released the mouse button.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

Mouse Event Parameters

Define constants for parameters to mouse events.

```
enum {
    kEventParamMouseLocation = 'mloc',
    kEventParamWindowMouseLocation = 'wmou',
    kEventParamMouseButton = 'mbtn',
    kEventParamClickCount = 'ccnt',
    kEventParamMouseWheelAxis = 'mwax',
    kEventParamMouseWheelDelta = 'mwdl',
    kEventParamMouseDelta = 'mdta',
    kEventParamMouseChord = 'chor',
    kEventParamTabletEventType = 'tblt',
    kEventParamMouseTrackingRef = 'mtrf',
    typeMouseButton = 'mbtn',
    typeMouseWheelAxis = 'mwax',
    typeMouseTrackingRef = 'mtrf'
};
```

Constants

`kEventParamMouseLocation`

typeQDPoint

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventParamMouseButton`

typeMouseButton

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventParamClickCount`
typeUInt32
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamMouseWheelAxis`
typeMouseWheelAxis
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamMouseWheelDelta`
typeSInt32
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamMouseDelta`
typeQDPoint
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamMouseChord`
typeUInt32
Available in Mac OS X v10.1 and later.
Declared in `CarbonEvents.h`.

`typeMouseButton`
EventMouseButton
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`typeMouseWheelAxis`
EventMouseWheelAxis
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

Mouse Tracking Option Constant

Define options for the `TrackMouseLocationWithOptions` function.

```
enum {
    kTrackMouseLocationOptionDontConsumeMouseUp = (1 << 0)
};
```

Constants

`kTrackMouseLocationOptionDontConsumeMouseUp`
Leave mouse-up events in the event queue (the default is to pull them.)
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

Discussion

This constant can be passed to `TrackMouseLocationWithOptions` (page 304) in the `inOptions` parameter.

Mouse Tracking Constants

Define constants for mouse tracking.

```
typedef UInt16 MouseTrackingResult;
enum {
    kMouseTrackingMouseDown = 1,
    kMouseTrackingMouseUp = 2,
    kMouseTrackingMouseExited = 3,
    kMouseTrackingMouseEntered = 4,
    kMouseTrackingMouseDragged = 5,
    kMouseTrackingKeyModifiersChanged = 6,
    kMouseTrackingUserCancelled = 7,
    kMouseTrackingTimedOut = 8,
    kMouseTrackingMouseMoved = 9
};
```

Constants

`kMouseTrackingMouseDown`
The user pressed any mouse button.

Available in Mac OS X v10.1 and later.

Declared in `CarbonEvents.h`.

`kMouseTrackingMouseUp`
The user released the mouse button.

Available in Mac OS X v10.1 and later.

Declared in `CarbonEvents.h`.

`kMouseTrackingMouseExited`
The mouse exited the specified region.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kMouseTrackingMouseEntered`
The mouse entered the specified region.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kMouseTrackingMouseDragged`
The mouse moved while the mouse button was down.

Available in Mac OS X v10.1 and later.

Declared in `CarbonEvents.h`.

`kMouseTrackingKeyModifiersChanged`
One or more keyboard modifiers (option, control, and so on) for the mouse changed.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kMouseTrackingMouseMoved`
Prior to Mac OS X v10.2, this constant was equivalent to `kMouseTrackingMouseDragged`. In Mac OS X v10.2 and later, `kMouseTrackingMouseMoved` indicates that the mouse moved while the mouse button was up.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

Discussion
These constants are returned by `TrackMouseLocation` (page 303) and `TrackMouseRegion` (page 305), which are designed as replacements to calls such as `StillDown` and `WaitMouseUp`. The advantage over those routines is that `TrackMouseLocation` and `TrackMouseRegion` block if the user is not moving the mouse, whereas mouse tracking loops based on `StillDown` and `WaitMouseUp` spin, chewing up valuable CPU time that could be better spent elsewhere. It is highly recommended that any tracking loops in your application stop using `StillDown` and `WaitMouseUp` and start using `TrackMouseLocation` or `TrackMouseRegion`. See the notes on those functions for more information.

Mouse Tracking Selectors

Define a constant for "sticky" mode used by the `HIMouseTrackingGetParameters` function.

```
enum {
    kMouseParamsSticky = 'stic'
};
```

Constants
`kMouseParamsSticky`
Requests the time and distance for determining "sticky" mouse tracking. When the mouse is clicked on a menu title, the toolbox enters a sticky mouse-tracking mode that varies according to the time and distance between the mouse-down event and the mouse-up event. In this mode, the menu is tracked even though the mouse has already been released.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

Services Manager Constants

Services Manager Events

Define constants related to Services Manager events.

```
enum {
    kEventServiceCopy = 1,
    kEventServicePaste = 2,
    kEventServiceGetTypes = 3,
    kEventServicePerform = 4
};
```

Constants
`kEventServiceCopy`
The user wants to invoke a service that requires your application to provide data. Your application must update the scrap reference in the `kEventParamScrapRef` parameter to indicate the appropriate data from the current selection or user focus. See the Discussion section for additional information about this parameter for Mac OS X v10.3 and later.

Available in Mac OS X v10.1 and later.

Declared in `CarbonEvents.h`.

`kEventServicePaste`
The user has invoked a service that requires your application to receive data. Your application must update the current user focus with the data provided by the `kEventParamScrapRef` parameter. See the Discussion section for additional information about this parameter for Mac OS X v10.3 and later.

Available in Mac OS X v10.1 and later.

Declared in `CarbonEvents.h`.

`kEventServiceGetTypes`
Sent when the Services Manager needs to know what types of data it can cut-and-paste into the scrap. The Services Manager uses this information to update the Services menu, indicating which services are available for the current selection. This event passes two `CFMutableArray` references to you in the `kEventParamServiceCopyTypes` and `kEventParamServicePasteTypes` parameters. You should fill out these arrays with Core Foundation strings indicating which types your application can copy and paste. Note that you can use the `CreateTypeStringWithOSType` (page 255) to create a `CFStringRef` from an `OSType`.

Available in Mac OS X v10.1 and later.

Declared in `CarbonEvents.h`.

`kEventServicePerform`
Sent when your application must perform a service. The `kEventParamScrapRef` parameter holds the scrap information, and the `kEventParamServiceMessageName` parameter contains a Core Foundation string indicating what service was requested. Only applications that can provide services receive this event. See the Discussion section for additional information about this parameter for Mac OS X v10.3 and later.

Available in Mac OS X v10.1 and later.

Declared in `CarbonEvents.h`.

Discussion
In Mac OS X 10.3 and later, the `kEventServiceCopy`, `kEventServicePaste`, and `kEventServicePerform` events include a PasteboardRef and a ScrapRef. A handler for this event should provide its data using either Pasteboard or Scrap Manager APIs, and the corresponding pasteboard or scrap reference, depending on which is more convenient or appropriate. Data only needs to be placed on one of the pasteboard or scrap; it does not need to be placed on both. Data written to the pasteboard is also be available on the scrap, and vice versa.

Table 3-9 Parameter names and types for Service class events

Event kind	Parameter name	Parameter type
`kEventServiceCopy`	`kEventParamPasteboardRef` (Mac OS X v10.3 and later.)	`typePasteboardRef`
	`kEventParamScrapRef` (Mac OS X v10.1 and later.)	`typeScrapRef`
`kEventServicePaste`	`kEventParamPasteboardRef` (Mac OS X v10.3 and later.)	`typePasteboardRef`
	`kEventParamScrapRef` (Mac OS X v10.1 and later.)	`typeScrapRef`
`kEventService-GetTypes`	`kEventParamServiceCopyTypes`	`typeCFMutableArrayRef`

	kEventParamServicePasteTypes	typeCFMutableArrayRef
kEventServicePerform	kEventParamPasteboardRef (Mac OS X v10.3 and later.)	typePasteboardRef
	kEventParamScrapRef (Mac OS X v10.1 and later.)	typeScrapRef
	kEventParamServiceMessageName	typeCFStringRef
	kEventParamServiceUserData	typeCFStringRef

Services Manager Event Parameters

Define constants for parameters to Service Manager events.

```
enum {
    kEventParamScrapRef = 'scrp',
    kEventParamServiceCopyTypes = 'svsd',
    kEventParamServicePasteTypes = 'svpt',
    kEventParamServiceMessageName = 'svmg',
    kEventParamServiceUserData = 'svud',
    typeScrapRef = 'scrp',
    typeCFMutableArrayRef = 'cfma'
};
```

Constants

`kEventParamScrapRef`

When provided as a parameter to the `kEventServicePaste` event, the current selection should be replaced by data from this scrap. When provided as a parameter to `kEventServicePerform`, the scrap that should be used to send and receive data from the requester. When provided as a parameter to the `kEventServiceCopy` event, data from the current selection should be placed into this scrap.

Available in Mac OS X v10.1 and later.

Declared in `CarbonEvents.h`.

`kEventParamServiceCopyTypes`

When provided as a parameter to the `kEventServiceGetTypes` event, add CFString references to this array to report the types that can be pasted from the current selection. These strings will be released automatically after the event is handled.

Available in Mac OS X v10.1 and later.

Declared in `CarbonEvents.h`.

`kEventParamServicePasteTypes`

When provided as a parameter to the `kEventServiceGetTypes` event, add CFString references to this array to report the types that can be copied from the current selection. These strings will be released automatically after the event is handled.

Available in Mac OS X v10.1 and later.

Declared in `CarbonEvents.h`.

`kEventParamServiceMessageName`
When provided as a parameter to the `kEventServicePerform` event, contains the name of the advertised service that was invoked.

Available in Mac OS X v10.1 and later.

Declared in `CarbonEvents.h`.

`kEventParamServiceUserData`
When provided as a parameter to the `kEventServicePerform` event, contains extra data provided by the requestor.

Available in Mac OS X v10.1 and later.

Declared in `CarbonEvents.h`.

Tablet Event Constants

Tablet Events

Define constants for events related to drawing tablets.

```
enum {
    kEventTabletPoint = 1,
    kEventTabletProximity = 2,
    kEventTabletPointer = 1
};
```

Constants

`kEventTabletPoint`
Indicates that the pen has moved on a tablet. (Mac OS X only)

Available in Mac OS X v10.1 and later.

Declared in `CarbonEvents.h`.

`kEventTabletProximity`
Indicates that the pen has entered the proximity region of the tablet. (Mac OS X only)

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventTabletPointer`
Same as `kEventTabletPoint`. This deprecated constant is here for compatibility only.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

Tablet Event Parameters

Define constants for parameters to table events.

```
enum {
    kEventParamTabletPointRec = 'tbrc',
    kEventParamTabletProximityRec = 'tbpx',
    typeTabletPointRec = 'tbrc',
    typeTabletProximityRec = 'tbpx',
    kEventParamTabletPointerRec = 'tbrc',
    typeTabletPointerRec = 'tbrc'
};
```

Constants

`kEventParamTabletPointRec`
typeTabletPointRec
Available in Mac OS X v10.1 and later.
Declared in `CarbonEvents.h`.

`kEventParamTabletProximityRec`
typeTabletProximityRec
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`typeTabletPointRec`
kEventParamTabletPointRec
Available in Mac OS X v10.1 and later.
Declared in `CarbonEvents.h`.

`typeTabletProximityRec`
kEventParamTabletProximityRec
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamTabletPointerRec`
typeTabletPointerRec
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`typeTabletPointerRec`
kEventParamTabletPointerRec
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

Text Input Events

Text Input Event Constants

Define constants related to events from `kEventClassTextInput`.

```
enum {
    kEventTextInputUpdateActiveInputArea = 1,
    kEventTextInputUnicodeForKeyEvent = 2,
    kEventTextInputOffsetToPos = 3,
    kEventTextInputPosToOffset = 4,
    kEventTextInputShowHideBottomWindow = 5,
    kEventTextInputGetSelectedText = 6,
    kEventTextInputUnicodeText = 7,
    kEventTextInputFilterText = 14
};
```

Constants

`kEventTextInputUpdateActiveInputArea`

Tells the application text engine to initiate or terminate or manage the content of inline input session.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventTextInputUnicodeForKeyEvent`

Provides Unicode text resulting from a key event (in which TSM originates the event) or from a `kEventTextInputUnicodeText` event produced by an input method, such as the Character Palette class input method, or a Handwriting input method. A client need not be fully TSM-aware to process or receive this event, which has become the standard way of getting Unicode text from key events. You can also get Mac encoding characters from the raw keyboard event contained in this event. If no `UnicodeForKeyEvent` handler is installed, and no `kUnicodeNotFromInputMethod` AppleEvent handler is installed (or the application has not created a Unicode TSMDocument), the Mac encoding charCodes (if these can be converted from the Unicodes) are provided to `WaitNextEvent`. This event is generated automatically by TSM when a `kEventRawKeyDown` event is sent to the application event target. The typical keyboard event flow begins with a `kEventRawKeyDown` event posted to the event queue. This event is dequeued during `WaitNextEvent` or `RunApplicationEventLoop`, and sent to the event dispatcher target. If the key down event reaches the application target, it is handled by TSM, which generates a `kEventTextInputUnicodeForKeyEvent` event and sends it to the event dispatcher target. The event dispatcher resends the event to the user focus target, which sends it to the focused window.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventTextInputOffsetToPos`

Convert from inline session text offset to global QD Point. This event is typically produced by an input method so that it can best position a palette “near” the text being operated on by the user.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventTextInputPosToOffset`

Convert from global QD point to inline session text offset. This event is typically produced by an input method to perform proper cursor management as the cursor moves over various subranges, or clauses of text (or the boundaries between these) in the inline input session.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventTextInputShowHideBottomWindow`
Show/Hide the bottom line input window. This event is produced by Input Methods to control the Text Services Manager bottom-line input window, and is not normally handled by an application.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventTextInputGetSelectedText`
Get the selected text (or the character before or after the insertion point, based on the value of the `leadingEdge` parameter) from the application's text engine.)

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventTextInputUnicodeText`
Produced only by input methods or other text services and is delivered to the Text Services Manager by `SendTextInputEvent`. The Text Services Manager does not dispatch this event to the user focus, so application handlers should not install handlers for this event. Instead, the Text Services Manager chains this event into any active keyboard input method in order to prevent interference with existing inline input sessions. The keyboard input method can either insert the text into the inline session or confirm its session and return the `UnicodeText` event to the Text Services Manager unhandled, in which case the Text Services Manager converts the event into a `UnicodeForKey` event (converting the Unicodes to Mac character codes and synthesizing information where needed) and finally dispatch the resulting event to the user focus as usual.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

`kEventTextInputFilterText`
Sent before any final-form text is sent to the user focus. Final form text includes text produced by a keyboard layout, Ink input method, Character palette or any other Text Services Manager text service, and any text being "confirmed" (or committed) from an inline input session. In the case of text confirmed from an inline input session, the Text Services Manager takes the resulting text buffer filtered by the event handler and adjusts all parameters in the `UpdateActiveInputArea` event produced by the input method. The text filtering action is thus transparent to both the application's `UpdateActiveInputArea` handler and the input method confirming the text.

Available in Mac OS X v10.4 and later.

Declared in `CarbonEvents.h`.

Discussion
The following text input events re-implement the Apple events defined in the *Text Services Manager Reference*, and provide the benefits of Carbon event targeting, dispatching and propagation to applications that have formerly handled the Text Services Manager suite of Apple events. You can install text input handlers on controls, windows, or the application event target (which is equivalent to Apple-event-based handling). In all cases, if a given text input handler is not installed, the Text Services Manager converts the event into an AppleEvent and re-dispatches it via `AESend` to the current process, making adoption as gradual as is desired.

Table 3-10 shows the parameters related to text input events.

Table 3-10 Required parameter names and types for text input event kinds

Event kind	Parameter name	Parameter type
`kEventTextInput-UpdateActiveInput-Area`	`kEventParamTextInputSendComponent-Instance`	`typeComponentInstance`

	kEventParamTextInputSendRefCon	typeLongInteger
	kEventParamTextInputSendSLRec	typeIntlWritingCode
	kEventParamTextInputSendFixLen	typeLongInteger
	kEventParamTextInputSendText	typeUnicodeText for a Unicode document; typeChar otherwise
kEventTextInput-UnicodeForKeyEvent	kEventParamTextInputSendComponent-Instance	typeComponentInstance
	kEventParamTextInputSendRefCon	typeLongInteger
	kEventParamTextInputSendSLRec	typeIntlWritingCode
	kEventParamTextInputSendText	typeUnicodeText
	kEventParamTextInputSendKeyboard Event	typeEventRef
kEventTextInput-OffsetToPos	kEventParamTextInputSendComponent Instance	typeComponentInstance
	kEventParamTextInputSendRefCon	typeLongInteger
	kEventParamTextInputSendTextOffset	typeLongInteger
	kEventParamTextInputReplyPoint	typeQDPoint
	kEventParamTextInputSendSLRec (Optional)	typeIntlWritingCode
	kEventParamTextInputSendLeadingEdge (Optional)	typeBoolean
	kEventParamTextInputSendReplySLRec (Optional)	typeIntlWritingCode
	kEventParamTextInputSendReplyFont (Optional)	typeLongInteger
	kEventParamTextInputSendReplyFMFont (Optional)	typeUInt32
	kEventParamTextInput-SendReplyPointSize (Optional)	typeFixed
	kEventParamTextInput-SendReplyLineHeight (Optional)	typeShortInteger
	kEventParamTextInput-SendReplyLineAscent (Optional)	typeShortInteger

	kEventParamTextInputSendReplyTextAngle (Optional)	typeFixed
kEventTextInput-PosToOffset	kEventParamTextInputSendComponentInstance	typeComponentInstance
	kEventParamTextInputSendRefCon	typeLongInteger
	kEventParamTextInputSendCurrentPoint	typeQDPoint
	kEventParamTextInputReplyRegionClass	typeLongInteger
	kEventParamTextInputReplyTextOffset	typeLongInteger
	kEventParamTextInputSendDraggingMode (Optional)	typeBoolean
	kEventParamTextInputReplyLeadingEdge (Optional)	typeBoolean
	kEventParamTextInputSendReplySLRec (Optional)	typeIntlWritingCode
kEventTextInputShow-HideBottomWindow	kEventParamTextInputSendComponent-Instance	typeComponentInstance
	kEventParamTextInputSendRefCon	typeLongInteger
	kEventParamTextInputSendShowHide (Optional)	typeBoolean
	kEventParamTextInputReplyShowHide (Optional)	typeBoolean
kEventTextInputGet-SelectedText	kEventParamTextInputSendComponent-Instance	typeComponentInstance
	kEventParamTextInputSendRefCon	typeLongInteger
	kEventParamTextInputSendLeadingEdge (Optional)	typeBoolean
	kEventParamTextInputSendTextService-Encoding (Optional)	TypeUInt32
	kEventParamTextInput-SendTextServiceMacEncoding (Optional)	TypeUInt32
	kEventParamTextInputReplyText (Optional)	typeUnicodeText or typeChar depending on the TSMDocument type.
	kEventParamTextInputSendReplySLRec (Optional)	typeIntlWritingCode

	kEventParamTextInputGlyphInfoArray (Optional)	TypeGlyphInfoArray
kEventTextInput-UnicodeText	kEventParamTextInputSendComponent-Instance	typeComponentInstance
	kEventParamTextInputSendSLRec	typeIntlWritingCode
	kEventParamTextInputSendText	typeUnicodeText
	kEventParamTextInputSendTextService-Encoding (Optional)	TypeUInt32
	kEventParamTextInputSendText ServiceMacEncoding (Optional)	TypeUInt32
	kEventParamTextInputGlyphInfo Array (Optional)	TypeGlyphInfoArray
kEventTextInput-FilterText	kEventParamTextInputSendRefCon	typeLongInteger
	kEventParamTextInputSendText	typeUnicodeText
	kEventParamTextInputReplyText	typeUnicodeText

Deprecated Text Input Constants

Define deprecated text input events.

```
enum {
    kEventUpdateActiveInputArea = kEventTextInputUpdateActiveInputArea,
    kEventUnicodeForKeyEvent = kEventTextInputUnicodeForKeyEvent,
    kEventOffsetToPos = kEventTextInputOffsetToPos,
    kEventPosToOffset = kEventTextInputPosToOffset,
    kEventShowHideBottomWindow = kEventTextInputShowHideBottomWindow,
    kEventGetSelectedText = kEventTextInputGetSelectedText
};
```

Constants

`kEventUpdateActiveInputArea`

Equivalent to `kEventTextInputUpdateActiveInputArea`.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventUnicodeForKeyEvent`

Equivalent to `kEventTextInputUnicodeForKeyEvent`.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventOffsetToPos`
Equivalent to `kEventTextInputOffsetToPos`.
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventPosToOffset`
Equivalent to `kEventTextInputPosToOffset`.
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventShowHideBottomWindow`
Equivalent to `kEventTextInputShowHideBottomWindow`.
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventGetSelectedText`
Equivalent to `kEventTextInputGetSelectedText`.
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

Discussion
These constants are superseded by constants described in "Text Input Event Constants" (page 403) and are included for backwards compatibility.

Text Input Event Parameters

Define constants for parameters to text input events.

```
enum {
    kEventParamTextInputSendRefCon = 'tsrc',
    kEventParamTextInputSendComponentInstance = 'tsci',
    kEventParamTextInputSendSLRec = 'tssl',
    kEventParamTextInputReplySLRec = 'trsl',
    kEventParamTextInputSendText = 'tstx',
    kEventParamTextInputReplyText = 'trtx',
    kEventParamTextInputSendUpdateRng = 'tsup',
    kEventParamTextInputSendHiliteRng = 'tshi',
    kEventParamTextInputSendClauseRng = 'tscl',
    kEventParamTextInputSendPinRng = 'tspn',
    kEventParamTextInputSendFixLen = 'tsfx',
    kEventParamTextInputSendLeadingEdge = 'tsle',
    kEventParamTextInputReplyLeadingEdge = 'trle',
    kEventParamTextInputSendTextOffset = 'tsto',
    kEventParamTextInputReplyTextOffset = 'trto',
    kEventParamTextInputReplyRegionClass = 'trrg',
    kEventParamTextInputSendCurrentPoint = 'tscp',
    kEventParamTextInputSendDraggingMode = 'tsdm',
    kEventParamTextInputReplyPoint = 'trpt',
    kEventParamTextInputReplyFont = 'trft',
    kEventParamTextInputReplyFMFont = 'trfm',
    kEventParamTextInputReplyPointSize = 'trpz',
    kEventParamTextInputReplyLineHeight = 'trlh',
    kEventParamTextInputReplyLineAscent = 'trla',
    kEventParamTextInputReplyTextAngle = 'trta',
    kEventParamTextInputSendShowHide = 'tssh',
    kEventParamTextInputReplyShowHide = 'trsh',
    kEventParamTextInputSendKeyboardEvent = 'tske',
    kEventParamTextInputSendTextServiceEncoding = 'tsse',
    kEventParamTextInputSendTextServiceMacEncoding = 'tssm',
    kEventParamTextInputGlyphInfoArray = 'glph',
    kEventParamTextInputSendGlyphInfoArray =  kEventParamTextInputGlyphInfoArray,
    kEventParamTextInputReplyGlyphInfoArray = 'rgph',
    kEventParamTextInputSendReplaceRange = 'tsrp'
};
```

Constants

`kEventParamTextInputSendRefCon`

typeLongInteger

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventParamTextInputSendComponentInstance`

typeComponentInstance

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventParamTextInputSendSLRec`

typeIntlWritingCode

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventParamTextInputReplySLRec`
typeIntlWritingCode
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamTextInputSendText`
typeUnicodeText (if TSMDocument is Unicode), otherwise typeChar
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamTextInputReplyText`
typeUnicodeText (if TSMDocument is Unicode), otherwise typeChar
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamTextInputSendUpdateRng`
typeTextRangeArray
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamTextInputSendHiliteRng`
typeTextRangeArray
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamTextInputSendClauseRng`
typeOffsetArray
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamTextInputSendPinRng`
typeTextRange
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamTextInputSendFixLen`
typeLongInteger
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamTextInputSendLeadingEdge`
typeBoolean
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamTextInputReplyLeadingEdge`
typeBoolean
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamTextInputSendTextOffset`
typeLongInteger
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamTextInputReplyTextOffset`
typeLongInteger
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamTextInputReplyRegionClass`
typeLongInteger
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamTextInputSendCurrentPoint`
typeQDPoint
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamTextInputSendDraggingMode`
typeBoolean
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamTextInputReplyPoint`
typeQDPoint
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamTextInputReplyFont`
typeLongInteger
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamTextInputReplyFMFont`
typeUInt32
Available in Mac OS X v10.1 and later.
Declared in `CarbonEvents.h`.

`kEventParamTextInputReplyPointSize`
typeFixed
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamTextInputReplyLineHeight`
typeShortInteger
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamTextInputReplyLineAscent`
typeShortInteger
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamTextInputReplyTextAngle`
typeFixed
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamTextInputSendShowHide`
typeBoolean
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamTextInputReplyShowHide`
typeBoolean
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamTextInputSendKeyboardEvent`
typeEventRef
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamTextInputSendTextServiceEncoding`
typeUInt32
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamTextInputSendTextServiceMacEncoding`
typeUInt32
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamTextInputReplyGlyphInfoArray`
typeGlyphInfoArray
Available in Mac OS X v10.3 and later.
Declared in `CarbonEvents.h`.

`kEventParamTextInputSendReplaceRange`
typeCFRange
Available in Mac OS X v10.3 and later.
Declared in `CarbonEvents.h`.

Text Service Manager Document Events

Text Service Manager Document Event Parameters

Define constants for Text Service Manager Document event parameters and types.

```
enum {
    kEventParamTSMDocAccessSendRefCon = kEventParamTSMSendRefCon,
    kEventParamTSMDocAccessSendComponentInstance =
kEventParamTSMSendComponentInstance,
    kEventParamTSMDocAccessCharacterCount = 'tdct',
    kEventParamTSMDocAccessReplyCharacterRange = 'tdrr',
    kEventParamTSMDocAccessReplyCharactersPtr = 'tdrp',
    kEventParamTSMDocAccessSendCharacterIndex = 'tdsi',
    kEventParamTSMDocAccessSendCharacterRange = 'tdsr',
    kEventParamTSMDocAccessSendCharactersPtr = 'tdsp',
    kEventParamTSMDocAccessRequestedCharacterAttributes = 'tdca',
    kEventParamTSMDocAccessReplyATSFont = 'tdaf',
    kEventParamTSMDocAccessReplyFontSize = 'tdrs',
    kEventParamTSMDocAccessEffectiveRange = 'tder',
    kEventParamTSMDocAccessReplyATSUGlyphSelector = 'tdrg',
    kEventParamTSMDocAccessLockCount = 'tdlc',
    kEventParamTSMDocAccessLineBounds = 'tdlb',
    typeATSFontRef = 'atsf',
    typeGlyphSelector = 'glfs'
};
```

Constants

`kEventParamTSMDocAccessSendRefCon`

typeLongInteger

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`kEventParamTSMDocAccessSendComponentInstance`

typeComponentInstance

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`kEventParamTSMDocAccessCharacterCount`

typeCFIndex

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`kEventParamTSMDocAccessReplyCharacterRange`

typeCFRange

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`kEventParamTSMDocAccessReplyCharactersPtr`

typePtr

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`kEventParamTSMDocAccessSendCharacterIndex`

typeCFIndex

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`kEventParamTSMDocAccessSendCharactersPtr`
typePtr
Available in Mac OS X v10.3 and later.
Declared in `CarbonEvents.h`.

`kEventParamTSMDocAccessRequestedCharacterAttributes`
typeUInt32
Available in Mac OS X v10.3 and later.
Declared in `CarbonEvents.h`.

`kEventParamTSMDocAccessReplyATSFont`
typeATSFontRef
Available in Mac OS X v10.3 and later.
Declared in `CarbonEvents.h`.

`kEventParamTSMDocAccessReplyFontSize`
typeFloat
Available in Mac OS X v10.3 and later.
Declared in `CarbonEvents.h`.

`kEventParamTSMDocAccessEffectiveRange`
typeRange
Available in Mac OS X v10.3 and later.
Declared in `CarbonEvents.h`.

`kEventParamTSMDocAccessReplyATSUGlyphSelector`
typeGlyphSelector
Available in Mac OS X v10.3 and later.
Declared in `CarbonEvents.h`.

`kEventParamTSMDocAccessLockCount`
typeCFIndex
Available in Mac OS X v10.3 and later.
Declared in `CarbonEvents.h`.

`kEventParamTSMDocAccessLineBounds`
typeCFMutableArrayRef
Available in Mac OS X v10.4 and later.
Declared in `CarbonEvents.h`.

`typeATSFontRef`
ATSFontRef
Available in Mac OS X v10.3 and later.
Declared in `CarbonEvents.h`.

`typeGlyphSelector`
ATSUGlyphSelector
Available in Mac OS X v10.3 and later.
Declared in `CarbonEvents.h`.

Timer Constants

Idle Timer Event Constants

Define constants used for timer events.

```
enum {
    kEventLoopIdleTimerStarted = 1,
    kEventLoopIdleTimerIdling = 2,
    kEventLoopIdleTimerStopped = 3
};
```

Constants

`kEventLoopIdleTimerStarted`

The idle period has just begun (and this is the first time your callback is being called for this idle period).

Available in Mac OS X v10.2 and later.

Declared in `CarbonEventsCore.h`.

`kEventLoopIdleTimerIdling`

The idle period is continuing.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEventsCore.h`.

`kEventLoopIdleTimerStopped`

The idle period has just stopped (a user event occurred). Your callback should do any necessary cleanup of the idle process now that a user event has occurred.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEventsCore.h`.

Discussion

These constants are passed to your idle timer callback function. For more information, see the `InstallEventLoopIdleTimer` (page 275) function.

Toolbar Events

Toolbar Event Parameters

Define constants for parameters to toolbar events.

```
enum {
    kEventParamToolbar = 'tbar',
    kEventParamToolbarItem = 'tbit',
    kEventParamToolbarItemIdentifier = 'tbii',
    kEventParamToolbarItemConfigData = 'tbid',
    typeHIToolbarRef = 'tbar',
    typeHIToolbarItemRef = 'tbit'
};
```

Constants

`kEventParamToolbar`
typeHIToolbarRef
Available in Mac OS X v10.2 and later.
Declared in `HIToolbar.h`.

`kEventParamToolbarItem`
typeHIToolbarItemRef
Available in Mac OS X v10.2 and later.
Declared in `HIToolbar.h`.

`kEventParamToolbarItemIdentifier`
typeHIToolbarRef
Available in Mac OS X v10.2 and later.
Declared in `HIToolbar.h`.

`kEventParamToolbarItemConfigData`
typeCFStringRef
Available in Mac OS X v10.2 and later.
Declared in `HIToolbar.h`.

Discussion

For details about toolbar events and event parameters, see *HIToolbar Reference*in the User Experience section of the Carbon documentation.

Volume Events

Volume Event Constants

Define constants related to events from `kEventClassVolume`.

```
enum {
    kEventVolumeMounted = 1,
    kEventVolumeUnmounted = 2
};
```

Constants

`kEventVolumeMounted`
New volume (hard drive or removable media) mounted.
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventVolumeUnmounted`
Volume has been ejected or unmounted.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

Volume Reference Constant

Define the type of a volume reference.

```
enum {
    typeFSVolumeRefNum = 'voln'
};
```

Constants

`typeFSVolumeRefNum`
An `FSVolumeRefNum` identifying the volume that was mounted or unmounted.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

Window Events

Window Action Event Constants

Define constants related to events from `kEventClassWindow`.

```
enum {
    kEventWindowCollapse = 66,
    kEventWindowCollapseAll = 68,
    kEventWindowExpand = 69,
    kEventWindowExpandAll = 71,
    kEventWindowClose = 72,
    kEventWindowCloseAll = 74,
    kEventWindowZoom = 75,
    kEventWindowZoomAll = 77,
    kEventWindowContextualMenuSelect = 78,
    kEventWindowPathSelect = 79,
    kEventWindowGetIdealSize = 80,
    kEventWindowGetMinimumSize = 81,
    kEventWindowGetMaximumSize = 82,
    kEventWindowConstrain = 83,
    kEventWindowHandleContentClick = 85,
    kEventWindowTransitionStarted = 88,
    kEventWindowTransitionCompleted = 89,
    kEventWindowGetDockTileMenu = 90,
    kEventWindowGetDockTileMenu = 90,
    kEventWindowProxyBeginDrag = 128,
    kEventWindowProxyEndDrag = 129,
    kEventWindowToolbarSwitchMode = 150
};
```

Constants

`kEventWindowCollapse`

If the window is not collapsed, this event is sent by the standard window handler after it has received `kEventWindowClickCollapseRgn` and received `true` from a call to `TrackBox`. The default behavior is to call `CollapseWindow` and then send `kEventWindowCollapsed` if no error is received from `CollapseWindow`.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowCollapseAll`

Sent by the standard window handler (when the option key is down) after it has received `kEventWindowClickCollapseRgn` and then received true from a call to `TrackBox`. The default response is to send each window of the same class as the clicked window a `kEventWindowCollapse` event.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowExpand`

If the window is collapsed, this event is sent by the standard window handler after it has received `kEventWindowClickCollapseRgn` and received true from a call to `TrackBox`. The default response is to call `CollapseWindow`, then send `kEventWindowExpanded`. Note that you will not receive this event before a window is expanded from the dock, since minimized windows in the dock don't uses collapse boxes to unminimize.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowExpandAll`
Sent by the standard window handler (when the option key is down) after it has received `kEventWindowClickCollapseRgn` and then received true from a call to `TrackBox`. The default response is to send each window of the same class as the clicked window a `kEventWindowExpand` event.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowClose`
Sent by the standard window handler after it has received `kEventWindowClickCloseRgn` and successfully called `TrackBox`. Your application might intercept this event to check if the document is dirty, and display a Save/Don't Save/Cancel alert.

The default response is to call the Window Manager function `DisposeWindow`.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowCloseAll`
Sent by the standard window handler (when the option key is down) after it has received `kEventWindowClickCloseRgn` and received true from a call to `TrackGoAway`. The standard window handler's response is to send each window with the same class as the clicked window a `kEventWindowClose` event.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowZoom`
Sent by the standard window handler upon receiving `kEventWindowClickZoomRgn` and then receiving true from a call to `TrackBox`. The default behavior is to zoom the window using `ZoomWindowIdeal` then, if successful, send a `kEventWindowZoomed` event.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowZoomAll`
Sent by the standard window handler (when the option key is down) after it has received `kEventObjectClickZoomRgn` and received true from a call to `TrackBox`. The standard window handler's response is to send each window with the same class as the clicked window a `kEventObjectZoom` event and then to reposition all zoomed windows using the `kWindowCascadeOnParentWindowScreen` positioning method. For more details, see the *Window Manager Reference* for more details.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowContextualMenuSelect`
Sent when either the right mouse button is pressed, or the control key is held down and the left mouse button is pressed, or the left mouse button is held down for more than 1/4th of a second (and nothing else is handling the generated mouse tracking events). The standard window handler ignores this event. Note that this event supports `kEventParamMouseLocation` and other parameters associated with the `kEventMouseDown` event.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowPathSelect`

Sent when the Window Manager function `IsWindowPathSelectClick` would return true. The standard window handler sends this event while handling `kEventWindowClickDragRgn` if the click occurs in the proxy icon. Set the menu reference parameter (`kEventParamMenuRef`) in the event if you wish to customize the menu passed to the Window Manager function `WindowPathSelect`.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowGetIdealSize`

Sent by the standard window handler to determine the standard state for zooming. The standard window handler ignores this event.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowGetMinimumSize`

Sent by the standard window handler to determine the minimum size of the window (used during window resizing).

In Mac OS X v10.2 and later, the default behavior is to call the Window Manager function `GetWindowResizeLimits` and return the size obtained in the `kEventParamDimensions` parameter. There is no default behavior before Mac OS X v10.2.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowGetMaximumSize`

Sent by the standard window handler to determine the maximum size of the window (used during window resizing). In Mac OS X v10.2 and later, this event is also sent by the Window Manager functions `ResizeWindow` and `GrowWindow` if the `sizeContraints` parameter was set to `NULL`.

In Mac OS X v10.2 and later, the default behavior is to call the Window Manager function `GetWindowResizeLimits` and return the size obtained in the `kEventParamDimensions` parameter. There is no default behavior before Mac OS X v10.2.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowConstrain`

Sent by the standard window handler to warn of a change in the available window positioning bounds on the window (for example, due to a change in screen resolution or Dock size).

In Mac OS v10.0 the default behavior is to call the Window Manager function `ConstrainWindowToScreen` on the window with the `kWindowConstrainMoveRegardlessOfFit` attribute set and a window region code of `kWindowDragRgn`. The window is constrained to the bounds returned by the Window Manager function `GetAvailableWindowPositioningBounds` for that display device.

In Mac OS X v10.1 and later the default behavior is to call `ConstrainWindowToScreen` on the window with the `kWindowConstrainMoveRegardlessOfFit` and `kWindowConstrainAllowPartial` attributes set, and a window region code of `kWindowDragRgn`. Instead of accepting the normal device bounds, you can also modify the `kEventParamAvailableBounds` for this event, and the default handler constrains the window to those bounds.

In Mac OS X v10.2 and later, you can set the following optional parameters:

- `kEventParamAttributes`: You can set the constraint attributes to pass to `ConstrainWindowToScreen` by specifying them in this parameter.
- `kEventParamWindowRegionCode`: If you set this parameter (which must be of type `WindowRegionCode`), the standard window handler passes this value to `ConstrainWindowToScreen` instead of `kWindowDragRgn`.

In addition, the following optional parameters may exist in Mac OS X v10.2 and later:

- `kEventParamRgnHandle`: Contains the gray region before a configuration change in the available graphics devices (screens). This parameter exists only if the constrain event occurred because the user changed the screen configuration. You can call the Window Manager function `GetGrayRgn` to obtain the current gray region.
- `kEventParamCurrentDockRect`: Holds the current bounds of the dock. This parameter and `kEventParamPreviousDockRect` exist only if the constrain event resulted from a change in the Dock size or position.
- `kEventParamPreviousDockRect`: Holds the previous bounds of the dock.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowHandleContentClick`

Sent by the standard window handler in response to `kEventWindowClickContentRgn` when a mouse click is in the content region but is not a contextual menu invocation or a click on a control. Note that this event supports `kEventParamMouseLocation` and other parameters associated with the `kEventMouseDown` event.

The standard handler ignores this event.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CarbonEvents.h`.

`kEventWindowTransitionStarted`

Sent to all handlers registered for it. It is sent by the `TransitionWindow`, `TransitionWindowAndParent`, and `TransitionWindowWithOptions` APIs just before the first frame of the transition animation.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`kEventWindowTransitionCompleted`
Sent to all handlers registered for it. It is sent by the `TransitionWindow`, `TransitionWindowAndParent`, and `TransitionWindowWithOptions` APIs just after the last frame of the transition animation.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`kEventWindowGetDockTileMenu`
Sent when a dock tile wants to display a menu. The sender of this event releases the menu after the Dock has displayed it, so if you want to keep the menu, you must call `RetainMenu` on it before returning from the event handler.

If you do not handle this event, the default behavior is to call the Window Manager function `GetWindowDockTileMenu` and return the menu obtained in the `kEventParamMenuRef` parameter. If no menu is specified, the default handler returns `eventNotHandledErr`.

Note that in most cases it is simpler to call the `SetWindowDockTileMenu` function directly rather than register for this event.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

`kEventWindowProxyBeginDrag`
Sent before a proxy icon drag; you can attach data to the `DragRef` in the event. The standard handler ignores this event.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowProxyEndDrag`
Sent after the proxy icon drag is complete, whether successful or not. The standard handler ignores this event.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowToolbarSwitchMode`
The toolbar button (that is, the oblong button used to show and hide the toolbar) was successfully clicked. The standard window handler sends this event when receiving a `true` return value from `TrackBox` during the handling of the `kEventWindowClickToobarButtonRgn` event. Note, however, that you do not have to have the standard window handler installed to receive this event; any window that has a toolbar receives this event when its toolbar button is successfully clicked. Note that if you handle this event, your application is responsible for keeping track of the toolbar's mode (visible or hidden).

The default response is to toggle the toolbar (that is, show it if it is hidden, and vice-versa) when the toolbar button is clicked. If the option key is held down during the click, all toolbars in the windows of the current process are toggled. If the command key is down during the click, the toolbar mode is cycled between icons and text, icons alone, and text alone. If both the command and option keys are held during the click, the system displays the toolbar configuration sheet.

Available in Mac OS X v10.1 and later.

Declared in `CarbonEvents.h`.

Discussion
These events indicate that certain changes have been made to a window. These events have greater semantic meaning than the low-level window click events and are usually preferred for overriding.

Table 3-11 shows the parameters related to window action events.

Table 3-11 Parameter names and types for window action event kinds

Event kind	Parameter name	Parameter type
kEventWindowCollapse	kEventParamDirectObject	typeWindowRef
kEventWindowCollapsed	kEventParamDirectObject	typeWindowRef
kEventWindowCollapseAll	kEventParamDirectObject	typeWindowRef
kEventWindowExpand	kEventParamDirectObject	typeWindowRef
kEventWindowExpanded	kEventParamDirectObject	typeWindowRef
kEventWindowExpandAll	kEventParamDirectObject	typeWindowRef
kEventWindowClose	kEventParamDirectObject	typeWindowRef
kEventWindowClosed	kEventParamDirectObject	typeWindowRef
kEventWindowCloseAll	kEventParamDirectObject	typeWindowRef
kEventWindowZoom	kEventParamDirectObject	typeWindowRef
kEventWindowZoomed	kEventParamDirectObject	typeWindowRef
kEventWindowZoomAll	kEventParamDirectObject	typeWindowRef
kEventWindow-ContextualMenuSelect	kEventParamDirectObject	typeWindowRef
	Other parameters from kEventMouseDown	
kEventWindowPathSelect	kEventParamDirectObject	typeWindowRef
	kEventParamMenuRef	typeMenuRef
kEventWindowGetIdealSize	kEventParamDirectObject	typeWindowRef
	kEventParamDimensions	typeQDPoint
kEventWindowGetMinimumSize	kEventParamDirectObject	typeWindowRef
	kEventParamDimensions	typeQDPoint
kEventWindowGetMaximumSize	kEventParamDirectObject	typeWindowRef
	kEventParamDimensions	typeQDPoint
kEventWindowConstrain	kEventParamDirectObject	typeWindowRef
	kEventParamAvailableBounds	typeQDRectangle
	kEventParamAttributes	typeUInt32
	kEventParamWindowRegionCode	typeQDRgnHandle

	kEventParamPreviousDockRect	typeHIRect
	kEventParamCurrentDockRect	typeHIRect
kEventWindowHandle- WindowContent Click	kEventParamDirectObject	typeWindowRef
	Other parameters from kEventMouseDown	
kEventWindowProxyBeginDrag	kEventParamDirectObject	typeWindowRef
kEventWindowProxyEndDrag	kEventParamDirectObject	typeWindowRef
kEventWindowToolbarSwitchMode	kEventParamDirectObject	typeWindowRef

Window Activation Event Constants

Define constants related to events from kEventClassWindow that specify whether a window is activated or deactivated.

```
enum {
    kEventWindowActivated = 5,
    kEventWindowDeactivated = 6,
    kEventWindowHandleActivate = 91,
    kEventWindowHandleDeactivate = 92
    kEventWindowGetClickActivation = 7,
    kEventWindowGetClickModality = 8
};
```

Constants

kEventWindowActivated

The window is active now. Sent to any window that is activated, regardless of whether the window has the standard window handler installed.

In Mac OS X v10.3 and later, the standard window handler responds to this event by sending a kEventWindowHandleActivate event to the window. On earlier releases of Mac OS X and CarbonLib, the standard window handler calls ActivateControl on the window's root control.

Note that this event is sent directly to the window target. If no handler takes the event (that is, they all return eventNotHandledErr), then the Window Manager posts this event to the event queue. Doing so allows the Carbon Event Manager to convert the event into an old-style event record (EventRecord), to be returned from WaitNextEvent.

Available in Mac OS X v10.0 and later.

Declared in CarbonEvents.h.

`kEventWindowDeactivated`

The window is inactive now. Sent to any window that is deactivated, regardless of whether the window has the standard window handler installed.

In Mac OS X v10.3 and later, the standard window handler responds to this event by sending a `kEventWindowHandleDeactivate` event to the window. On earlier releases of Mac OS X and CarbonLib, the standard window handler calls `DeactivateControl` on the window's root control.

Note that this event is sent directly to the window target. If no handler takes the event (that is, they all return `eventNotHandledErr`), then the Window Manager posts this event to the event queue. Doing so allows the Carbon Event Manager to convert the event into an old-style event record (`EventRecord`), to be returned from `WaitNextEvent`.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowHandleActivate`

The window has received a `kEventWindowActivated` event, and its contents should become active. This event is generated by the standard window handler in response to a `kEventWindowActivated` event. You can handle this event by activating the window's content appropriately. The standard window handler responds to this event by calling `ActivateControl` on the window's root control.

Available in Mac OS X v10.4 and later.

Declared in `CarbonEvents.h`.

`kEventWindowHandleDeactivate`

The window has received a `kEventWindowDeactivated` event, and its contents should become inactive. This event is generated by the standard window handler in response to a `kEventWindowDeactivated` event. You can handle this event by deactivating the window's content appropriately. The standard window handler responds to this event by calling `DeactivateControl` on the window's root control.

Available in Mac OS X v10.4 and later.

Declared in `CarbonEvents.h`.

`kEventWindowGetClickActivation`

Sent when a click occurs in a background window that has the standard window handler installed.

The default behavior is to bring the window forward and absorb the click (that is, the mouse click is not passed on to the window). In addition, the appropriate click activation part code is returned in the `kEventParamClickActivationResult` parameter. You can use this event to override this behavior and implement click-through.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowGetClickModality`

Sent to a window by the event dispatcher target before dispatching a mouse-down or mouse-up event to the clicked window. A handler for this event may examine application state to determine whether this click should be allowed. This event may also be sent in circumstances other than a mouse event:

- In `SelectWindow`
- When handling the cmd-~ key sequence
- When restoring a collapsed window from the Dock
- When handling the `kHICommandSelectWindow` command
- When activating a clicked window during application activation,

In each case, the result of this event is used to determine whether z-ordering, activation, and highlighting of the window should be allowed.

This event contains an optional EventRef parameter that is the original mouse event. If the parameter is not present, the handler should generally assume that the click was a single click.

A default handler for this event is installed on the application target. The default handler determines whether this is a modal click by examining the modality of the visible, uncollapsed windows in front of the clicked window, the location of the click, and the keyboard modifiers. A custom handler may, of course, entirely ignore window z-order or modality, and determine modality in any way it deems necessary.

Available in Mac OS X v10.4 and later.

Declared in `CarbonEvents.h`.

Discussion
Events related to activating and deactivating a window.

Table 3-12 shows the parameters related to window activation events.

Table 3-12 Parameter names and types for window activation event kinds

Event kind	Parameter name	Parameter type
`kEventWindow-Activated`	`kEventParamDirectObject`	`typeWindowRef`
`kEventWindow-Deactivated`	`kEventParamDirectObject`	`typeWindowRef`
`kEventWindowHandle-Activate`	`kEventParamDirectObject`	`typeWindowRef`
`kEventWindowHandle-Deactivate`	`kEventParamDirectObject`	`typeWindowRef`
`kEventWindowGetClick-Activation`	`kEventParamDirectObject`	`typeWindowRef`
	`kEventParamMouseLocation`	`typeQDPoint`
	`kEventParamKeyModifiers`	`typeUInt32`
	`kEventParamWindowDefPart`	`typeWindowDefPartCode`

	kEventParamControlRef	typeControlRef
	kEventParamClickActivation	typeClickActivation-Result
kEventWindowGet-ClickModality	kEventParamDirectObject	typeWindowRef
	kEventParamWindowPartCode	typeWindowPartCode
	kEventParamKeyModifiers	typeUInt32
	kEventParamEventRef (Optional)	typeEventRef
	kEventParamModalClickResult	typeModalClickResult
	kEventParamModalWindow (Required only if kEventParamModalClickResult is kHIModalClickIsModal.)	typeWindowRef
	kEventParamWindowModality (Required only if kEventParamModalClickResult is kHIModalClickIsModal.)	typeWindowModality

Window Click Event Constants

Define constants related to events from `kEventClassWindow` occurring in the standard window controls or regions (close, resize, drag, and so on).

```
enum {
    kEventWindowClickDragRgn = 32,
    kEventWindowClickResizeRgn = 33,
    kEventWindowClickCollapseRgn = 34,
    kEventWindowClickCloseRgn = 35,
    kEventWindowClickZoomRgn = 36,
    kEventWindowClickContentRgn = 37,
    kEventWindowClickProxyIconRgn = 38,
    kEventWindowClickToolbarButtonRgn = 41,
    kEventWindowClickStructureRgn = 42
};
```

Constants

`kEventWindowClickDragRgn`

Sent when the mouse is down in the drag region. The standard window handler calls `DragWindow`.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CarbonEvents.h`.

`kEventWindowClickResizeRgn`
Sent when the mouse is down in the resize area. The standard window handler calls `ResizeWindow`.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CarbonEvents.h`.

`kEventWindowClickCollapseRgn`
Sent when the mouse is down in the collapse widget. The default behavior is to call `CollapseWindow`, and then generate `kEventWindowExpand` or `kEventWindowCollapse` (whichever is the opposite of the window's original collapse state).

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CarbonEvents.h`.

`kEventWindowClickCloseRgn`
Sent when the mouse is down in the close widget. The standard window handler calls `TrackGoAway`, and then generates `kEventWindowClose`.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CarbonEvents.h`.

`kEventWindowClickZoomRgn`
Sent when the mouse is down in the zoom widget. The standard window handler calls `TrackBox`, and then generates `kEventWindowZoom`.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CarbonEvents.h`.

`kEventWindowClickContentRgn`
Sent when the mouse is down in the content region. The standard window handler checks for contextual menu clicks and clicks on controls, and sends `kEventWindowContextualMenuSelect`, `kEventControlClick`, and `kEventWindowHandleContentClick` events as appropriate.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CarbonEvents.h`.

`kEventWindowClickProxyIconRgn`
Sent when the mouse is down in the proxy icon. The standard window handler handles proxy icon dragging, and generates proxy icon events.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CarbonEvents.h`.

`kEventWindowClickToolbarButtonRgn`
Sent when the mouse is down in the toolbar button. The default behavior is to call `TrackBox` and then generate a `kEventWindowToolbarSwitchMode` event.

Available in Mac OS X v10.1 and later.

Not available to 64-bit applications.

Declared in `CarbonEvents.h`.

Discussion
Low-level events which generate higher-level "action" events described in "Window Action Event Constants" (page 418). These events are generated only for windows with the standard window handler installed. Most clients should allow the standard window handler to implement these events.

Window State Event Constants

Define constants related to events from `kEventClassWindow` that notify of a change in the window's state.

```
enum {
    kEventWindowShowing = 22,
    kEventWindowHiding = 23,
    kEventWindowShown = 24,
    kEventWindowHidden = 25,
    kEventWindowCollapsing = 86,
    kEventWindowCollapsed = 67,
    kEventWindowExpanding = 87,
    kEventWindowExpanded = 70,
    kEventWindowZoomed = 76,
    kEventWindowBoundsChanging = 26,
    kEventWindowBoundsChanged = 27,
    kEventWindowResizeStarted = 28,
    kEventWindowResizeCompleted = 29,
    kEventWindowDragStarted = 30,
    kEventWindowDragCompleted = 31,
    kEventWindowClosed = 73
};
```

Constants

`kEventWindowShowing`

A window is being shown. This is sent inside `ShowHide`. This event is propagated to all handlers that registered for the event in the event target's handler chain, regardless of return value.

The standard window handler ignores this event.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowHiding`

A window is being hidden. This is sent inside `ShowHide`. This event is propagated to all handlers that registered for the event in the event target's handler chain, regardless of return value.

The standard window handler ignores this event.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowShown`

The window has been shown. This event is propagated to all handlers that registered for the event in the event target's handler chain, regardless of return value.

The standard window handler ignores this event.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowHidden`
The window has been hidden. This event is propagated to all handlers that registered for the event in the event target's handler chain, regardless of return value.

The standard window handler ignores this event.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowCollapsing`
The window is collapsing. This event is propagated to all handlers that registered for the event in the event target's handler chain, regardless of return value.

The standard window handler ignores this event.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

`kEventWindowCollapsed`
The object has successfully collapsed. This event is propagated to all handlers that registered for the event in the event target's handler chain, regardless of return value.

The standard window handler ignores this event.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowExpanding`
The window is expanding. This event is propagated to all handlers that registered for the event in the event target's handler chain, regardless of return value.

The standard window handler ignores this event.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

`kEventWindowExpanded`
The window has successfully expanded. This event is propagated to all handlers that registered for the event in the event target's handler chain, regardless of return value.

The standard window handler ignores this event.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowZoomed`
The window has been successfully zoomed. This event is propagated to all handlers that registered for the event in the event target's handler chain, regardless of return value.

In CarbonLib 1.1 through 1.4 and Mac OS X prior to v10.2, this event is sent only by the standard window handler after handling `kEventWindowZoom`. In CarbonLib 1.5 and later and Mac OS X v10.2 and later, this event is sent by the Window Manager functions `ZoomWindow` and `ZoomWindowIdeal`.

The standard window handler ignores this event.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowBoundsChanging`

Sent during `DragWindow` or `ResizeWindow`, before the window is actually moved or resized. You can alter the current bounds in the event (the `kEventParamCurrentBounds` parameter) to change the eventual size and location of the window. Do not call the Window Manager functions `SizeWindow` or `SetWindowBounds` from inside a handler for this event.

In Mac OS X v10.1 and later, this event is sent before all changes to a window's bounds, whether initiated by a user or by a Window Manager call. If the event was sent in response to a user action, the `kWindowBoundsChangeUserDrag` or `kWindowBoundsChangeUserResize` attribute will be set in the `kEventParamAttributes` parameter.

The standard window handler ignores this event.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowBoundsChanged`

The window has been moved or resized (or both). Do not call the Window Manager functions `SizeWindow` or `SetWindowBounds` from inside a handler for this event. If you want to enforce certain window bounds, you should do so from a `kEventWindowBoundsChanging` event handler.

This event is propagated to all handlers that registered for the event in the event target's handler chain, regardless of return value.

In Mac OS X v10.2 and later, the standard window handler can take this event under the following conditions:

- the window uses live resizing (the `kWindowLiveResizeAttribute` attribute is set.
- the user is the one resizing the window
- an update event for the window exists in the event queue

If these conditions are met, the standard window handler removes the update event from the event queue and sends it to the event dispatcher target. Doing so simplifies redrawing window content during live resizing.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowResizeStarted`

The user has just started to resize a window. This event is propagated to all handlers that registered for the event in the event target's handler chain, regardless of return value.

The standard window handler ignores this event.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowResizeCompleted`

The user has just finished resizing a window. This event is propagated to all handlers that registered for the event in the event target's handler chain, regardless of return value.

The standard window handler ignores this event.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowDragStarted`

The user has just started to drag a window. This event is propagated to all handlers that registered for the event in the event target's handler chain, regardless of return value.

The standard window handler ignores this event.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowDragCompleted`

The user has completed a window drag. This event is propagated to all handlers that registered for the event in the event target's handler chain, regardless of return value.

The standard window handler ignores this event.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowClosed`

Dispatched by `DisposeWindow` before the window is disposed. This event is propagated to all handlers that registered for the event in the event target's handler chain, regardless of return value.

In CarbonLib 1.5 and earlier and Mac OS X prior to v10.2, if a visible window is destroyed, `kEventWindowClosed` is sent before the `kEventWindowHidden` event. In CarbonLib 1.6 and Mac OS X v10.2 and later, the `kEventWindowClosed` event is sent after `kEventWindowHidden`.

The standard window handler ignores this event.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

Discussion

Table 3-13 shows the parameters related to window state change events.

Table 3-13 Parameter names and types for window state change event kinds

Event kind	Parameter name	Parameter type
`kEventWindowBoundsChanging`	`kEventParamDirectObject`	`typeWindowRef`
	`kEventParamAttributes`	`typeUInt32`
	`kEventParamOriginalBounds`	`typeQDRectangle`
	`kEventParamPreviousBounds`	`typeQDRectangle`
	`kEventParamCurrentBounds`	`typeQDRectangle`
`kEventWindowBoundsChanged`	`kEventParamDirectObject`	`typeWindowRef`
	`kEventParamAttributes`	`typeUInt32`
	`kEventParamOriginalBounds`	`typeQDRectangle`
	`kEventParamPreviousBounds`	`typeQDRectangle`
	`kEventParamCurrentBounds`	`typeQDRectangle`
`kEventWindowShown`	`kEventParamDirectObject`	`typeWindowRef`

kEventWindowShowing	kEventParamDirectObject	typeWindowRef
kEventWindowHidden	kEventParamDirectObject	typeWindowRef
kEventWindowHiding	kEventParamDirectObject	typeWindowRef
kEventWindowResizeStarted	kEventParamDirectObject	typeWindowRef
kEventWindowResizeCompleted	kEventParamDirectObject	typeWindowRef
kEventWindowDragStarted	kEventParamDirectObject	typeWindowRef
kEventWindowDragCompleted	kEventParamDirectObject	typeWindowRef

Window Refresh Event Constants

Define constants related to window refresh events from `kEventClassWindow`.

```
enum {
    kEventWindowUpdate = 1,
    kEventWindowDrawContent = 2
};
```

Constants

`kEventWindowUpdate`

Low-level update event. Sent to any window that needs updating regardless of whether the window has the standard window handler installed. You must call the Window Manager function `BeginUpdate`, and the QuickDraw function `SetPort` before drawing your window content, then call `EndUpdate` when you are finished.

The standard window handler for this event calls `BeginUpdate` and `SetPort`, sends a `kEventWindowDrawContent` event to the window, and then calls `EndUpdate`.

Note that this event is sent directly to the window target. If no handler takes the event (that is, they all return `eventNotHandledErr`), then the Window Manager posts this event to the event queue. Doing so allows the Carbon Event Manager to convert the event into an old-style event record (`EventRecord`), to be returned from `WaitNextEvent`.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CarbonEvents.h`.

`kEventWindowDrawContent`

Higher-level update event sent only if you have the standard window handler installed. Functions exactly as `kEventWindowUpdate`, except that the Carbon Event Manager calls `Begin/EndUpdate` and `SetPort` for you. All you need to do is draw the window content.

The standard window handler calls `DrawControls` for this window.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `CarbonEvents.h`.

Discussion

Events related to drawing a window's content.

Table 3-14 shows the parameters related to window refresh events.

Table 3-14 Parameter names and types for window refresh event kinds

Event kind	Parameter name	Parameter type
`kEventWindowUpdate`	`kEventParamDirectObject`	`typeWindowRef`
`kEventWindowDrawContent`	`kEventParamDirectObject`	`typeWindowRef`

Window Cursor Change Event Constant

Define a constant related to events from `kEventClassWindow` that specify that the cursor must change.

```
enum {
    kEventWindowCursorChange = 40
};
```

Constants

`kEventWindowCursorChange`

Sent when the mouse is moving over the content region. This event is used to manage ownership of the cursor. You should only change the cursor if you receive this event; otherwise, someone else needed to adjust the cursor and handled the event (for example, a TSM Input Method when the mouse is over an inline input region).

The standard handler ignores this event.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

Discussion

Table 3-15 shows the parameters related to window cursor change events.

Table 3-15 Parameter names and types for window cursor change event kinds

Event kind	Parameter name	Parameter type
`kEventWindowCursorChange`	`kEventParamDirectObject`	`typeWindowRef`
	`kEventParamMouseLocation`	`typeQDPoint`
	`kEventParamKeyModifiers`	`typeUInt32`

Window Focus Event Constants

Define constants related to events from `kEventClassWindow` that describe changes in the user focus.

```
enum {
    kEventWindowFocusAcquired = 200,
    kEventWindowFocusRelinquish = 201,
    kEventWindowFocusContent = 202,
    kEventWindowFocusToolbar = 203,
    kEventWindowFocusDrawer = 204
};
```

Constants

`kEventWindowFocusAcquired`

The user (or some other action) has caused the focus to shift to your window. In response to this, you should focus any control that might need to be focused.

The standard window handler calls the Control Manager function `SetKeyboardFocus` to highlight the first control in the window.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowFocusRelinquish`

The user has shifted the focus to another window. You should take the necessary steps to unhighlight the focus and so on.

The default behavior is to clear the current keyboard focus.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowFocusContent`

Focus should be shifted to the main content area of your window. You should set the focus to the content view of your window; if that area already has focus, then do nothing.

If the content area of the window already has focus, the standard handler does nothing. Otherwise, it calls the HIView function `HIViewAdvanceFocus` to move the focus to the first control in the content area.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

`kEventWindowFocusToolbar`

Focus should be shifted to the window's toolbar. You should set the focus to the first item in the toolbar; if the toolbar already has focus, then do nothing.

If the toolbar already has focus, the default behavior is to do nothing. Otherwise, it calls the HIView function `HIViewAdvanceFocus` to move the focus to the first control in the toolbar.

If the window does not have a toolbar, this event is not handled.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

`kEventWindowFocusDrawer`

Focus should be shifted to the drawer of a window. You should set the focus to the first item in the drawer. If the drawer already has focus, you should move the focus to the next or previous drawer, if any, depending on whether the modifiers parameter contains the shift key modifier. If the focus is not already contained within the drawer, the basic window handler responds to this event by calling `SetUserFocusWindow` and sending a `kEventWindowFocusContent` event to the appropriate drawer.

Available in Mac OS X v10.4 and later.

Declared in `CarbonEvents.h`.

Discussion
These events are related to focus changes between windows. They are generated by `SetUserFocusWindow` (page 1934). Because that function is called by default only by the standard window handler, these events are normally sent only to windows with the standard window handler installed.

Table 3-16 shows the parameters related to window focus events.

Table 3-16 Parameter names and types for window focus event kinds

Event kind	Parameter name	Parameter type
`kEventWindowFocusAcquire`	`kEventParamDirectObject`	`typeWindowRef`
`kEventWindowFocusRelinquish`	`kEventParamDirectObject`	`typeWindowRef`
`kEventWindowFocusContent`	`kEventParamDirectObject`	`typeWindowRef`
`kEventWindowFocusToolbar`	`kEventParamDirectObject`	`typeWindowRef`
`kEventWindowFocusDrawer`	`kEventParamDirectObject`	`typeWindowRef`
	`kEventParamKeyModifiers`	`typeUInt32`

Window Sheet Event Constants

Define constants for events from `kEventClassWindow` that describe window sheet changes.

```
enum {
    kEventWindowSheetOpening = 210,
    kEventWindowSheetOpened = 211,
    kEventWindowSheetClosing = 212,
    kEventWindowSheetClosed = 213
};
```

Constants

`kEventWindowSheetOpening`
A sheet is opening. This event is sent to the sheet, its parent window, and the application target before the sheet begins to open. An event handler for this event may return `userCanceledErr` if the sheet should not be opened. Any other return value is ignored, and the sheet is allowed to open.

Available in Mac OS X v10.4 and later.

Declared in `CarbonEvents.h`.

`kEventWindowSheetOpened`
A sheet has opened. This event is sent to the sheet, its parent window, and the application target after the sheet is fully open.

Available in Mac OS X v10.4 and later.

Declared in `CarbonEvents.h`.

`kEventWindowSheetClosing`
A sheet is closing. This event is sent to the sheet, its parent window, and the application target before the sheet begins to close. An event handler for this event may return `userCanceledErr` if the sheet should not close. Any other return value is ignored, and the sheet is allowed to close.

Available in Mac OS X v10.4 and later.

Declared in `CarbonEvents.h`.

`kEventWindowSheetClosed`
A sheet has closed. This event is sent to the sheet, its parent window, and the application target after the sheet is fully closed.

Available in Mac OS X v10.4 and later.

Declared in `CarbonEvents.h`.

Discussion
These events are related to sheet changes. Table 3-17 shows the parameters related to window sheet events.

Table 3-17 Parameter names and types for window sheet event kinds

Event kind	Parameter name	Parameter type
`kEventWindowSheetOpening`	`kEventParamDirectObject`	`typeWindowRef`
`kEventWindowSheetOpened`	`kEventParamDirectObject`	`typeWindowRef`
`kEventWindowSheetClosing`	`kEventParamDirectObject`	`typeWindowRef`
`kEventWindowSheetClosed`	`kEventParamDirectObject`	`typeWindowRef`

Window Drawer Event Constants

Define constants related to events from `kEventClassWindow` describing window drawer changes.

```
enum {
    kEventWindowDrawerOpening = 220,
    kEventWindowDrawerOpened = 221,
    kEventWindowDrawerClosing = 222,
    kEventWindowDrawerClosed = 223
};
```

Constants
`kEventWindowDrawerOpening`
Sent to the drawer and its parent window when the drawer is opening. If you don't want the drawer to open, your handler should return `userCanceledErr`.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

`kEventWindowDrawerOpened`
Sent to the drawer and its parent window when the drawer has fully opened.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

`kEventWindowDrawerClosing`
Sent to the drawer and its parent window when the drawer is closing. If you don't want the drawer to close, your handler should return `userCanceledErr`.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

`kEventWindowDrawerClosed`
Sent to the drawer and its parent when the drawer has fully closed.

Available in Mac OS X v10.2 and later.

Declared in `CarbonEvents.h`.

Discussion
Table 3-18 shows parameters related to window drawer events.

Table 3-18 Parameter names and types for window drawer event kinds

Event kind	Parameter name	Parameter type
`kEventWindowDrawerOpening`	`kEventParamDirectObject`	`typeWindowRef`
`kEventWindowDrawerOpened`	`kEventParamDirectObject`	`typeWindowRef`
`kEventWindowDrawerClosing`	`kEventParamDirectObject`	`typeWindowRef`
`kEventWindowDrawerClosed`	`kEventParamDirectObject`	`typeWindowRef`

Window Definition Message Constants

Define constants for events that correspond to classic WDEF messages.

```
enum {
    kEventWindowDrawFrame = 1000,
    kEventWindowDrawPart = 1001,
    kEventWindowGetRegion = 1002,
    kEventWindowHitTest = 1003,
    kEventWindowInit = 1004,
    kEventWindowDispose = 1005,
    kEventWindowDragHilite = 1006,
    kEventWindowModified = 1007,
    kEventWindowSetupProxyDragImage = 1008,
    kEventWindowStateChanged = 1009,
    kEventWindowMeasureTitle = 1010,
    kEventWindowDrawGrowBox = 1011,
    kEventWindowGetGrowImageRegion = 1012,
    kEventWindowPaint = 1013
};
```

Constants

`kEventWindowDrawFrame`

Sent by the Window Manager when it's time to draw a window's structure. This is the replacement to the old `wDraw` defProc message (though it is a special case of the 0 part code indicating to draw the entire window frame).

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowDrawPart`

Sent by the Window Manager when it's time to draw a specific part of a window's structure, such as the close box. This is typically sent during window tracking.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowGetRegion`

Sent by the Window Manager when it needs to get a specific region from a window, or when the `GetWindowRegion` function is called. The region you should modify is sent in the `kEventParamRgnHandle` parameter.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowHitTest`

Sent when the Window Manager needs to determine what part of a window would be "hit" with a given mouse location in global coordinates. If you handle this event, you should set the `kEventParamWindowDefPart` parameter to reflect the part code hit.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowInit`

Sent by the Window Manager when the window is being created. This is a hook to allow you to do any initialization you might need to do. Note that if the window definition changes, you may receive this event more than once. See the `kEventWindowDispose` event for more information.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowDispose`
Sent by the Window Manager when the window is being disposed. You should dispose of any private data structures associated with the window. Note, however, that receiving this event does not necessarily mean that the window is being destroyed. Sometimes the Window Manager may need to change the window definition (such as when `ChangeWindowAttributes` is used to change the appearance of the window). In such cases, the window receives a `kEventWindowDispose` event followed by a `kEventWindowInit` event to disconnect the old window definition and connect the new one.

If you want to know when your window is actually being destroyed, you should register for the `kEventWindowClosed` event.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowDragHilite`
Sent by the Window Manager when it is time to draw/erase any drag highlight in the window structure. This is typically sent from within the `HiliteWindowFrameForDrag` function.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowModified`
Sent by the Window Manager when it is time to redraw window structure to account for a change in the document modified state. This is typically sent from within the `SetWindowModified` function.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowSetupProxyDragImage`
Sent by the Window Manager when it is time to generate a drag image for the window proxy. This is typically sent from within the `BeginWindowProxyDrag` function.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowStateChanged`
Sent by the Window Manager when a particular window state changes. See the state-change flags in `MacWindows.h`.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowMeasureTitle`
Sent when the Window Manager needs to know how much space the window's title area takes up.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowDrawGrowBox`
This is a compatibility event used before Mac OS 8 and not useful now. When the `DrawGrowIcon` function is called, this event is sent to the window to tell it to draw the grow box. This is really needed only for windows that do not have the grow box integrated into the window frame. Scroll bar delimiter lines are also drawn.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowGetGrowImageRegion`

This is a special way for a window to override the standard resize outline for windows that do not do live resizing. As the user resizes the window, this event is sent with the current size the user has chosen expressed as a rectangle. You should calculate your window outline and modify the `kEventParamRgnHandle` parameter to reflect your desired outline.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowPaint`

Sent when it is time to draw the entire window (such as when the window is first displayed). This is a convenience event that gives you a chance to draw all the window elements at once.

If you do not handle this event, the Window Manager sends the `kEventWindowDrawFrame` event to your window and erases the content region to its background color.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

Discussion

These events, which correspond to WDEF messages, are sent to all windows, regardless of whether they have the standard window handler installed.

Table 3-19 shows the parameters related to window definition events.

Table 3-19 Parameter names and types for window definition event kinds

Event kind	Parameter name	Parameter type
`kEventWindowDrawFrame`	`kEventParamDirectObject`	`typeWindowRef`
`kEventWindowDrawPart`	`kEventParamDirectObject`	`typeWindowRef`
	`kEventParamWindowDefPart`	`typeWindowDefPartCode`
`kEventWindowGetRegion`	`kEventParamDirectObject`	`typeWindowRef`
	`kEventParamWindowRegionCode`	`typeWindowRegionCode`
	`kEventParamRgnHandle`	`typeQDRgnHandle`
`kEventWindowHitTest`	`kEventParamDirectObject`	`typeWindowRef`
	`kEventParamMouseLocation`	`typeQDPoint`
	`kEventParamWindowDefPart`	`typeWindowDefPartCode`
`kEventWindowInit`	`kEventParamDirectObject`	`typeWindowRef`
	`kEventParamWindowFeatures`	`typeUInt32`
`kEventWindowDispose`	`kEventParamDirectObject`	`typeWindowRef`
`kEventWindowDragHilite`	`kEventParamDirectObject`	`typeWindowRef`
	`kEventParamWindowDragHiliteFlag`	`typeBoolean`
`kEventWindowModified`	`kEventParamDirectObject`	`typeWindowRef`

	kEventParamWindowModifiedFlag	typeBoolean
kEventWindowSetup-ProxyDrag Image	kEventParamDirectObject	typeWindowRef
	kEventParamWindowProxyImageRgn	typeQDRgnHandle
	kEventParamWindowProxyOutlineRgn	typeQDRgnHandle
	kEventParamWindowProxyGWorlPtr	typeGWorldPtr
kEventWindowState-Changed	kEventParamDirectObject	typeWindowRef
	kEventParamWindowStateChangedFlags	typeUInt32
kEventWindowMeasure-Title	kEventParamDirectObject	typeWindowRef
	kEventParamWindowTitleFullWidth	typeSInt16
	kEventParamWindowTitleTextWidth	typeSInt16
kEventWindow-DrawGrowBox	kEventParamDirectObject	typeWindowRef
kEventWindowGetGrow-Image Region	kEventParamDirectObject	typeWindowRef
	kEventParamWindowGrowRect	typeQDRectangle
	kEventParamRgnHandle	typeQDRectangle
kEventWindowPaint	kEventParamDirectObject	typeWindowRef

Alternate Window Definition Event Constants

Define alternate names for window definition events.

```
enum {
    kEventWindowDefDrawFrame = kEventWindowDrawFrame,
    kEventWindowDefDrawPart = kEventWindowDrawPart,
    kEventWindowDefGetRegion = kEventWindowGetRegion,
    kEventWindowDefHitTest = kEventWindowHitTest,
    kEventWindowDefInit = kEventWindowInit,
    kEventWindowDefDispose = kEventWindowDispose,
    kEventWindowDefDragHilite = kEventWindowDragHilite,
    kEventWindowDefModified = kEventWindowModified,
    kEventWindowDefSetupProxyDragImage = kEventWindowSetupProxyDragImage,
    kEventWindowDefStateChanged = kEventWindowStateChanged,
    kEventWindowDefMeasureTitle = kEventWindowMeasureTitle,
    kEventWindowDefDrawGrowBox = kEventWindowDrawGrowBox,
    kEventWindowDefGetGrowImageRegion = kEventWindowGetGrowImageRegion
};
```

Constants

`kEventWindowDefDrawFrame`
Equivalent to `kEventWindowDrawFrame`.
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventWindowDefDrawPart`
Equivalent to `kEventWindowDrawPart`.
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventWindowDefGetRegion`
Equivalent to `kEventWindowGetRegion`.
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventWindowDefHitTest`
Equivalent to `kEventWindowHitTest`.
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventWindowDefInit`
Equivalent to `kEventWindowInit`.
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventWindowDefDispose`
Equivalent to `kEventWindowDispose`.
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventWindowDefDragHilite`
Equivalent to `kEventWindowDragHilite`.
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventWindowDefModified`
Equivalent to `kEventWindowModified`.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowDefSetupProxyDragImage`
Equivalent to `kEventWindowSetupProxyDragImage`.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowDefStateChanged`
Equivalent to `kEventWindowStateChanged`.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowDefMeasureTitle`
Equivalent to `kEventWindowMeasureTitle`.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowDefDrawGrowBox`
Equivalent to `kEventWindowDrawGrowBox`.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventWindowDefGetGrowImageRegion`
Equivalent to `kEventWindowGetGrowImageRegion`.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

Discussion
For clarity, you can use these event names in place of the standard window events when using them in your custom window definitions. For descriptions of these events, see "Window Definition Message Constants" (page 439).

Window Bounds Attributes

Define constants that describe how a window's bounds are changing.

```
enum {
    kWindowBoundsChangeUserDrag = (1 << 0),
    kWindowBoundsChangeUserResize = (1 << 1),
    kWindowBoundsChangeSizeChanged = (1 << 2),
    kWindowBoundsChangeOriginChanged = (1 << 3),
    kWindowBoundsChangeZoom = (1 << 4)
};
```

Constants
`kWindowBoundsChangeUserDrag`
The bounds are changing because the user is dragging the window around.

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kWindowBoundsChangeUserResize`
: The bounds are changing because the user is resizing the window.

  Available in Mac OS X v10.0 and later.

  Declared in `CarbonEvents.h`.

`kWindowBoundsChangeSizeChanged`
: The dimensions of the window (width and height) are changing.

  Available in Mac OS X v10.0 and later.

  Declared in `CarbonEvents.h`.

`kWindowBoundsChangeOriginChanged`
: The origin of the window is changing.

  Available in Mac OS X v10.0 and later.

  Declared in `CarbonEvents.h`.

`kWindowBoundsChangeZoom`
: The bounds are changing as a result of the user clicking the zoom button.

  Available in Mac OS X v10.2 and later.

  Declared in `CarbonEvents.h`.

Discussion
When the toolbox sends out a `kEventWindowBoundsChanging` or `kEventWindowBoundsChanged` event, it also sends along a parameter containing attributes of the event (`kEventParamAttributes`). You can use these attributes to determine what aspect of the window changed (origin, size, or both), and whether or not some user action is driving the change (drag, resize, or zoom).

Window Event Parameters and Types

Define constants for parameters and attributes related to window events.

```
enum {
    kEventParamWindowFeatures = 'wftr',
    kEventParamWindowDefPart = 'wdpc',
    kEventParamWindowPartCode = wpar',
    kEventParamCurrentBounds = 'crct',
    kEventParamOriginalBounds = 'orct',
    kEventParamPreviousBounds = 'prct',
    kEventParamClickActivation = 'clac',
    kEventParamWindowRegionCode = 'wshp',
    kEventParamWindowDragHiliteFlag = 'wdhf',
    kEventParamWindowModifiedFlag = 'wmff',
    kEventParamWindowProxyGWorldPtr = 'wpgw',
    kEventParamWindowProxyImageRgn = 'wpir',
    kEventParamWindowProxyOutlineRgn = 'wpor',
    kEventParamWindowStateChangedFlags = 'wscf',
    kEventParamWindowTitleFullWidth = 'wtfw',
    kEventParamWindowTitleTextWidth = 'wttw',
    kEventParamWindowGrowRect = 'grct',
    kEventParamPreviousDockRect = 'pdrc',
    kEventParamPreviousDockDevice = 'pdgd',
    kEventParamCurrentDockRect = 'cdrc',
    kEventParamCurrentDockDevice = 'cdgd',
    kEventParamWindowTransitionAction = 'wtac',
    kEventParamWindowTransitionEffect = 'wtef',
    typeWindowRegionCode = 'wshp',
    typeWindowDefPartCode = 'wdpt',
    typeClickActivationResult = 'clac',
    typeWindowTransitionAction = 'wtac',
    typeWindowTransitionEffect = 'wtef'
};
```

Constants

`kEventParamWindowFeatures`

typeUInt32

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventParamWindowDefPart`

typeWindowDefPartCode

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventParamWindowPartCode`

typeWindowPartCode

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`kEventParamCurrentBounds`

typeQDRectangle

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`kEventParamOriginalBounds`
typeQDRectangle
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamPreviousBounds`
typeQDRectangle
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamClickActivation`
typeClickActivationResult
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamWindowRegionCode`
typeWindowRegionCode
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamWindowDragHiliteFlag`
typeBoolean
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamWindowModifiedFlag`
typeBoolean
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamWindowProxyGWorldPtr`
typeGWorldPtr
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamWindowProxyImageRgn`
typeQDRgnHandle
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamWindowProxyOutlineRgn`
typeQDRgnHandle
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamWindowStateChangedFlags`
typeUInt32
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamWindowTitleFullWidth`
typeSInt16
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamWindowTitleTextWidth`
typeSInt16
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamWindowGrowRect`
typeQDRectangle
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`kEventParamPreviousDockRect`
typeHIRect
Available in Mac OS X v10.2 and later.
Declared in `CarbonEvents.h`.

`kEventParamPreviousDockDevice`
typeGDHandle
Available in Mac OS X v10.3 and later.
Declared in `CarbonEvents.h`.

`kEventParamCurrentDockRect`
typeHIRect
Available in Mac OS X v10.2 and later.
Declared in `CarbonEvents.h`.

`kEventParamCurrentDockDevice`
typeGDHandle
Available in Mac OS X v10.3 and later.
Declared in `CarbonEvents.h`.

`kEventParamWindowTransitionAction`
typeWindowTransitionAction
Available in Mac OS X v10.3 and later.
Declared in `CarbonEvents.h`.

`kEventParamWindowTransitionEffect`
typeWindowTransitionEffect
Available in Mac OS X v10.3 and later.
Declared in `CarbonEvents.h`.

`typeWindowRegionCode`
WindowRegionCode
Available in Mac OS X v10.0 and later.
Declared in `CarbonEvents.h`.

`typeWindowDefPartCode`
WindowDefPartCode

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`typeWindowPartCode`
WindowPartCode

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`typeClickActivationResult`
ClickActivationResult

Available in Mac OS X v10.0 and later.

Declared in `CarbonEvents.h`.

`typeWindowTransitionAction`
WindowTransitionAction

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`typeWindowTransitionEffect`
WindowTransitionEffect

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

Modal Window Event Parameters and Types

Define constants related to events from `kEventClassWindow` used to determine whether a mouse-down or mouse-up event is blocked by a modal window.

```
enum {
    typeModalClickResult = 'wmcr',
    typeWindowModality = 'wmod',
    kEventParamModalClickResult = typeModalClickResult,
    kEventParamModalWindow = 'mwin',
    kEventParamWindowModality = typeWindowModality
};
```

Constants

`typeModalClickResult`
HIModalClickResult

Available in Mac OS X v10.4 and later.

Declared in `CarbonEvents.h`.

`typeWindowModality`
WindowModality

Available in Mac OS X v10.4 and later.

Declared in `CarbonEvents.h`.

`kEventParamModalClickResult`
On exit, a value indicating how the click should be handled. For details, see "Modal Window Click Constants" (page 451).

Available in Mac OS X v10.4 and later.

Declared in `CarbonEvents.h`.

`kEventParamModalWindow`
On exit, the modal window that caused the click to be blocked, if any. The sender of this event uses this information to determine which window should be activated if the application is inactive. This parameter is only required if the `kEventParamModalClickResult` parameter contains `kHIModalClickIsModal`. If an event handler wants to report that a click has been blocked by modality, but cannot determine which window blocked the click, it is acceptable to either not add this parameter to the event, or to set the parameter to a `NULL` window reference.

Available in Mac OS X v10.4 and later.

Declared in `CarbonEvents.h`.

`kEventParamWindowModality`
On exit, the modality of the modal window that is in front of the clicked window, if any. This parameter is required only if the `kEventParamModalClickResult` parameter contains `kHIModalClickIsModal`.

Available in Mac OS X v10.4 and later.

Declared in `CarbonEvents.h`.

Modal Window Click Constants

Define constants that describe responses to the `kEventWindowGetModalityClick` event.

```
typedef UInt32 HIModalClickResult;
enum {
    kHIModalClickIsModal = 1 << 0,
    kHIModalClickAllowEvent = 1 << 1,
    kHIModalClickAnnounce = 1 << 2,
    kHIModalClickRaiseWindow = 1 << 3,
};
```

Constants

`kHIModalClickIsModal`
A modal window prevents the mouse event from being passed to the clicked window. If this bit is set, the `kEventParamModalWindow` and `kEventParamWindowModality` parameters should be set before the event handler returns. If this bit is clear, normal event handling occurs: the clicked window is typically z-ordered to the top of its window group, activated, becomes the user focus window, and receives the mouse event for further processing.

Available in Mac OS X v10.4 and later.

Declared in `CarbonEvents.h`.

`kHIModalClickAllowEvent`
If `kHIModalClickIsModal` is set, the `kHIModalClickAllowEvent` flag indicates whether the click event should be allowed to pass to the clicked window. If `kHIModalClickIsModal` is not set, the setting of `kHIModalClickAllowEvent` is ignored.

Available in Mac OS X v10.4 and later.

Declared in `CarbonEvents.h`.

`kHIModalClickAnnounce`
If `kHIModalClickIsModal` is set and `kHIModalClickAllowEvent` is not set, `kHIModalClickAnnounce` indicates whether the caller should announce that the click has been blocked by a modal window using the appropriate UI (typically, by calling `SysBeep`). If `kHIModalClickIsModal` is not set, or if `kHIModalClickAllowEvent` is set, the setting of `kHIModalClickAnnounce` is ignored.

Available in Mac OS X v10.4 and later.

Declared in `CarbonEvents.h`.

`kHIModalClickRaiseWindow`
If `kHIModalClickIsModal` and `kHIModalClickAllowEvent` are set, `kHIModalClickRaiseWindow` indicates whether the clicked window should be z-ordered to the top of its window group. The window is not, however, activated, nor does it become the user focus window. If `kHIModalClickIsModal` or `kHIModalClickAllowEvent` is not set, `kHIModalClickRaiseWindow` is ignored.

Available in Mac OS X v10.4 and later.

Declared in `CarbonEvents.h`.

Result Codes

The most common result codes returned by the Carbon Event Manager are listed below.

Result Code	Value	Description
`noErr`	0	No error. Available in Mac OS X v10.0 and later.
`eventAlreadyPostedErr`	-9860	Returned from `PostEventToQueue` (page 286) if the event in question is already in the queue you are posting it to (or any other queue). Available in Mac OS X v10.0 and later.
`eventTargetBusyErr`	-9861	The event target you are trying to modify is busy (for example, dispatching an event). Available in Mac OS X v10.1 and later.
`eventClassInvalidErr`	-9862	This is obsolete and will be removed. Available in Mac OS X v10.0 and later.
`eventClassIncorrectErr`	-9864	This is obsolete and will be removed. Available in Mac OS X v10.0 and later.
`eventHandlerAlreadyInstalledErr`	-9866	Returned from `InstallEventHandler` (page 274) if the handler callback you pass is already installed for a given event type you are trying to register. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
eventInternalErr	-9868	A generic error. Available in Mac OS X v10.0 and later.
eventKindIncorrectErr	-9869	This is obsolete and will be removed. Available in Mac OS X v10.0 and later.
eventParameterNotFoundErr	-9870	The piece of data you are requesting from an event is not present. Available in Mac OS X v10.0 and later.
eventNotHandledErr	-9874	This is what you should return from an event handler when your handler has received an event it doesn't currently want to (or isn't able to) handle. If you handle an event, you should return noErr from your event handler. Available in Mac OS X v10.0 and later.
eventLoopTimedOutErr	-9875	The event loop has timed out. This can be returned from calls to ReceiveNextEvent (page 289) or RunCurrentEventLoop (page 298). Available in Mac OS X v10.0 and later.
eventLoopQuitErr	-9876	The event loop was quit, probably by a call to QuitEventLoop (page 289). This can be returned from ReceiveNextEvent (page 289) or RunCurrentEventLoop (page 298). Available in Mac OS X v10.0 and later.
eventNotInQueueErr	-9877	Returned from RemoveEventFromQueue (page 294) when trying to remove an event that's not in any queue. Available in Mac OS X v10.0 and later.
eventHotKeyExistsErr	-9878	The hot key combination you chose already exists Available in Mac OS X v10.0 and later.
eventHotKeyInvalidErr	-9879	This error code is not currently used. Available in Mac OS X v10.0 and later.

Carbon Help Manager Reference

Framework: Carbon/Carbon.h

Declared in MacHelp.h

Overview

The Carbon Help Manager is an API that provides an interface for displaying short onscreen hints in help tags. You can attach identifying text to individual windows, menus, or controls. Help tags replace Balloon Help, introduced with System 7.

This document is relevant for developers creating a user interface for their Carbon application. To use this document, you should be familiar with the basics of programming with the fundamental objects of the Mac OS user interface (windows, menus, controls, and so forth)

Functions by Task

Obtaining a Reference to the Help Menu

`HMGetHelpMenu` (page 461)
Returns a reference to the Help menu.

Attaching Help Tag Content Directly to an Object

`HMSetControlHelpContent` (page 468)
Associates a help tag with a control.

`HMSetWindowHelpContent` (page 471)
Associates a help tag with a window.

`HMSetMenuItemHelpContent` (page 469)
Associates a help tag with a menu item.

`HMGetControlHelpContent` (page 460)
Returns the help tag associated with a control.

`HMGetWindowHelpContent` (page 464)
Returns the help tag associated with a window.

`HMGetMenuItemHelpContent` (page 462)
Returns the help tag associated with a menu item.

Installing and Retrieving Help Tag Callbacks

`HMInstallControlContentCallback` (page 465)
Installs a help tag callback for a control.

`HMInstallWindowContentCallback` (page 468)
Installs a help tag callback for a window.

`HMInstallMenuItemContentCallback` (page 466)
Installs a help tag callback for a menu's items.

`HMInstallMenuTitleContentCallback` (page 467)
Installs a help tag callback for a menu title.

`HMGetControlContentCallback` (page 460)
Retrieves the help tag callback associated with a control.

`HMGetWindowContentCallback` (page 464)
Retrieves the help tag callback associated with a window.

`HMGetMenuItemContentCallback` (page 462)
Retrieves the help tag callback associated with a menu's items.

`HMGetMenuTitleContentCallback` (page 463)
Retrieves the help tag callback associated with a menu's title.

Displaying and Hiding Help Tags

`HMDisplayTag` (page 459)
Displays a help tag at an application-defined location.

`HMHideTag` (page 465)
Hides the most recently displayed help tag.

Enabling and Disabling Help Tags

`HMAreHelpTagsDisplayed` (page 459)
Determines whether help tags are currently enabled.

`HMSetHelpTagsDisplayed` (page 469)
Enables or disables help tags.

Getting and Setting Help Tag Delay Time

`HMSetTagDelay` (page 470)
Sets the help tag delay time.

`HMGetTagDelay` (page 463)
Returns the current help tag delay time.

Working With Universal Procedure Pointers to Help Tag Callback Functions

`NewHMControlContentUPP` (page 475)
Creates a new universal procedure pointer (UPP) to a help tag callback for a control.

`NewHMWindowContentUPP` (page 477)
Creates a new universal procedure pointer (UPP) to a help tag callback for a window.

`NewHMMenuItemContentUPP` (page 476)
Creates a new universal procedure pointer (UPP) to a help tag callback for a menu item.

`NewHMMenuTitleContentUPP` (page 477)
Creates a new universal procedure pointer (UPP) to a help tag callback for a menu title.

`DisposeHMControlContentUPP` (page 457)
Disposes of a universal procedure pointer (UPP) to a help tag callback for a control.

`DisposeHMWindowContentUPP` (page 459)
Disposes of a universal procedure pointer (UPP) to a help tag callback for a window.

`DisposeHMMenuItemContentUPP` (page 458)
Disposes of a universal procedure pointer (UPP) to a help tag callback for a menu item.

`DisposeHMMenuTitleContentUPP` (page 458)
Disposes of a universal procedure pointer (UPP) to a help tag callback for a menu title.

`InvokeHMControlContentUPP` (page 471)
Calls your help tag callback for a control.

`InvokeHMWindowContentUPP` (page 474)
Calls your help tag callback for a window.

`InvokeHMMenuItemContentUPP` (page 472)
Calls your help tag callback for a menu item.

`InvokeHMMenuTitleContentUPP` (page 473)
Calls your help tag callback for a menu title.

Functions

DisposeHMControlContentUPP

Disposes of a universal procedure pointer (UPP) to a help tag callback for a control.

```
void DisposeHMControlContentUPP (
   HMControlContentUPP userUPP
);
```

Parameters

userUPP
The UPP to dispose of.

Discussion

To create a UPP to a help tag callback for a control, use `NewHMControlContentUPP` (page 475). For a description of the help tag callback for a control, see `HMControlContentProcPtr` (page 478).

Availability
Available in CarbonLib 1.0.2 and later.
Available in Mac OS X 10.0 and later.

Declared In
MacHelp.h

DisposeHMMenuItemContentUPP

Disposes of a universal procedure pointer (UPP) to a help tag callback for a menu item.

```
void DisposeHMMenuItemContentUPP (
   HMMenuItemContentUPP userUPP
);
```

Parameters

userUPP
The UPP to dispose of.

Discussion
To create a UPP to a help tag callback for a menu item, use NewHMMenuItemContentUPP (page 476). For a description of the help tag callback for a menu item, see HMMenuItemContentProcPtr (page 479).

Availability
Available in CarbonLib 1.0.2 and later.
Available in Mac OS X 10.0 and later.

Declared In
MacHelp.h

DisposeHMMenuTitleContentUPP

Disposes of a universal procedure pointer (UPP) to a help tag callback for a menu title.

```
void DisposeHMMenuTitleContentUPP (
   HMMenuTitleContentUPP userUPP
);
```

Parameters

userUPP
The UPP to dispose of.

Discussion
To create a UPP to a help tag callback for a menu title, use NewHMMenuTitleContentUPP (page 477). For a description of the help tag callback for a menu title, see HMMenuTitleContentProcPtr (page 480).

Availability
Available in CarbonLib 1.0.2 and later.
Available in Mac OS X 10.0 and later.

Declared In
MacHelp.h

DisposeHMWindowContentUPP

Disposes of a universal procedure pointer (UPP) to a help tag callback for a window.

```
void DisposeHMWindowContentUPP (
   HMWindowContentUPP userUPP
);
```

Parameters

userUPP

The UPP to dispose of.

Discussion

To create a UPP to a help tag callback for a window, use `NewHMWindowContentUPP` (page 477). For a description of the help tag callback for a window, see `HMWindowContentProcPtr` (page 482).

Availability

Available in CarbonLib 1.0.2 and later.

Available in Mac OS X 10.0 and later.

Declared In

`MacHelp.h`

HMAreHelpTagsDisplayed

Determines whether help tags are currently enabled.

```
Boolean HMAreHelpTagsDisplayed (
   void
);
```

Return Value

`true` if help tags are currently enabled; otherwise, `false`.

Availability

Available in CarbonLib 1.0.2 and later.

Available in Mac OS X 10.0 and later.

Not available to 64-bit applications.

Declared In

`MacHelp.h`

HMDisplayTag

Displays a help tag at an application-defined location.

```
OSStatus HMDisplayTag (
   const HMHelpContentRec *inContent
);
```

Parameters

inContent

A pointer to a help tag structure that describes the help tag you want to display.

Return Value
A result code. See "Result Codes" (page 494).

Discussion
Use the `HMDisplayTag` function to display a help tag at a location not associated with a control, window, or menu.

Availability
Available in CarbonLib 1.5 and later.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`MacHelp.h`

HMGetControlContentCallback

Retrieves the help tag callback associated with a control.

```
OSStatus HMGetControlContentCallback (
   ControlRef inControl,
   HMControlContentUPP *outContentUPP
);
```

Parameters

inControl
A reference to the control for which to retrieve the help tag callback.

outContentUPP
On return, a universal procedure pointer to the help tag callback.

Return Value
A result code. See "Result Codes" (page 494).

Discussion
To install a help tag callback for a control, use the `HMInstallControlContentCallback` (page 465) function.

Availability
Available in CarbonLib 1.0.2 and later.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`MacHelp.h`

HMGetControlHelpContent

Returns the help tag associated with a control.

```
OSStatus HMGetControlHelpContent (
   ControlRef inControl,
   HMHelpContentRec *outContent
);
```

Parameters

inControl

A reference to the control for which to retrieve the help tag.

outContent

A pointer to a help tag structure. On return, this structure describes the help tag for the control.

Return Value
A result code. See "Result Codes" (page 494).

Discussion
To attach a help tag to a control, use the `HMSetControlHelpContent` (page 468) function.

Availability
Available in CarbonLib 1.0.2 and later.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`MacHelp.h`

HMGetHelpMenu

Returns a reference to the Help menu.

```
OSStatus HMGetHelpMenu (
   MenuRef *outHelpMenu,
   MenuItemIndex *outFirstCustomItemIndex
);
```

Parameters

outHelpMenu

On return, a pointer to a menu reference to the Help menu.

outFirstCustomItemIndex

On return, a pointer to the menu item index that will be used by the first application-supplied item added to the menu. This parameter may be `NULL`.

Return Value
A result code. See "Result Codes" (page 494).

Discussion
The `HMGetHelpMenu` function returns a reference to the Help menu, to which you can add your own custom menu items.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
MacHelp.h

HMGetMenuItemContentCallback

Retrieves the help tag callback associated with a menu's items.

```
OSStatus HMGetMenuItemContentCallback (
   MenuRef inMenu,
   HMMenuItemContentUPP *outContentUPP
);
```

Parameters

inMenu
A reference to the menu for which to retrieve the menu item help tag callback.

outContentUPP
On return, a universal procedure pointer to the help tag callback.

Return Value
A result code. See "Result Codes" (page 494).

Discussion
To install a menu item help tag callback, use the `HMInstallMenuItemContentCallback` (page 466) function.

Availability
Available in CarbonLib 1.0.2 and later.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
MacHelp.h

HMGetMenuItemHelpContent

Returns the help tag associated with a menu item.

```
OSStatus HMGetMenuItemHelpContent (
   MenuRef inMenu,
   MenuItemIndex inItem,
   HMHelpContentRec *outContent
);
```

Parameters

inMenu
A reference to the menu containing the menu item for which to retrieve the help tag.

inItem
The index of the menu item.

outContent
A pointer to a help tag structure. On return, this structure describes the help tag for the menu item.

Return Value
A result code. See "Result Codes" (page 494).

Discussion
To attach a help tag to a menu item, use the `HMSetMenuItemHelpContent` (page 469) function.

Availability
Available in CarbonLib 1.0.2 and later.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`MacHelp.h`

HMGetMenuTitleContentCallback

Retrieves the help tag callback associated with a menu's title.

```
OSStatus HMGetMenuTitleContentCallback (
   MenuRef inMenu,
   HMMenuTitleContentUPP *outContentUPP
);
```

Parameters

inMenu
A reference to the menu for which to retrieve the help tag callback.

outContentUPP
On return, a universal procedure pointer to the help tag callback.

Return Value
A result code. See "Result Codes" (page 494).

Discussion
To install a menu title help tag callback, use the `HMInstallMenuTitleContentCallback` (page 467) function.

Availability
Available in CarbonLib 1.0.2 and later.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`MacHelp.h`

HMGetTagDelay

Returns the current help tag delay time.

```
OSStatus HMGetTagDelay (
   Duration *outDelay
);
```

Parameters

outDelay
On return, a pointer to the help tag delay time. A positive value represents the delay time in milliseconds; a negative value represents the delay time in microseconds.

Return Value
A result code. See "Result Codes" (page 494).

Discussion
The help tag delay time returned by the `HMGetTagDelay` function is the amount of time that the mouse must remain motionless over a control, window, menu title, or menu item before the associated help tag is displayed.

Availability
Available in CarbonLib 1.0.2 and later.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`MacHelp.h`

HMGetWindowContentCallback

Retrieves the help tag callback associated with a window.

```
OSStatus HMGetWindowContentCallback (
   WindowRef inWindow,
   HMWindowContentUPP *outContentUPP
);
```

Parameters

`inWindow`
A reference to the window for which to retrieve the help tag callback.

`outContentUPP`
On return, a universal procedure pointer to the help tag callback.

Return Value
A result code. See "Result Codes" (page 494).

Discussion
To install a help tag callback for a window, use the `HMInstallWindowContentCallback` (page 468) function.

Availability
Available in CarbonLib 1.0.2 and later.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`MacHelp.h`

HMGetWindowHelpContent

Returns the help tag associated with a window.

```
OSStatus HMGetWindowHelpContent (
   WindowRef inWindow,
   HMHelpContentRec *outContent
);
```

Parameters

inWindow

A reference to the window for which to retrieve the help tag.

outContent

A pointer to a help tag structure. On return, this structure describes the help tag for the window.

Return Value
A result code. See "Result Codes" (page 494).

Discussion
To attach a help tag to a window, use the `HMSetWindowHelpContent` (page 471) function.

Availability
Available in CarbonLib 1.0.2 and later.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`MacHelp.h`

HMHideTag

Hides the most recently displayed help tag.

```
OSStatus HMHideTag (
   void
);
```

Return Value
A result code. See "Result Codes" (page 494).

Discussion
The Carbon Help Manager automatically removes help tags from the screen when user input occurs. To hide a tag before the Carbon Help Manager removes it, use the `HMHideTag` function. `HMHideTag` removes the most recently displayed help tag from the screen. If no help tag is currently visible, `HMHideTag` does nothing.

Availability
Available in CarbonLib 1.5 and later.
Available in Mac OS X 10.1 and later.
Not available to 64-bit applications.

Declared In
`MacHelp.h`

HMInstallControlContentCallback

Installs a help tag callback for a control.

```
OSStatus HMInstallControlContentCallback (
   ControlRef inControl,
   HMControlContentUPP inContentUPP
);
```

Parameters

inControl

A reference to the control for which to install the help tag callback.

inContentUPP

A universal procedure pointer to the help tag callback.

Return Value

A result code. See "Result Codes" (page 494).

Discussion

`HMInstallControlContentCallback` associates your help tag callback with the control specified in the `inControl` parameter. Thereafter, whenever the user hovers the mouse over this control, the Carbon Help Manager calls your callback to determine the content to display in the help tag for the control. For a description of the help tag callback for a control, see `HMControlContentProcPtr` (page 478).

Availability

Available in CarbonLib 1.0.2 and later.

Available in Mac OS X 10.0 and later.

Not available to 64-bit applications.

Declared In

`MacHelp.h`

HMInstallMenuItemContentCallback

Installs a help tag callback for a menu's items.

```
OSStatus HMInstallMenuItemContentCallback (
   MenuRef inMenu,
   HMMenuItemContentUPP inContentUPP
);
```

Parameters

inMenu

A reference to the menu for which to install the help tag callback.

inContentUPP

A universal procedure pointer to the help tag callback.

Return Value

A result code. See "Result Codes" (page 494).

Discussion

`HMInstallMenuItemContentCallback` associates your help tag callback with the menu specified in the `inMenu` parameter. Thereafter, whenever the user hovers the mouse over a menu item in this menu, the Carbon Help Manager calls your callback to determine the content to display in the help tag for the menu item. For a description of the help tag callback for a menu item, see `HMMenuItemContentProcPtr` (page 479).

Special Considerations

Although the `HMInstallMenuItemContentCallback` function is available in CarbonLib, CarbonLib currently does not support menu item help tags. Any help tag content you supply for a menu item using a help tag callback in CarbonLib is not displayed.

Availability

Available in CarbonLib 1.0.2 and later.

Available in Mac OS X 10.0 and later.

Not available to 64-bit applications.

Declared In

`MacHelp.h`

HMInstallMenuTitleContentCallback

Installs a help tag callback for a menu title.

```
OSStatus HMInstallMenuTitleContentCallback (
   MenuRef inMenu,
   HMMenuTitleContentUPP inContentUPP
);
```

Parameters

inMenu

A reference to the menu for which to install the help tag callback.

inContentUPP

A universal procedure pointer to the help tag callback.

Return Value

A result code. See "Result Codes" (page 494).

Discussion

`HMInstallMenuTitleContentCallback` associates your help tag callback with the menu specified in the `inMenu` parameter. Thereafter, whenever the user hovers the mouse over the title of this menu in the menu bar, the Carbon Help Manager calls your callback to determine the content to display in the help tag for the menu title. For a description of the help tag callback for a menu title, see `HMMenuTitleContentProcPtr` (page 480).

Special Considerations

Although the `HMInstallMenuTitleContentCallback` function is available in CarbonLib, CarbonLib does not support menu title help tags. Any help tag content you supply for a menu title using a help tag callback in CarbonLib is not displayed.

Availability

Available in CarbonLib 1.0.2 and later.

Available in Mac OS X 10.0 and later.

Not available to 64-bit applications.

Declared In

`MacHelp.h`

HMInstallWindowContentCallback

Installs a help tag callback for a window.

```
OSStatus HMInstallWindowContentCallback (
   WindowRef inWindow,
   HMWindowContentUPP inContentUPP
);
```

Parameters

inWindow

A reference to the window for which to install the help tag callback.

inContentUPP

A universal procedure pointer to the help tag content callback.

Return Value

A result code. See "Result Codes" (page 494).

Discussion

`HMInstallWindowContentCallback` associates your help tag callback with the window specified in the `inWindow` parameter. Thereafter, whenever the user hovers the mouse over this window, the Carbon Help Manager calls this callback to determine the content to display in the help tag for the window. For a description of the help tag callback for a window, see `HMWindowContentProcPtr` (page 482).

Availability

Available in CarbonLib 1.0.2 and later.

Available in Mac OS X 10.0 and later.

Not available to 64-bit applications.

Declared In

`MacHelp.h`

HMSetControlHelpContent

Associates a help tag with a control.

```
OSStatus HMSetControlHelpContent (
   ControlRef inControl,
   const HMHelpContentRec *inContent
);
```

Parameters

inControl

A reference to the control with which you want to associate the help tag.

inContent

A pointer to a help tag structure that describes the help tag for the control.

Return Value

A result code. See "Result Codes" (page 494).

Discussion

Once you attach a help tag to a control using the `HMSetControlHelpContent` function, this help tag is displayed by the Carbon Help Manager whenever the user hovers the mouse over the control while help tags are enabled.

Use `HMSetControlHelpContent` to supply the content for a help tag when you create a control. To supply the content for a help tag only when the tag is about to be displayed to the user, install a help tag callback for the control with the function `HMInstallControlContentCallback` (page 465).

Availability
Available in CarbonLib 1.0.2 and later.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`MacHelp.h`

HMSetHelpTagsDisplayed

Enables or disables help tags.

```
OSStatus HMSetHelpTagsDisplayed (
   Boolean inDisplayTags
);
```

Parameters

inDisplayTags
Pass `true` to enable help tags; `false` to disable help tags.

Return Value
A result code. See "Result Codes" (page 494).

Discussion
Help tags are enabled by default when an application is launched; your application does not need to call `HMSetHelpTagsDisplayed` to turn on help tag display. Disabling help tags with `HMSetHelpTagsDisplayed` only turns off help tags in your application; help tag display in other applications is unaffected.

Availability
Available in CarbonLib 1.0.2 and later.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`MacHelp.h`

HMSetMenuItemHelpContent

Associates a help tag with a menu item.

```
OSStatus HMSetMenuItemHelpContent (
   MenuRef inMenu,
   MenuItemIndex inItem,
   const HMHelpContentRec *inContent
);
```

Parameters

inMenu
A reference to the menu containing the menu item with which to associate the help tag.

inItem
The index of the menu item.

inContent
A pointer to a help tag structure that describes the help tag for the menu item.

Return Value
A result code. See "Result Codes" (page 494).

Discussion
Once you attach a help tag to a menu item using the `HMSetMenuItemHelpContent` function, this help tag is displayed by the Carbon Help Manager whenever the user hovers the mouse over the menu item while help tags are enabled.

Use `HMSetMenuItemHelpContent` to supply the content for a help tag when you create a menu. To supply the content for a help tag only when the tag is about to be displayed to the user, install a help tag callback for the menu's items with the function `HMInstallMenuItemContentCallback` (page 466).

Special Considerations

CarbonLib does not support menu item help tags. Any help tag content you supply with the `HMSetMenuItemHelpContent` function in CarbonLib is not displayed.

In versions of Mac OS X version 10.1 and earlier, the Carbon Help Manager does not correctly interpret an empty rectangle in the `absHotRect` field of the help tag structure passed to `HMSetMenuItemHelpContent` when setting the help content for a menu title. Because there is no Menu Manager function for determining the bounds of a menu title, Apple recommends that you install a help tag callback to supply help content for a menu title. See `HMMenuTitleContentProcPtr` (page 480) for a description of the help tag callback for a menu title.

Availability
Available in CarbonLib 1.0.2 and later.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`MacHelp.h`

HMSetTagDelay

Sets the help tag delay time.

```
OSStatus HMSetTagDelay (
   Duration inDelay
);
```

Parameters

inDelay
The help tag delay time. A positive value represents the delay time in milliseconds; a negative value represents the delay time in microseconds.

Return Value
A result code. See "Result Codes" (page 494).

Discussion
The help tag delay time is the amount of time that the mouse must remain motionless over a control, window, menu title, or menu item before the associated help tag is displayed.

Availability
Available in CarbonLib 1.0.2 and later.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`MacHelp.h`

HMSetWindowHelpContent

Associates a help tag with a window.

```
OSStatus HMSetWindowHelpContent (
   WindowRef inWindow,
   const HMHelpContentRec *inContent
);
```

Parameters

inWindow
A reference to the window with which to associate the help tag.

inContent
A pointer to a help tag structure that describes the help tag for the window.

Return Value
A result code. See "Result Codes" (page 494).

Discussion
Once you attach a help tag to a window using the `HMSetWindowHelpContent` function, this help tag is displayed by the Carbon Help Manager whenever the user hovers the mouse over the window while help tags are enabled.

Use `HMSetWindowHelpContent` to supply the content for a help tag when you create a window. To supply the content for a help tag only when the tag is about to be displayed to the user, install a help tag callback for the window with the function `HMInstallWindowContentCallback` (page 468).

Availability
Available in CarbonLib 1.0.2 and later.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`MacHelp.h`

InvokeHMControlContentUPP

Calls your help tag callback for a control.

```
OSStatus InvokeHMControlContentUPP (
   ControlRef inControl,
   Point inGlobalMouse,
   HMContentRequest inRequest,
   HMContentProvidedType *outContentProvided,
   HMHelpContentRec *ioHelpContent,
   HMControlContentUPP userUPP
);
```

Parameters

inControl

A reference to the control for which the callback should supply help tag content.

inGlobalMouse

The current mouse position, in global coordinates.

inRequest

A value that specifies the type of the help tag content request. See "Content Request Types" (page 487) for a description of the possible requests.

outContentProvided

On output, a pointer to a value that indicates whether the help tag callback was able to fulfill the request specified in the `inRequest` parameter. See "Content Provided Types" (page 492) for a description of the values returned here.

ioHelpContent

A pointer to a help tag structure that describes the help tag for the control. On input, you must supply a value in the `version` field. On output, if the value of the `outContentProvided` parameter is `kHMContentProvided`, the help tag structure describes the help tag for the control.

userUPP

A universal procedure pointer to the help tag callback to invoke.

Return Value

A result code. See "Result Codes" (page 494).

Discussion

Use this function to invoke your help tag callback for a control, rather than calling your callback directly, to ensure code compatibility across different compiler targets. Typically, you do not need to invoke a help tag callback yourself. If you associate your callback with a control, using the `HMInstallControlContentCallback` (page 465) function, the operating system calls your callback when it is needed. For more information on the help tag callback for a control, see `HMControlContentProcPtr` (page 478).

Availability

Available in CarbonLib 1.0.2 and later.

Available in Mac OS X 10.0 and later.

Declared In

`MacHelp.h`

InvokeHMMenuItemContentUPP

Calls your help tag callback for a menu item.

```
OSStatus InvokeHMMenuItemContentUPP (
   const MenuTrackingData *inTrackingData,
   HMContentRequest inRequest,
   HMContentProvidedType *outContentProvided,
   HMHelpContentRec *ioHelpContent,
   HMMenuItemContentUPP userUPP
);
```

Parameters

inTrackingData

A pointer to the tracking information for the menu for which the callback should provide a menu-item help tag. The specific menu item for which the callback should provide a help tag is at the index number given in the `itemUnderMouse` field of the tracking data structure.

inRequest

A value that specifies the type of the help tag content request. See "Content Request Types" (page 487) for a description of the possible requests.

outContentProvided

On output, a pointer to a value that indicates whether the help tag callback was able to fulfill the request specified in the `inRequest` parameter. See "Content Provided Types" (page 492) for a description of the values returned here.

ioHelpContent

A pointer to a help tag structure that describes the help tag for the menu item. On input, you must supply a value in the `version` field. On output, if the value of the `outContentProvided` parameter is `kHMContentProvided`, the help tag structure describes the menu-item help tag.

userUPP

A universal procedure pointer to the help tag callback to invoke.

Return Value

A result code. See "Result Codes" (page 494).

Discussion

Use this function to invoke your help tag callback for a menu item, rather than calling your callback directly, to ensure code compatibility across different compiler targets. Typically, you do not need to invoke a help tag callback yourself. If you associate your callback with a menu's items, using the `HMInstallMenuItemContentCallback` (page 466) function, the operating system calls your callback when it is needed. For more information on the help tag callback for a menu item, see `HMMenuItemContentProcPtr` (page 479).

Availability

Available in CarbonLib 1.0.2 and later.

Available in Mac OS X 10.0 and later.

Declared In

`MacHelp.h`

InvokeHMMenuTitleContentUPP

Calls your help tag callback for a menu title.

```
OSStatus InvokeHMMenuTitleContentUPP (
   MenuRef inMenu,
   HMContentRequest inRequest,
   HMContentProvidedType *outContentProvided,
   HMHelpContentRec *ioHelpContent,
   HMMenuTitleContentUPP userUPP
);
```

Parameters

inMenu

A reference to the menu for which to provide a help tag.

inRequest

A value that specifies the type of the help tag content request. See "Content Request Types" (page 487) for a description of the possible requests.

outContentProvided

On output, a pointer to a value that indicates whether the help tag callback was able to fulfill the request specified in the `inRequest` parameter. See "Content Provided Types" (page 492) for a description of the values returned here

ioHelpContent

A pointer to a help tag structure that describes the help tag for the menu title. On input, you must supply a value in the `version` field. On output, if the value of the `outContentProvided` parameter is `kHMContentProvided`, the help tag structure describes the menu-title help tag.

userUPP

A universal procedure pointer to the help tag callback to invoke.

Return Value

A result code. See "Result Codes" (page 494).

Discussion

Use this function to invoke your help tag callback for a menu title, rather than calling your callback directly, to ensure code compatibility across different compiler targets. Typically, you do not need to invoke a help tag callback yourself. If you associate your callback with a menu, using the `HMInstallMenuTitleContentCallback` (page 467) function, the operating system calls your callback when it is needed. For more information on the help tag callback for a menu title, see `HMMenuTitleContentProcPtr` (page 480).

Availability

Available in CarbonLib 1.0.2 and later.

Available in Mac OS X 10.0 and later.

Declared In

`MacHelp.h`

InvokeHMWindowContentUPP

Calls your help tag callback for a window.

```
OSStatus InvokeHMWindowContentUPP (
   WindowRef inWindow,
   Point inGlobalMouse,
   HMContentRequest inRequest,
   HMContentProvidedType *outContentProvided,
   HMHelpContentRec *ioHelpContent,
   HMWindowContentUPP userUPP
);
```

Parameters

inWindow

A reference to the window for which the callback should provide a help tag.

inGlobalMouse

The current mouse position, in global coordinates.

inRequest

A value that specifies the type of the help tag content request. See "Content Request Types" (page 487) for a description of the possible requests.

outContentProvided

On output, a pointer to a value that indicates whether the help tag callback was able to fulfill the request specified in the `inRequest` parameter. See "Content Provided Types" (page 492) for a description of the values returned here.

ioHelpContent

A pointer to a help tag structure that describes the help tag for the window. On input, you must supply a value in the `version` field. On output, if the value of the `outContentProvided` parameter is `kHMContentProvided`, the help tag structure describes the help tag for the window.

userUPP

A universal procedure pointer to the help tag callback to invoke.

Return Value

A result code. See "Result Codes" (page 494).

Discussion

Use this function to invoke your help tag callback for a window, rather than calling your callback directly, to ensure code compatibility across different compiler targets. Typically, you do not need to invoke a help tag callback yourself. If you associate your callback with a window, using the `HMInstallWindowContentCallback` (page 468) function, the operating system calls your callback when it is needed. For more information on the help tag callback for a window, see `HMWindowContentProcPtr` (page 482).

Availability

Available in CarbonLib 1.0.2 and later.

Available in Mac OS X 10.0 and later.

Declared In

`MacHelp.h`

NewHMControlContentUPP

Creates a new universal procedure pointer (UPP) to a help tag callback for a control.

```
HMControlContentUPP NewHMControlContentUPP (
   HMControlContentProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your help tag callback. See `HMControlContentProcPtr` data type fir a description.

Return Value

On return, a UPP to the help tag callback. See `HMControlContentUPP` data type for a description.

Discussion

Pass the UPP returned by `NewHMControlContentUPP` to the `HMInstallControlContentCallback` (page 465) function to install your help tag callback with a control. When you are finished with the help tag callback—for example, when you dispose of the control—you should dispose of the UPP with the `DisposeHMControlContentUPP` (page 457) function.

Availability

Available in CarbonLib 1.0.2 and later.

Available in Mac OS X 10.0 and later.

Declared In

`MacHelp.h`

NewHMMenuItemContentUPP

Creates a new universal procedure pointer (UPP) to a help tag callback for a menu item.

```
HMMenuItemContentUPP NewHMMenuItemContentUPP (
   HMMenuItemContentProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your help tag callback.

Return Value

On return, a UPP to the help tag callback. See the `HMMenuItemContentUPP` data type for a description.

Discussion

Pass the UPP returned by `NewHMMenuItemContentUPP` to the `HMInstallMenuItemContentCallback` (page 466) function to install your help tag callback with a menu's items. When you are finished with the help tag callback—for example, when you dispose of the menu—you should dispose of the UPP with the `DisposeHMMenuItemContentUPP` (page 458) function.

Availability

Available in CarbonLib 1.0.2 and later.

Available in Mac OS X 10.0 and later.

Declared In

`MacHelp.h`

NewHMMenuTitleContentUPP

Creates a new universal procedure pointer (UPP) to a help tag callback for a menu title.

```
HMMenuTitleContentUPP NewHMMenuTitleContentUPP (
   HMMenuTitleContentProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your help tag callback.

Return Value

On return, a UPP to the help tag callback. See the `HMMenuTitleContentUPP` data type for a description

Discussion

Pass the UPP returned by `NewHMMenuTitleContentUPP` to the `HMInstallMenuTitleContentCallback` (page 467) function to install your help tag callback with a menu's title. When you are finished with the help tag callback—for example, when you dispose of the menu—you should dispose of the UPP with the `DisposeHMMenuTitleContentUPP` (page 458) function.

Availability

Available in CarbonLib 1.0.2 and later.

Available in Mac OS X 10.0 and later.

Declared In

`MacHelp.h`

NewHMWindowContentUPP

Creates a new universal procedure pointer (UPP) to a help tag callback for a window.

```
HMWindowContentUPP NewHMWindowContentUPP (
   HMWindowContentProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your help tag callback.

Return Value

On return, a UPP to the help tag callback function. See the `HMWindowContentUPP` data type for a description.

Discussion

Pass the UPP returned by `NewHMWindowContentUPP` to the `HMInstallWindowContentCallback` (page 468) function to install your help tag callback with a window. When you are finished with the help tag callback—for example, when you dispose of the window—you should dispose of the UPP with the `DisposeHMWindowContentUPP` (page 459) function.

Availability

Available in CarbonLib 1.0.2 and later.

Available in Mac OS X 10.0 and later.

Declared In

`MacHelp.h`

Callbacks

HMControlContentProcPtr

Defines a pointer to the help tag callback for a control. Your help tag callback provides help tag content for a control.

```
typedef OSStatus(* HMControlContentProcPtr)
(
    ControlRef inControl,
    Point inGlobalMouse,
    HMContentRequest inRequest,
    HMContentProvidedType *outContentProvided,
    HMHelpContentPtr ioHelpContent
);
```

If you name your function `MyHMControlContentCallback`, you would declare it like this:

```
OSStatus MyHMControlContentCallback (
    ControlRef inControl,
    Point inGlobalMouse,
    HMContentRequest inRequest,
    HMContentProvidedType *outContentProvided,
    HMHelpContentPtr ioHelpContent
);
```

Parameters

inControl
A reference to the control for which your callback should supply help tag content.

inGlobalMouse
The current mouse position, in global coordinates.

inRequest
A value that specifies the type of the help tag content request. See "Content Request Types" (page 487) for a description of the possible requests.

outContentProvided
On output, a pointer to a value that indicates whether your help tag callback was able to fulfill the request specified in the `inRequest` parameter. Your callback should return one of the constants described in "Content Provided Types" (page 492).

ioHelpContent
A pointer to a help tag structure that describes the help tag for the control. On input, the Carbon Help Manager supplies a value in the `version` field. If the value of the `inRequest` parameter is `kHMSupplyContent`, your callback must fill in the remaining fields of the structure or specify that it was unable to fulfill the help tag content request.

Return Value

A result code. See "Result Codes" (page 494).

Discussion
When the user hovers the mouse over a control for which you've registered a help tag callback, the Carbon Help Manager calls your callback with a `kHMSupplyContent` request in the `inRequest` parameter. To supply a help tag, your callback should fill in the fields of the help tag structure pointed to in the `ioHelpContent` parameter.

When the help tag for the control is no longer needed, the Carbon Help Manager calls your callback with a `kHMDisposeContent` request. When you receive this request, you should free any memory allocated for the help tag content and perform any other cleanup necessary before the Carbon Help Manager removes the help tag from the screen.

If your help tag callback handles the content request, your callback should return the constant `kHMContentProvided` in the `outContentProvided` parameter. Otherwise, your callback should indicate that it was unable to handle the help tag content request by returning the constant `kHMContentNotProvided` in the `outContentProvided` parameter.

To register a help tag callback with a control, pass a universal procedure pointer (UPP) to your callback function to the `HMInstallControlContentCallback` (page 465) function. You can create a UPP to your callback function with the `NewHMControlContentUPP` (page 475) function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacHelp.h`

HMMenuItemContentProcPtr

Defines a pointer to the help tag callback for a menu item. Your help tag callback provides help tag content for a menu item.

```
typedef OSStatus(* HMMenuItemContentProcPtr)
(
    const MenuTrackingData *inTrackingData,
    HMContentRequest inRequest,
    HMContentProvidedType *outContentProvided,
    HMHelpContentPtr ioHelpContent
);
```

If you name your function `MyHMMenuItemContentCallback`, you would declare it like this:

```
OSStatus MyHMMenuItemContentCallback (
    const MenuTrackingData *inTrackingData,
    HMContentRequest inRequest,
    HMContentProvidedType *outContentProvided,
    HMHelpContentPtr ioHelpContent
);
```

Parameters

`inTrackingData`

A pointer to the tracking information for the menu for which your callback should provide a menu-item help tag. The specific menu item for which your callback should provide a help tag is at the index number given in the `itemUnderMouse` field of the tracking data structure.

inRequest

A value that specifies the type of the help tag content request. See "Content Request Types" (page 487) for a description of the possible requests.

outContentProvided

On output, a pointer to a value that indicates whether your help tag callback was able to fulfill the request specified in the `inRequest` parameter. Your callback should return one of the constants described in "Content Provided Types" (page 492).

ioHelpContent

A pointer to a help tag structure that describes the help tag for the menu item. On input, the Carbon Help Manager supplies a value in the `version` field. If the value of the `inRequest` parameter is `kHMSupplyContent`, your callback must fill in the remaining fields of the structure or specify that it was unable to fulfill the help tag content request.

Return Value
A result code. See "Result Codes" (page 494).

Discussion
When the user hovers the mouse over a menu item in a menu for which you've registered a help tag callback, the Carbon Help Manager calls your callback with a `kHMSupplyContent` request in the `inRequest` parameter. To supply a help tag, your callback should fill in the remaining fields of the help tag structure pointed to in the `ioHelpContent` parameter.

When the help tag for the menu item is no longer needed, the Carbon Help Manager calls your callback with a `kHMDisposeContent` request. When you receive this request, you should free any memory allocated for the help tag content and perform any other cleanup necessary before the Carbon Help Manager removes the help tag from the screen.

If your help tag callback handles the request for help tag content, your callback should return the constant `kHMContentProvided` in the `outContentProvided` parameter. Otherwise, your callback should indicate that it was unable to handle the help tag content request by returning the constant `kHMContentNotProvided` in the `outContentProvided` parameter.

To register a menu-item help tag callback with a menu, pass a universal procedure pointer (UPP) to your callback function to the `HMInstallMenuItemContentCallback` (page 466) function. You can create a UPP to your callback function with the `NewHMMenuItemContentUPP` (page 476) function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacHelp.h`

HMMenuTitleContentProcPtr

Defines a pointer to the help tag callback for a menu title. Your help tag callback provides help tag content for a menu's title.

```
typedef OSStatus(* HMMenuTitleContentProcPtr)
(
    MenuRef inMenu,
    HMContentRequest inRequest,
    HMContentProvidedType *outContentProvided,
    HMHelpContentPtr ioHelpContent
);
```

If you name your function `MyHMMenuTitleContentCallback`, you would declare it like this:

```
OSStatus MyHMMenuTitleContentCallback (
    MenuRef inMenu,
    HMContentRequest inRequest,
    HMContentProvidedType *outContentProvided,
    HMHelpContentPtr ioHelpContent
);
```

Parameters

inMenu

A reference to the menu for which to provide help tag content.

inRequest

A value that specifies the type of the help tag content request. See "Content Request Types" (page 487) for a description of the possible requests.

outContentProvided

On output, a pointer to a value that indicates whether your help tag callback was able to fulfill the request specified in the `inRequest` parameter. Your callback should return one of the constants described in "Content Provided Types" (page 492).

ioHelpContent

A pointer to a help tag structure that describes the help tag for the menu title. On input, the Carbon Help Manager supplies a value in the `version` field. If the value of the `inRequest` parameter is `kHMSupplyContent`, your callback must fill in the remaining fields of the structure or specify that it was unable to fulfill the help tag content request.

Return Value

A result code. See "Result Codes" (page 494).

Discussion

When the user hovers the mouse over the title of a menu for which you've registered a help tag callback, the Carbon Help Manager calls your callback with a `kHMSupplyContent` request in the `inRequest` parameter. To supply a help tag, your callback should fill in the remaining fields of the help tag structure pointed to in the `ioHelpContent` parameter.

When the help tag for the menu title is no longer needed, the Carbon Help Manager calls your callback with a `kHMDisposeContent` request. When you receive this request, you should free any memory allocated for the help tag content and perform any other cleanup necessary before the Carbon Help Manager removes the help tag from the screen.

If your help tag callback handles the request for help tag content, your callback should return the constant `kHMContentProvided` in the `outContentProvided` parameter. Otherwise, your callback should indicate that it was unable to handle the help tag content request by returning the constant `kHMContentNotProvided` in the `outContentProvided` parameter.

To register a menu title help tag callback with a menu, pass a universal procedure pointer (UPP) to your callback function to the `HMInstallMenuTitleContentCallback` (page 467) function. You can create a UPP to your callback function with the `NewHMMenuTitleContentUPP` (page 477) function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacHelp.h`

HMWindowContentProcPtr

Defines a pointer to the help tag callback for a window. Your help tag callback provides help tag content for a window.

```
typedef OSStatus(* HMWindowContentProcPtr)
(
    WindowRef inWindow,
    Point inGlobalMouse,
    HMContentRequest inRequest,
    HMContentProvidedType *outContentProvided,
    HMHelpContentPtr ioHelpContent
);
```

If you name your function `MyHMWindowContentCallback`, you would declare it like this:

```
OSStatus MyHMWindowContentCallback (
    WindowRef inWindow,
    Point inGlobalMouse,
    HMContentRequest inRequest,
    HMContentProvidedType *outContentProvided,
    HMHelpContentPtr ioHelpContent
);
```

Parameters

inWindow
A reference to the window for which your callback should provide help tag content.

inGlobalMouse
The current mouse position, in global coordinates.

inRequest
A value that specifies the type of the help tag content request. See "Content Request Types" (page 487) for a description of the possible requests.

outContentProvided
On output, a pointer to a value that indicates whether your help tag callback was able to fulfill the request specified in the `inRequest` parameter. Your callback should return one of the constants described in "Content Provided Types" (page 492).

ioHelpContent
A pointer to a help tag structure that describes the help tag for the window. On input, the Carbon Help Manager supplies a value in the `version` field. If the value of the `inRequest` parameter is `kHMSupplyContent`, your callback must fill in the remaining fields of the structure or specify that it was unable to fulfill the help tag content request.

Return Value
A result code. See "Result Codes" (page 494).

Discussion
When the user hovers the mouse over a window for which you've registered a help tag callback, the Carbon Help Manager calls your callback with a `kHMSupplyContent` request in the `inRequest` parameter. To supply a help tag, your callback should fill in the fields of the help tag structure pointed to in the `ioHelpContent` parameter.

When the help tag for the window is no longer needed, the Carbon Help Manager calls your callback with a `kHMDisposeContent` request. When you receive this request, you should free any memory allocated for the help tag content and perform any other cleanup necessary before the Carbon Help Manager removes the help tag from the screen.

If your help tag callback handles the request for help tag content, your callback should return the constant `kHMContentProvided` in the `outContentProvided` parameter. Otherwise, your callback should indicate that it was unable to handle the help tag content request by returning the constant `kHMContentNotProvided` in the `outContentProvided` parameter.

To register a help tag callback with a window, pass a universal procedure pointer (UPP) to your callback function to the `HMInstallWindowContentCallback` (page 468) function. You can create a UPP to your callback function with the `NewHMWindowContentUPP` (page 477) function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacHelp.h`

Data Types

HMHelpContentRec

Describes a help tag.

```
struct HMHelpContentRec {
    SInt32 version;
    Rect absHotRect;
    HMTagDisplaySide tagSide;
    HMHelpContent content[2];
};
typedef struct HMHelpContentRec HMHelpContentRec;
```

Fields
`version`
The structure version.

`absHotRect`

The hot rectangle for the help tag, expressed in global coordinates. The hot rectangle defines the area on the screen over which the user must hover the mouse to trigger the Carbon Help Manager to display the help tag.

You may pass an empty rectangle—a rectangle with coordinates (0,0,0,0)—in this field. The Carbon Help Manager automatically substitutes the current location of the control, window, or menu item for the empty rectangle when the help tag is displayed.

`tagSide`

A value that specifies which side of the hot rectangle the help tag is displayed. For a description of the constants used here, see "Help Tag Display Locations" (page 489).

`content`

An array of two help tag content structures describing the help message for the tag. The first structure describes the content displayed by default in the help tag (the minimum content). The second structure describes additional content that is displayed in the help tag if the user holds down the Command key while the help tag is displayed (the maximum, or expanded, content).

Special Considerations

On Mac OS X version 10.1.x and earlier, you cannot supply an empty hot rectangle from a help tag callback. You can, however, supply an empty hot rectangle to the `HMSetControlHelpContent`, `HMSetWindowHelpContent`, and `HMSetMenuItemHelpContent` functions.

When you supply an empty hot rectangle for a control help tag, the Carbon Help Manager uses the function `LocalToGlobal` to convert the control's bounds from window-local coordinates into global coordinates. `LocalToGlobal` only returns true global coordinates if the port origin is (0,0). If your application sets the port origin to a non-zero value, it should do so only temporarily; your application should never leave the port origin set to a non-zero value across calls to the event system that may attempt to display a help tag. Setting the port origin to a non-zero value typically prevents help tags from being displayed.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacHelp.h`

HMHelpContentPtr

A pointer to a help tag structure.

```
typedef HMHelpContentRec* HMHelpContentPtr;
```

Discussion
See `HMHelpContentRec` (page 483) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacHelp.h`

HMHelpContent

Contains a help tag message.

```
struct HMHelpContent {
    HMContentType contentType;
    union {
        CFStringRef tagCFString;
        Str255 tagString;
        HMStringResType tagStringRes;
        TEHandle tagTEHandle;
        SInt16 tagTextRes;
          SInt16 tagStrRes;
    } u;
};
```

Fields

`contentType`

A value that indicates the format of the help content contained in the structure. The help tag content formats that are allowed are described by the constants "Help Tag Content Types" (page 488).

`tagCFString`

If the value of the `contentType` field is `kHMCFStringContent`, a CFString specifying the help tag message. If the value of the `contentType` field is `kHMCFStringLocalizedContent`, a CFString containing the name of the localized help tag message in the `Localizable.strings` file.

`tagString`

If the value of the `contentType` field is `kHMPascalStrContent`, a Pascal string specifying the help tag message.

`tagStringRes`

If the value of the `contentType` field is `kHMStringResContent`, a 'STR#' resource ID and an index number specifying the help tag message.

`tagTEHandle`

If the value of the `contentType` field is `kHMTEHandleContent`, a TextEdit handle specifying the help tag message. This type of help tag content is only supported in CarbonLib and in Mac OS X version 10.2 and later.

`tagTextRes`

If the value of the `contentType` field is `kHMTextResContent`, the resource ID of a 'TEXT' resource and a 'styl' resource describing the help tag message. This type of help tag content is only supported in CarbonLib and in Mac OS X version 10.2 and later.

`tagStrRes`

If the value of the `contentType` field is `kHMStrResContent`, a 'STR ' resource ID, specifying the help tag message.

Discussion

The `HMHelpContent` structure is used in the `content` field of the `HMHelpContentRec` (page 483) structure to hold the help content associated with a help tag. The `HMHelpContent` structure describes a single help message.

HMControlContentUPP

Defines a universal procedure pointer to the help tag callback for a control.

```
typedef struct OpaqueHMControlContentProcPtr* HMControlContentUPP;
```

Discussion

For more information, see the description of the `HMControlContentProcPtr` (page 478) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacHelp.h`

HMWindowContentUPP

Defines a universal procedure pointer to the help tag callback for a window.

```
typedef struct OpaqueHMWindowContentProcPtr* HMWindowContentUPP;
```

Discussion
For more information, see the description of the `HMWindowContentProcPtr` (page 482) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacHelp.h`

HMMenuItemContentUPP

Defines a universal procedure pointer to the help tag callback for a menu item.

```
typedef struct OpaqueHMMenuItemContentProcPtr* HMMenuItemContentUPP;
```

Discussion
For more information, see the description of the `HMMenuItemContentProcPtr` (page 479) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacHelp.h`

HMMenuTitleContentUPP

Defines a universal procedure pointer to the help tag callback for a menu title.

```
typedef struct OpaqueHMMenuTitleContentProcPtr* HMMenuTitleContentUPP;
```

Discussion
For more information, see the description of the `HMMenuTitleContentProcPtr` (page 480) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacHelp.h`

Constants

Help Manager Version

Represents the current version of the `HMHelpContentRec` structure.

```
enum {
    kMacHelpVersion = 3
};
```

Constants

`kMacHelpVersion`

The current structure version.

Available in Mac OS X v10.0 and later.

Declared in `MacHelp.h`.

Discussion

When the Carbon Help Manager calls your help tag callback, it supplies the `kMacHelpVersion` constant in the `version` field of the `HMHelpContentRec` (page 483) structure it passes to your callback.

Content Request Types

Identify the type of request made to a help tag callback.

```
typedef SInt16 HMContentRequest;
enum {
    kHMSupplyContent =                0,
    kHMDisposeContent =                1
};
```

Constants

`kHMSupplyContent`

The help tag callback should supply help content.

Available in Mac OS X v10.0 and later.

Declared in `MacHelp.h`.

`kHMDisposeContent`

The help tag callback should dispose of help content.

Available in Mac OS X v10.0 and later.

Declared in `MacHelp.h`.

Discussion

These constants are passed to your help tag callback in the `inRequest` parameter to indicate the nature of the help tag content request. In particular, the Carbon Help Manager passes `kHMSupplyContent` when the user pauses with the mouse above a user interface object for which a help tag callback is registered. When the user moves the mouse away from the object, the Carbon Help Manager passes `kHMDisposeContent` to your callback.

Help Tag Content Types

Identify the type of content contained in a help tag.

```
typedef UInt32 HMContentType;
enum {
    kHMNoContent = 'none',
    kHMCFStringContent = 'cfst',
    kHMCFStringLocalizedContent = ‘cfsl’,
    kHMPascalStrContent = 'pstr',
    kHMStringResContent = 'str#',
    kHMTEHandleContent = 'txth',
    kHMTextResContent = 'text',
    kHMStrResContent = 'str '
};
```

Constants

`kHMNoContent`

The help tag contains no content.

Available in Mac OS X v10.0 and later.

Declared in `MacHelp.h`.

`kHMCFStringContent`

The help tag content is a CFString object.

Available in Mac OS X v10.0 and later.

Declared in `MacHelp.h`.

`kHMCFStringLocalizedContent`

The help tag content is a localized string contained in the `Localizeable.strings` file. The `content` field of the help tag structure contains a CFString key which the Carbon Help Manager uses to retrieve the help tag string from the `Localizable.strings` file in the appropriate language folder.

Available in Mac OS X v10.2 and later.

Declared in `MacHelp.h`.

`kHMPascalStrContent`

The help tag content is a Pascal string.

Available in Mac OS X v10.0 and later.

Declared in `MacHelp.h`.

`kHMStringResContent`

The help tag content is a text string, stored in the list of strings in a resource of type '`STR#`'.

Available in Mac OS X v10.0 and later.

Declared in `MacHelp.h`.

`kHMTEHandleContent`

The help tag content is contained in a TextEdit record, referred to by a TextEdit handle. This type of help tag content is only supported in CarbonLib and in Mac OS X version 10.2 and later.

Available in Mac OS X v10.0 and later.

Declared in `MacHelp.h`.

`kHMTextResContent`
The help tag content is styled text, described by the combination of a text resource ('`TEXT`') and a style resource ('`styl`'). This type of help tag content is only supported in CarbonLib and in Mac OS X version 10.2 and later.

Available in Mac OS X v10.0 and later.

Declared in `MacHelp.h`.

`kHMStrResContent`
The help tag content is a text string stored in a resource of type '`STR `'.

Available in Mac OS X v10.0 and later.

Declared in `MacHelp.h`.

Discussion
These constants are used in the `contentType` field of the `HMHelpContent` (page 484) structure to identify the format of the help content contained in that structure.

Help Tag Display Locations

Specify which side of the hot rectangle to display the help tag.

```
typedef SInt16 HMTagDisplaySide;
enum {
    kHMDefaultSide = 0,
    kHMOutsideTopScriptAligned = 1,
    kHMOutsideLeftCenterAligned = 2,
    kHMOutsideBottomScriptAligned = 3,
    kHMOutsideRightCenterAligned = 4,
    kHMOutsideTopLeftAligned = 5,
    kHMOutsideTopRightAligned = 6,
    kHMOutsideLeftTopAligned = 7,
    kHMOutsideLeftBottomAligned = 8,
    kHMOutsideBottomLeftAligned = 9,
    kHMOutsideBottomRightAligned = 10,
    kHMOutsideRightTopAligned = 11,
    kHMOutsideRightBottomAligned = 12,
    kHMOutsideTopCenterAligned = 13,
    kHMOutsideBottomCenterAligned = 14,
    kHMInsideRightCenterAligned = 15,
    kHMInsideLeftCenterAligned = 16,
    kHMInsideBottomCenterAligned = 17,
    kHMInsideTopCenterAligned = 18,
    kHMInsideTopLeftCorner = 19,
    kHMInsideTopRightCorner = 20,
    kHMInsideBottomLeftCorner = 21,
    kHMInsideBottomRightCorner = 22,
    kHMAbsoluteCenterAligned = 23
};
```

Constants
`kHMDefaultSide`
Display the help tag at the default system location. The system default is below the hot rectangle, horizontally centered.

Available in Mac OS X v10.0 and later.

Declared in `MacHelp.h`.

`kHMOutsideTopScriptAligned`
Display the help tag above the hot rectangle, aligned with the right or left side, depending upon the direction of the system script.

Available in Mac OS X v10.0 and later.

Declared in `MacHelp.h`.

`kHMOutsideLeftCenterAligned`
Display the help tag to the left of the hot rectangle, centered vertically.

Available in Mac OS X v10.0 and later.

Declared in `MacHelp.h`.

`kHMOutsideBottomScriptAligned`
Display the help tag below the hot rectangle, aligned with the right or left side, depending upon the direction of the system script.

Available in Mac OS X v10.0 and later.

Declared in `MacHelp.h`.

`kHMOutsideRightCenterAligned`
Display the help tag to the right of the hot rectangle, centered vertically.

Available in Mac OS X v10.0 and later.

Declared in `MacHelp.h`.

`kHMOutsideTopLeftAligned`
Display the help tag above the hot rectangle, with their left edges aligned.

Available in Mac OS X v10.0 and later.

Declared in `MacHelp.h`.

`kHMOutsideTopRightAligned`
Display the help tag above the hot rectangle, with their right edges aligned.

Available in Mac OS X v10.0 and later.

Declared in `MacHelp.h`.

`kHMOutsideLeftTopAligned`
Display the help tag to the left of the hot rectangle, with their top edges aligned.

Available in Mac OS X v10.0 and later.

Declared in `MacHelp.h`.

`kHMOutsideLeftBottomAligned`
Display the help tag to the left of the hot rectangle, with their bottom edges aligned.

Available in Mac OS X v10.0 and later.

Declared in `MacHelp.h`.

`kHMOutsideBottomLeftAligned`
Display the help tag below the hot rectangle, with their left edges aligned.

Available in Mac OS X v10.0 and later.

Declared in `MacHelp.h`.

`kHMOutsideBottomRightAligned`
Display the help tag below the hot rectangle, with their right edges aligned.

Available in Mac OS X v10.0 and later.

Declared in `MacHelp.h`.

`kHMOutsideRightTopAligned`
Display the help tag to the right of the hot rectangle, with their top edges aligned.

Available in Mac OS X v10.0 and later.

Declared in `MacHelp.h`.

`kHMOutsideRightBottomAligned`
Display the help tag to the right of the hot rectangle, with their bottom edges aligned.

Available in Mac OS X v10.0 and later.

Declared in `MacHelp.h`.

`kHMOutsideTopCenterAligned`
Display the help tag above the hot rectangle, centered horizontally.

Available in Mac OS X v10.0 and later.

Declared in `MacHelp.h`.

`kHMOutsideBottomCenterAligned`
Display the help tag below the hot rectangle, centered horizontally.

Available in Mac OS X v10.0 and later.

Declared in `MacHelp.h`.

`kHMInsideRightCenterAligned`
Display the help tag inside the hot rectangle, with their right edges aligned and centered vertically.

Available in Mac OS X v10.0 and later.

Declared in `MacHelp.h`.

`kHMInsideLeftCenterAligned`
Display the help tag inside the hot rectangle, with their left edges aligned and centered vertically.

Available in Mac OS X v10.0 and later.

Declared in `MacHelp.h`.

`kHMInsideBottomCenterAligned`
Display the help tag inside the hot rectangle, with their bottom edges aligned and centered horizontally.

Available in Mac OS X v10.0 and later.

Declared in `MacHelp.h`.

`kHMInsideTopCenterAligned`
Display the help tag inside the hot rectangle, with their top edges aligned and centered horizontally.

Available in Mac OS X v10.0 and later.

Declared in `MacHelp.h`.

`kHMInsideTopLeftCorner`
Display the help tag inside the hot rectangle, with their top and left edges aligned.

Available in Mac OS X v10.0 and later.

Declared in `MacHelp.h`.

`kHMInsideTopRightCorner`
Display the help tag inside the hot rectangle, with their top and right edges aligned.

Available in Mac OS X v10.0 and later.

Declared in `MacHelp.h`.

`kHMInsideBottomLeftCorner`
Display the help tag inside the hot rectangle, with their bottom and left edges aligned.

Available in Mac OS X v10.0 and later.

Declared in `MacHelp.h`.

`kHMInsideBottomRightCorner`
Display the help tag inside the hot rectangle, with their bottom and right edges aligned.

Available in Mac OS X v10.0 and later.

Declared in `MacHelp.h`.

`kHMAbsoluteCenterAligned`
Display the help tag centered vertically and horizontally within the hot rectangle.

Available in Mac OS X v10.0 and later.

Declared in `MacHelp.h`.

Discussion
The locations described by these constants are relative to the help tag's hot rectangle, which defines, in global coordinates, the area on the screen with which the help tag is associated. These constants are used in the `tagSide` field of the `HMHelpContentRec` (page 483) structure.

Content Provided Types

Indicate whether or not help tag content has been supplied.

```
typedef SInt16 HMContentProvidedType;
enum {
    kHMContentProvided = 0,
    kHMContentNotProvided = 1,
    kHMContentNotProvidedDontPropagate = 2
};
```

Constants
`kHMContentProvided`
Help tag content has been provided.

Available in Mac OS X v10.0 and later.

Declared in `MacHelp.h`.

`kHMContentNotProvided`
Help tag content was not provided. When your callback returns this constant, the Carbon Help Manager calls up to the next help tag callback in the hierarchy. If that callback also fails to provide help content, the Carbon Help Manager continues to propagate the request for help tag content until the request is fulfilled, the top of the hierarchy is reached, or a help tag callback tells the Carbon Help Manager to stop propagating the request.

Available in Mac OS X v10.0 and later.

Declared in `MacHelp.h`.

`kHMContentNotProvidedDontPropagate`
Help tag content was not provided. When your callback returns this constant, the Carbon Help Manager assumes that there is no help content for the control, window, menu title, or menu item and does not propagate the request.

Available in Mac OS X v10.0 and later.

Declared in `MacHelp.h`.

Discussion
Your help tag callback function should return one of these constants in its `outContentProvided` parameter.

Help Tag Content Indexes

Identify an index into the array of help tag content structures in a help tag.

```
enum {
    kHMMinimumContentIndex = 0,
    kHMMaximumContentIndex = 1
};
```

Constants

`kHMMinimumContentIndex`
The index of the help tag's minimum (or default) content.
Available in Mac OS X v10.0 and later.
Declared in `MacHelp.h`.

`kHMMaximumContentIndex`
The index of the help tag's maximum (or expanded) content.
Available in Mac OS X v10.0 and later.
Declared in `MacHelp.h`.

Discussion
These constants are used to access the minimum or maximum content of a help tag, in the `content` field of the `HMHelpContentRec` (page 483) structure.

Obsolete Help Tag Display Locations

Indicate the side of the hot rectangle on which the help tag should be displayed.

```
enum {
    kHMTopSide = 1,
    kHMLeftSide = 2,
    kHMBottomSide = 3,
    kHMRightSide = 4,
    kHMTopLeftCorner = 5,
    kHMTopRightCorner = 6,
    kHMLeftTopCorner = 7,
    kHMLeftBottomCorner = 8,
    kHMBottomLeftCorner = 9,
    kHMBottomRightCorner = 10,
    kHMRightTopCorner = 11,
    kHMRightBottomCorner = 12
};
```

Discussion
These constants have been replaced by the constants described in "Help Tag Display Locations"; you should use those constants instead.

Result Codes

The most common result codes returned by the Carbon Help Manager are listed in the table below.

Result Code	Value	Description
`errHMIllegalContentForMinimumState`	-10980	Minimum content for a help tag is unrecognized. Available in Mac OS X v10.0 and later.
`errHMIllegalContentForMaximumState`	-10981	Maximum content for a help tag is unrecognized. Available in Mac OS X v10.0 and later.

Color Picker Manager Reference

Framework: Carbon/Carbon.h

Declared in ColorPicker.h

Overview

Using Color Picker Manager functions, your application can use the standard user interface for soliciting color choices from users. Your application can use these functions to display, respond to events within, and close a color picker dialog box. The Color Picker Manager provides standard color pickers (which allow users to select colors from ranges of colors) and a standard dialog box allowing users to interact with the color picker. Your application can also create its own color pickers and dialog boxes to work with the Color Picker Manager.

Carbon supports the Color Picker Manager functions that most applications rely on, `GetColor`, `PickColor`, and `NPickColor`. In addition, Carbon will support all the functions described as supported in "Technote 1100: Color Picker 2.1."

That same Technote specifies certain functions from earlier versions of the Color Picker Manager that are no longer supported by the Mac OS; Carbon will not support these functions. In addition, many of the functions that start with the word "Picker", such as `PickerInit`, `PickerGetProfile`, and `PickerItemHit`, are no longer supported. These routines provided low-level access to the Color Picker Manager that was rarely used.

Functions by Task

Converting Between SmallFract and Fixed Values

`Fix2SmallFract` (page 498)
Converts a fixed integer to a `SmallFract` value.

`SmallFract2Fix` (page 506)
Converts a `SmallFract` value to a fixed integer.

Converting Colors Among Color Models

`CMY2RGB` (page 497)
Converts a CMY color to its equivalent RGB color.

`HSL2RGB` (page 500)
Converts an HSL color to an RGB color.

`HSV2RGB` (page 500)
Converts an HSV color to an RGB color.

`RGB2CMY` (page 505)
Converts an RGB color to a CMY color.

`RGB2HSL` (page 505)
Converts an RGB color to an HSL color.

`RGB2HSV` (page 505)
Converts an RGB color to an HSV color.

Using the Standard Color Picker Dialog Box

`NPickColor` (page 503)
Displays the Color Picker dialog.

`GetColor` (page 499)
Requests the user to choose a color. This function is obsolete; use the `PickColor` function instead.

`PickColor` (page 504)
Requests the user to choose a color from the standard color picker dialog box.

Working With Universal Procedure Pointers

`NewUserEventUPP` (page 503)
Creates a universal procedure pointer (UPP) to an event filter callback.

`DisposeUserEventUPP` (page 498)
Disposes of a a universal procedure pointer (UPP) to an event filter callback.

`InvokeUserEventUPP` (page 501)
Invokes an event filter callback.

`NewNColorChangedUPP` (page 502)
Creates a universal procedure pointer (UPP) to a color-changed callback.

`InvokeNColorChangedUPP` (page 501)
Invokes a color-changed callback.

`DisposeNColorChangedUPP` (page 497)
Disposes of a a universal procedure pointer (UPP) to a color-changed callback.

`NewColorChangedUPP` (page 502)
Creates a universal procedure pointer (UPP) to a color-changed callback.

`InvokeColorChangedUPP` (page 501)
Invokes a color-changed callback.

`DisposeColorChangedUPP` (page 497)
Disposes of a a universal procedure pointer (UPP) to a color-changed callback.

Functions

CMY2RGB

Converts a CMY color to its equivalent RGB color.

```
void CMY2RGB (
   const CMYColor *cColor,
   RGBColor *rColor
);
```

Parameters

cColor

A pointer to a `CMYColor` structure to be converted.

rColor

A pointer to an `RGBColor` structure for the converted color.

Availability

Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.

Available in Mac OS X 10.0 and later.

Not available to 64-bit applications.

Declared In

`ColorPicker.h`

DisposeColorChangedUPP

Disposes of a a universal procedure pointer (UPP) to a color-changed callback.

```
void DisposeColorChangedUPP (
   ColorChangedUPP userUPP
);
```

Parameters

userUPP

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X 10.0 and later.

Not available to 64-bit applications.

Declared In

`ColorPicker.h`

DisposeNColorChangedUPP

Disposes of a a universal procedure pointer (UPP) to a color-changed callback.

```
void DisposeNColorChangedUPP (
   NColorChangedUPP userUPP
);
```

Parameters

userUPP

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.

Declared In
`ColorPicker.h`

DisposeUserEventUPP

Disposes of a a universal procedure pointer (UPP) to an event filter callback.

```
void DisposeUserEventUPP (
   UserEventUPP userUPP
);
```

Parameters

userUPP

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`ColorPicker.h`

Fix2SmallFract

Converts a fixed integer to a `SmallFract` value.

```
SmallFract Fix2SmallFract (
   Fixed f
);
```

Parameters

f

The value of type `Fixed` to be converted to a `SmallFract` value.

Return Value
A `SmallFract` converted from the fixed integer value specified in the `f` parameter. See the description of the `SmallFract` data type.

Discussion
A `SmallFract` value can represent a value between 0 and 65,535. Introduced by the original Color Picker Package, `SmallFract` values are used in `CMYColor`, `HSLColor`, and `HSVColor` structures. They can be assigned directly to and from integers.

Availability
Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`ColorPicker.h`

GetColor

Requests the user to choose a color. This function is obsolete; use the `PickColor` function instead.

```
Boolean GetColor (
   Point where,
   ConstStr255Param prompt,
   const RGBColor *inColor,
   RGBColor *outColor
);
```

Parameters

`where`

A point defining the location of the upper-left corner of the dialog box. If you set this parameter to (0,0), the dialog box is centered horizontally on the main screen, with one-third of the empty space above the box and two-thirds below, regardless of the screen size. If you set this parameter to (–1,–1), the `GetColor` function displays the dialog box on the screen supporting the greatest pixel depth.

`prompt`

Text for prompting the user to choose a color. This string is displayed in the upper-left corner of the dialog box.

`inColor`

A pointer to an `RGBColor` structure for a color at entry to the picker. This is the original color, which the user may want for comparison.

`outColor`

A pointer to an `RGBColor` structure describing the new color. This is set to the last color that the user picked before clicking OK. On entry, the `outColor` parameter is treated as undefined, so the output color sample initially matches the input. Although the color being picked may vary widely, the input color sample remains fixed, and clicking the input sample resets the output color sample to match it.

Return Value
`True` if the user clicks the OK button; `false` if the user clicks the Cancel button. In either case, the dialog box is removed.

Discussion
The `GetColor` function does not support ColorSync 1.0 color matching; however, the `PickColor` function does. This function was designed for use for version 1.0 of the Color Picker Package and is still supported for backward compatibility.

Availability
Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.
Available in Mac OS X 10.0 and later.

Declared In
`ColorPicker.h`

HSL2RGB

Converts an HSL color to an RGB color.

```
void HSL2RGB (
   const HSLColor *hColor,
   RGBColor *rColor
);
```

Parameters

hColor

A pointer to the `HSLColor` structure to be converted.

rColor

A pointer to an `RGBColor` structure for the converted color.

Availability
Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`ColorPicker.h`

HSV2RGB

Converts an HSV color to an RGB color.

```
void HSV2RGB (
   const HSVColor *hColor,
   RGBColor *rColor
);
```

Parameters

hColor

A pointer to the `HSVColor` structure to be converted.

rColor

A pointer to an `RGBColor` structure for the converted color.

Availability
Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`ColorPicker.h`

InvokeColorChangedUPP

Invokes a color-changed callback.

```
void InvokeColorChangedUPP (
   SInt32 userData,
   PMColor *newColor,
   ColorChangedUPP userUPP
);
```

Parameters

newColor

userUPP

Discussion

You should not need to use the function `InvokeColorChangedUPP`, as the Color Picker Manager calls your color-changed callback function for you.

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X 10.0 and later.

Not available to 64-bit applications.

Declared In

`ColorPicker.h`

InvokeNColorChangedUPP

Invokes a color-changed callback.

```
void InvokeNColorChangedUPP (
   SRefCon userData,
   NPMColor *newColor,
   NColorChangedUPP userUPP
);
```

Parameters

newColor

userUPP

Discussion

You should not need to use the function `InvokeNColorChangedUPP`, as the Color Picker Manager calls your color-changed callback function for you.

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X 10.0 and later.

Declared In

`ColorPicker.h`

InvokeUserEventUPP

Invokes an event filter callback.

```
Boolean InvokeUserEventUPP (
   EventRecord *event,
   UserEventUPP userUPP
);
```

Parameters

event

userUPP

Discussion

You should not need to use the function `InvokeUserEventUPP`, as the Color Picker Manager calls your event filter callback function for you.

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X 10.0 and later.

Not available to 64-bit applications.

Declared In

`ColorPicker.h`

NewColorChangedUPP

Creates a universal procedure pointer (UPP) to a color-changed callback.

```
ColorChangedUPP NewColorChangedUPP (
   ColorChangedProcPtr userRoutine
);
```

Parameters

userRoutine

Return Value

See the description of the `ColorChangedUPP` data type.

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X 10.0 and later.

Not available to 64-bit applications.

Declared In

`ColorPicker.h`

NewNColorChangedUPP

Creates a universal procedure pointer (UPP) to a color-changed callback.

```
NColorChangedUPP NewNColorChangedUPP (
   NColorChangedProcPtr userRoutine
);
```

Parameters

userRoutine

Return Value
See the description of the `NColorChangedUPP` data type.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.

Declared In
`ColorPicker.h`

NewUserEventUPP

Creates a universal procedure pointer (UPP) to an event filter callback.

```
UserEventUPP NewUserEventUPP (
   UserEventProcPtr userRoutine
);
```

Parameters

userRoutine

Return Value
See the description of the `UserEventUPP` data type.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`ColorPicker.h`

NPickColor

Displays the Color Picker dialog.

```
OSErr NPickColor (
   NColorPickerInfo *theColorInfo
);
```

Parameters

theColorInfo

A pointer to a color picker parameter (`NColorPickerInfo`) data structure. On input, you specify information such as the location of the dialog. Make sure that you set `theColor.profile` field in this strucure to the color space that you want the color returned in. On output, the data structure specifies information such as whether the user changed the color.

Return Value
A result code. See "Color Picker Result Codes" (page 522).

Discussion
The `NPickColor` function displays the standard dialog for color pickers. Use the color picker parameter data structure to specify information to and obtain information from the Color Picker Manager.

Availability
Available in CarbonLib 1.0 and later when ColorPicker 2.1 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

This call is identical to the unsupported function `PickColor` , but it replaces the older ColorSync 1.0 data types with new ColorSync 2.x profile references. When filling out the parameter block for a call to the function `NPickColor`, you must replace all profile handles with profile references. The optional color-changed proc you has also changed; a new data structure `NCMColor` replaces the `CMColor` data type and uses profile references.

Related Sample Code
CarbonSketch

Declared In
`ColorPicker.h`

PickColor

Requests the user to choose a color from the standard color picker dialog box.

```
OSErr PickColor (
   ColorPickerInfo *theColorInfo
);
```

Parameters

`theColorInfo`

A pointer to the `ColorPickerInfo (Old)` (page 510) structure, to specify information to and obtain information from the Color Picker Manager.

Return Value
A result code. See "Color Picker Result Codes" (page 522).

Discussion
The `PickColor` function displays the standard modal dialog box for color pickers. When the user clicks the OK button, `PickColor` removes the dialog box and returns `true` in the `newColorChosen` field and the user's selected color in the `theColor` field of `theColorInfo`. When the user clicks the Cancel button, `PickColor` removes the dialog box and returns `false` in the `newColorChosen` field.

Availability
Available in CarbonLib 1.0 and later when ColorPicker 2.0 or later is present.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`ColorPicker.h`

RGB2CMY

Converts an RGB color to a CMY color.

```
void RGB2CMY (
   const RGBColor *rColor,
   CMYColor *cColor
);
```

Parameters

rColor

A pointer to an `RGBColor` structure to be converted.

cColor

A pointer to a `CMYColor` structure for the converted color.

Availability

Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In

`ColorPicker.h`

RGB2HSL

Converts an RGB color to an HSL color.

```
void RGB2HSL (
   const RGBColor *rColor,
   HSLColor *hColor
);
```

Parameters

rColor

A pointer to the `RGBColor` structure to be converted.

hColor

A pointer to an `HSLColor` structure for the converted color.

Availability

Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In

`ColorPicker.h`

RGB2HSV

Converts an RGB color to an HSV color.

```
void RGB2HSV (
   const RGBColor *rColor,
   HSVColor *hColor
);
```

Parameters

rColor

A pointer to the `RGBColor` structure to be converted.

hColor

A pointer to an `HSVColor` structure for the converted color.

Availability
Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`ColorPicker.h`

SmallFract2Fix

Converts a `SmallFract` value to a fixed integer.

```
Fixed SmallFract2Fix (
   SmallFract s
);
```

Parameters

s

The value of type `SmallFract` to be converted into a fixed integer.

Return Value
A fixed integer converted from the `SmallFract` value specified in the `s` parameter.

Availability
Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`ColorPicker.h`

Callbacks by Task

Changing Colors in a Document

`NColorChangedProcPtr` (page 508)

Defines a pointer to your color-changed callback function, which applies a new color to a user's document.

`ColorChangedProcPtr` (page 507)
Defines a pointer to your color-changed callback function, which applies a new color to a user document.

Handling Application-Directed Events in a Color Picker

`UserEventProcPtr` (page 508)
Defines a pointer to your event filter callback function, which determines whether your application or the Color Picker Manager handles this user event.

Callbacks

ColorChangedProcPtr

Defines a pointer to your color-changed callback function, which applies a new color to a user document.

```
typedef void (*ColorChangedProcPtr) (
    SInt32 userData,
    PMColor *newColor
);
```

If you name your function `MyColorChangedProc`, you would declare it like this:

```
void MyColorChangedProc (
    SInt32 userData,
    PMColor *newColor
);
```

Parameters

userData
Data that your application supplies in the `colorProcData` field of `ColorPickerInfo (Old)` (page 510). Your application can use this value for any purpose it needs.

newColor
A pointer to a `PMColor` (page 516) structure that contains the new color selected by the user. Your color-changed function should update the user's document to use this color.

Discussion

Your application should supply the `colorProc` field of the color picker parameter with a pointer to your color change callback function.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`ColorPicker.h`

NColorChangedProcPtr

Defines a pointer to your color-changed callback function, which applies a new color to a user's document.

```
typedef void (*NColorChangedProcPtr)
(
    SInt32 userData,
    NPMColor *newColor
);
```

If you name your function `MyNColorChangedProc`, you would declare it like this:

```
void MyNColorChangedProc (
    SInt32 userData,
    NPMColor *newColor
);
```

Parameters

newColor

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ColorPicker.h`

UserEventProcPtr

Defines a pointer to your event filter callback function, which determines whether your application or the Color Picker Manager handles this user event.

```
typedef Boolean (*UserEventProcPtr) (
    EventRecord *event
);
```

If you name your function `MyUserEventProc`, you would declare it like this:

```
Boolean MyUserEventProc (
    EventRecord *event
);
```

Parameters

event

A pointer to an event record.

Return Value

`True` if your application handles the event, `false` otherwise. The Color Picker Manager will process the event further only if `false` is returned.

Discussion

Your application should supply the `eventProc` field of the color picker parameter block with a pointer to your filter function. Your filter function should examine the event record passed in the first parameter to determine whether your application needs to handle the event contained in the record.

Applications can generally allow the Color Picker Manager to handle all events that might occur while displaying the standard dialog box. Update events are exceptions to this, however.

The `PickColor` function calls the Dialog Manager function `DialogSelect`. `DialogSelect` does not allow background windows to receive update events; therefore, at a minimum, your event filter function should handle update events. If your application needs to filter or preprocess other events before `DialogSelect` handles them, your application should do so in its event filter function.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ColorPicker.h`

Data Types

CMYColor

Contains cyan, magenta, and yellow values for a color.

```
struct CMYColor {
    SmallFract cyan;
    SmallFract magenta;
    SmallFract yellow;
};
typedef struct CMYColor CMYColor;
```

Fields

`cyan`
The `SmallFract` (page 517) value for the cyan component.

`magenta`
The `SmallFract` value for the magenta component.

`yellow`
The `SmallFract` value for the yellow component.

Discussion
The `CMYColor` structure contains cyan, magenta, and yellow colors. Your application can use a `CMYColor` structure to specify a color in a `PMColor` (page 516) structure. For example, your application supplies a `PMColor` structure in a `ColorPickerInfo (Old)` (page 510) block that it passes to the `PickColor` (page 504) function. Note that CMY and RGB colors are complementary.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ColorPicker.h`

ColorChangedUPP

Defines a universal procedure pointer to a color-changed callback.

```
typedef ColorChangedProcPtr ColorChangedUPP;
```

Discussion
For more information, see the description of the `ColorChangedProcPtr` (page 507) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ColorPicker.h`

ColorPickerInfo (Old)

Contains information needed to display a standard color picker dialog box.

```
struct ColorPickerInfo {
    PMColor theColor;
    CMProfileHandle dstProfile;
    UInt32 flags;
    DialogPlacementSpec placeWhere;
    Point dialogOrigin;
    OSType pickerType;
    UserEventUPP eventProc;
    ColorChangedUPP colorProc;
    UInt32 colorProcData;
    Str255 prompt;
    PickerMenuItemInfo mInfo;
    Boolean newColorChosen;
    SInt8 filler;
};
typedef struct ColorPickerInfo ColorPickerInfo;
```

Fields

`theColor`
A `PMColor` (page 516) structure. The color that your application supplies in this field is passed to the color picker for editing. This becomes the original color for the color picker. After the user clicks either the OK or Cancel button to close the dialog box, this field contains the new color, that is, the color last chosen by the user. Although the new colors selected by the user may vary widely, the original color remains fixed for comparison.

`dstProfile`
A handle to a ColorSync 1.0 profile for the final output device. To use the default system profile, set this field to `NULL`.

`flags`
Bits representing the color picker flags, which are described in detail in "Color Picker Flags" (page 518). Your application can set any of the following flags: `CanModifyPalette`, `CanAnimatePalette`, `AppIsColorSyncAware`. The color picker may set any of the following flags and override your application settings: `InSystemDialog`, `InApplicationDialog`, `InPickerDialog`, `DetachedFromChoices`.

`placeWhere`
: A specification for where to position the dialog box. Your application uses one of the following constants (described in "Dialog Placement Constants" (page 520)) to specify the position for the color picker dialog box: `kAtSpecifiedOrigin`, `kDeepestColorScreen`, `kCenterOnMainScreen`.

`dialogOrigin`
: The point, in global coordinates, at which to locate the upper-left corner of the dialog box. This origin point is used only if your application supplies the `kAtSpecifiedOrigin` specifier in the `placeWhere` field.

`pickerType`
: The component subtype of the initial color picker. If your application sets this field to 0, the default color picker is used (that is, the last color picker chosen by the user). When `PickColor` returns, this field contains the component subtype of the color picker that was chosen when the user closed the color picker dialog box.

`eventProc`
: A pointer to an application-defined event filter function for handling user events meant for your application. If your filter function returns `true`, the Color Picker Manager will not process the event any further. If your filter function returns `false`, the Color Picker Manager handles the event as if it were meant for the color picker. The event filter function you can supply here is described in `UserEventProcPtr` (page 508).

`colorProc`
: A pointer to an application-defined function to handle color changes. This function, described in `ColorChangedProcPtr` (page 507), should support the updating of colors in a document as the user selects them.

`colorProcData`
: A long integer that the Color Picker Manager passes to the application-defined function supplied in the `colorProc` field. Your application-defined function can use this value for any purpose it needs.

`prompt`
: A text string prompting the user to choose a color for a particular use (for example, "Choose a highlight color:").

`mInfo`
: A `PickerMenuItemInfo` (page 515) structure. This structure allows your application to specify your Edit menu for use when a color picker dialog box is displayed.

`newColorChosen`
: `True` if the user chose a color and clicked the OK button, and `false` if the user clicked Cancel.

`filler`

Discussion
When your application calls the `PickColor` (page 504) function to display a standard color picker dialog box, your application uses a color picker parameter block to specify information to and obtain information from the Color Picker Manager. The color picker parameter block is defined by the `ColorPickerInfo` data type.

This version of the Color Picker Manager uses ColorSync 1.0 profiles only. The ColorSync 1.0 profile is a handle-based profile. The profile format is defined by Apple Computer. You cannot use version 2.0 profiles, which are identified by profile references, with this version of the Color Picker Manager. ColorSync 1.0 profiles typically reside in the ColorSyncTM Profiles folder (within the Preferences folder of the System Folder). They may also be embedded with the images to which they pertain in graphics files.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
ColorPicker.h

HSLColor

Contains hue, saturation and lightness information for a color.

```
struct HSLColor {
    SmallFract hue;
    SmallFract saturation;
    SmallFract lightness;
};
typedef struct HSLColor HSLColor;
```

Fields

`hue`
The `SmallFract` (page 517) value for the hue.

`saturation`
The `SmallFract` value for the saturation, where 1 is full color.

`lightness`
The `SmallFract` value for lightness, where 1 is full white.

Discussion
The `HSLColor` structure contains a color's hue, saturation, and lightness values. Your application can use an `HSLColor` structure to specify a color in a `PMColor` (page 516) structure. For example, your application supplies a `PMColor` structure in a `ColorPickerInfo (Old)` (page 510) block that it passes to the `PickColor` (page 504) function. Note that the standard HLS order is altered to HSL.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
ColorPicker.h

HSVColor

Contains hue, saturation and value information for a color.

```
struct HSVColor {
    SmallFract hue;
    SmallFract saturation;
    SmallFract value;
};
typedef struct HSVColor HSVColor;
```

Fields

`hue`
The `SmallFract` (page 517) value for the hue.

`saturation`
The `SmallFract` value for the saturation, where 1 is full color.

`value`
The `SmallFract` value for the color's value, where 1 is maximum intensity.

Discussion
The `HSVColor` structure contains the hue, saturation, and value of a color. Your application can use an `HSVColor` structure to specify a color in a `PMColor` (page 516) structure. For example, your application supplies a `PMColor` structure in a `ColorPickerInfo (Old)` (page 510) block that it passes to the `PickColor` (page 504) function.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ColorPicker.h`

NColorChangedUPP

Defines a universal procedure pointer to your color-changed callback.

```
typedef NColorChangedProcPtr NColorChangedUPP;
```

Discussion
For more information, see the description of the `NColorChangedProcPtr` (page 508) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ColorPicker.h`

NColorPickerInfo

Contains information needed to display a standard color picker dialog.

```
struct NColorPickerInfo {
    NPMColor theColor;
    CMProfileRef dstProfile;
    UInt32 flags;
    DialogPlacementSpec placeWhere;
    Point dialogOrigin;
    OSType pickerType;
    UserEventUPP eventProc;
    NColorChangedUPP colorProc;
    UInt32 colorProcData;
    Str255 prompt;
    PickerMenuItemInfo mInfo;
    Boolean newColorChosen;
    UInt8 reserved;
};
typedef struct NColorPickerInfo NColorPickerInfo;
```

Fields

`theColor`
: An `NPMColor` structure that contains a color profile and a color.

`dstProfile`
: The destination profile.

`flags`
: The options to apply when displaying the color picker dialog. For information on the flags that you can supply, see "Color Picker Flags" (page 518).

`dialogOrigin`
: A constant that specifies where on the screen to place the color picker dialog. The constant `kAtSpecifiedOrigin` specifies to place the top-left corner of the color picker dialog at the point specified in the `dialogOrigin` field of the color picker parameter block. The constant `kDeepestColorScreen` specifies to center the color picker dialog on the screen with the greatest color depth. The constant `kCenterOnMainScreen` specifies to center the color picker dialog on the main screen.

`pickerType`
: The type of color picker.

`eventProc`
: A a universal procedure pointer to an event-filter callback.

`colorProc`
: A universal procedure pointer to a color-changed callback.

`colorProcData`
: Data needed for your color-changed callback.

`prompt`
: A string to use as a prompt.

`mInfo`
: A structure that contains information needed for your application to specify an Edit menu for use when a color picker dialog box is displayed.

`newColorChosen`
: A Boolean value that specifies whether or not a new color was chosen.

`reserved`
: Reserved for future use.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ColorPicker.h`

NPMColor

Contains a color profile and a color.

```
struct NPMColor {
    CMProfileRef profile;
    CMColor color;
};
typedef struct NPMColor NPMColor;
```

Fields

`profile`
A color-matching profile.

`color`
A color, as specified in a color-matching structure.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ColorPicker.h`

NPMColorPtr

A pointer to an `NPMColor` data structure.

```
typedef NPMColor * NPMColorPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ColorPicker.h`

PickerMenuItemInfo

Contains information needed for your application to specify an Edit menu for use when a color picker dialog box is displayed.

```
struct PickerMenuItemInfo {
    short editMenuID;
    short cutItem;
    short copyItem;
    short pasteItem;
    short clearItem;
    short undoItem;
};
typedef struct PickerMenuItemInfo PickerMenuItemInfo;
```

Fields

`editMenuID`
The resource ID of the Edit menu for the color picker.

`cutItem`
The item number of the Cut command.

`copyItem`
The item number of the Copy command.

`pasteItem`
The item number of the Paste command.

`clearItem`
The item number of the Clear command.

`undoItem`
The item number of the Undo command.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ColorPicker.h`

PMColor

Contains a color profile and a color.

```
 {
    CMProfileHandle profile;
    CMColor color;
};
typedef struct PMColor PMColor;
```

Fields

`profile`
A handle to a color-matching profile (`CMProfile` structure). If your application sets this field to `NULL`, then the Color Picker Manager uses the default system profile.

`color`
A color, as specified in a color-matching structure. (`CMColor` is a union data type defined in the ColorSync programming interface.

Discussion
For defining colors, version 2.0 of the Color Picker Manager uses a picker color structure. For example, when your application creates a `ColorPickerInfo` parameter block to pass to `PickColor` (page 504) , your application supplies a picker color structure. The color that your application supplies in this field is passed to the color picker for editing. After the user clicks either the OK or Cancel button to close the dialog box, this field contains the color last chosen by the user.

The picker color structure includes a ColorSync 1.0 profile, a structure that matches colors among hardware devices such as displays, printers, and scanners. This color-matching profile (a data structure of type `CMProfile`) defines the color space of the color (which includes the type of color—RGB, CMYK, HSL, and so on). Using the `dstProfile` field of `ColorPickerInfo (Old)` (page 510) or the PickerSetProfile function, your application can specify a destination color-matching profile, which describes the color space of the device for which the color is being chosen. Given information about the destination profile, color pickers that are ColorSync aware can help the user choose a color that's within the destination device's gamut.

This version of the Color Picker Manager uses ColorSync 1.0 profiles only. The ColorSync 1.0 profile is a handle-based profile. The profile format is defined by Apple Computer. You cannot use version 2.0 profiles, which are identified by profile references, with this version of the Color Picker Manager. ColorSync 1.0 profiles typically reside in the ColorSyncTM Profiles folder (within the Preferences folder of the System Folder). They may also be embedded with the images to which they pertain in graphics files.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ColorPicker.h`

PMColorPtr

A pointer to a `PMColor` data structure.

```
typedef PMColor * PMColorPtr;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ColorPicker.h`

SmallFract

Defines a data type for an unsigned short value.

```
typedef  unsigned short SmallFract;
```

Discussion
The `SmallFract` type is derived from the low-order word of a fixed integer. The Color Picker Manager uses `SmallFract` values to save memory and to be compatible with the Color QuickDraw `RGBColor` structure. You can use the `Fix2SmallFract` (page 498) function to convert a fixed integer to a `SmallFract` value. You can use the `SmallFract2Fix` (page 506) function to convert a `SmallFract` value to a fixed integer.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ColorPicker.h`

UserEventUPP

Defines a universal procedure pointer to an event-filter callback.

```
typedef UserEventProcPtr UserEventUPP;
```

Discussion
For more information, see the description of the `UserEventProcPtr` (page 508) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ColorPicker.h`

Constants

Color Picker Flags

Specify a variety of options to apply when displaying the color picker dialog.

```
enum {
    kColorPickerDialogIsMoveable = 1,
    kColorPickerDialogIsModal = 2,
    kColorPickerCanModifyPalette = 4,
    kColorPickerCanAnimatePalette = 8,
    kColorPickerAppIsColorSyncAware = 16,
    kColorPickerInSystemDialog = 32,
    kColorPickerInApplicationDialog = 64,
    kColorPickerInPickerDialog = 128,
    kColorPickerDetachedFromChoices = 256,
    kColorPickerCallColorProcLive = 512
};
```

Constants
`kColorPickerDialogIsMoveable`
If your application sets the bit represented by this constant when creating a custom dialog box, then the color picker dialog box is moveable by the user.

Available in Mac OS X v10.0 and later.

Declared in `ColorPicker.h`.

`kColorPickerDialogIsModal`

If your application sets the bit represented by this constant when creating a custom dialog box, then the color picker dialog box is a modal dialog box.

Available in Mac OS X v10.0 and later.

Declared in `ColorPicker.h`.

`kColorPickerCanModifyPalette`

Your application should set the bit represented by this constant if your application can install a palette of its own that may modify (but not animate) the current color table. If you do not want the colors in the document to change as the user makes choices in the color picker dialog box, do not set this flag.

Available in Mac OS X v10.0 and later.

Declared in `ColorPicker.h`.

`kColorPickerCanAnimatePalette`

If your application sets the bit represented by this constant, then the color picker may modify or animate the palette.

Available in Mac OS X v10.0 and later.

Declared in `ColorPicker.h`.

`kColorPickerAppIsColorSyncAware`

Your application should set the bit represented by this constant if your application uses ColorSync color matching. If your application sets this bit, a color may be returned to your application in a different color space than the one initially passed to the `PickColor` function. For example, your application could pass an RGB color with no color-matching profile in the field `theColor` in `ColorPickerInfo`, and the Color Picker Manager could return a CMYK color with its associated profile. If your application does not set this flag, the Color Picker Manager automatically converts any color it receives back from the color picker to an RGB color.

This version of the Color Picker Manager uses ColorSync 1.0 profiles only. The ColorSync 1.0 profile is a handle-based profile. The profile format is defined by Apple Computer. You cannot use version 2.0 profiles, which are identified by profile references, with this version of the Color Picker Manager. ColorSync 1.0 profiles typically reside in the ColorSyncTM Profiles folder (within the Preferences folder of the System Folder). They may also be embedded with the images to which they pertain in graphics files.

Available in Mac OS X v10.0 and later.

Declared in `ColorPicker.h`.

`kColorPickerInSystemDialog`

The color picker sets this flag to indicate that the color picker is in a system-owned dialog box.

Available in Mac OS X v10.0 and later.

Declared in `ColorPicker.h`.

`kColorPickerInApplicationDialog`

The color picker sets this flag to indicate that the color picker is in an application-owned dialog box.

Available in Mac OS X v10.0 and later.

Declared in `ColorPicker.h`.

`kColorPickerInPickerDialog`

The color picker sets this flag to indicate that the color picker is in its own dialog box.

Available in Mac OS X v10.0 and later.

Declared in `ColorPicker.h`.

`kColorPickerDetachedFromChoices`
The color picker sets this flag to indicate that the color picker has been detached from the choices list.

Available in Mac OS X v10.0 and later.

Declared in `ColorPicker.h`.

`kColorPickerCallColorProcLive`
Available in Mac OS X v10.0 and later.

Declared in `ColorPicker.h`.

Discussion
In the `flags` field of the `ColorPickerInfo (Old)` (page 510) parameter block your application specifies characteristics for the color picker dialog box.

The color picker may set any of the `InSystemDialog`, `InApplicationDialog`, `InPickerDialog`, or `DetachedFromChoices` flags and override your application settings.

Special Considerations

Dialog Placement Constants

Specify where on the screen to place the color picker dialog.

```
typedef SInt16 DialogPlacementSpec;
enum {
    kAtSpecifiedOrigin = 0,
    kDeepestColorScreen = 1,
    kCenterOnMainScreen = 2
};
```

Constants
`kAtSpecifiedOrigin`
Specify to place the top-left corner of the color picker dialog at the point specified in the `dialogOrigin` field of the color picker parameter block.

Available in Mac OS X v10.0 and later.

Declared in `ColorPicker.h`.

`kDeepestColorScreen`
Specify to center the color picker dialog on the screen with the greatest color depth.

Available in Mac OS X v10.0 and later.

Declared in `ColorPicker.h`.

`kCenterOnMainScreen`
Specify to center the color picker dialog on the main screen.

Available in Mac OS X v10.0 and later.

Declared in `ColorPicker.h`.

Discussion
In the `placeWhere` field of the `ColorPickerInfo (Old)` (page 510) , your application specifies where to place the color picker dialog box. Your application uses the `DialogPlacementSpec` enumeration to specify the position of the color picker dialog box.

Maximum Small Fraction

Defines the maximum value for the `SmallFract` data type.

```
enum {
    kMaximumSmallFract = 0x0000FFFF
};
```

Width and Height Constants

Specify the width and height of the color picker dialog.

```
enum {
    kDefaultColorPickerWidth = 383,
    kDefaultColorPickerHeight = 238
};
```

Constants

`kDefaultColorPickerWidth`
Specifies the width of the color picker dialog.
Available in Mac OS X v10.0 and later.
Declared in `ColorPicker.h`.

`kDefaultColorPickerHeight`
Specifies the height of the color picker dialog.
Available in Mac OS X v10.0 and later.
Declared in `ColorPicker.h`.

Old Maximum Small Fraction

Defines the maximum value for the `SmallFract` data type.

```
enum {
    MaxSmallFract = 0x0000FFFF
};
```

Constants

`MaxSmallFract`
The maximum value of the `SmallFract` data type, as a long integer.
Available in Mac OS X v10.0 and later.
Declared in `ColorPicker.h`.

Old Color Picker Flags

Specify a variety of options to apply when displaying the color picker dialog.

```
enum {
    DialogIsMoveable = 1,
    DialogIsModal = 2,
    CanModifyPalette = 4,
    CanAnimatePalette = 8,
    AppIsColorSyncAware = 16,
    InSystemDialog = 32,
    InApplicationDialog = 64,
    InPickerDialog = 128,
    DetachedFromChoices = 256,
    CallColorProcLive = 512
};
```

Result Codes

The most common result codes returned by Color Picker are listed below.

Result Code	Value	Description
`firstPickerError`	-4000	Available in Mac OS X v10.0 and later.
`invalidPickerType`	-4000	Available in Mac OS X v10.0 and later.
`requiredFlagsDontMatch`	-4001	Available in Mac OS X v10.0 and later.
`pickerResourceError`	-4002	Available in Mac OS X v10.0 and later.
`cantLoadPicker`	-4003	Available in Mac OS X v10.0 and later.
`cantCreatePickerWindow`	-4004	Available in Mac OS X v10.0 and later.
`cantLoadPackage`	-4005	Available in Mac OS X v10.0 and later.
`pickerCantLive`	-4006	Available in Mac OS X v10.0 and later.
`colorSyncNotInstalled`	-4007	Available in Mac OS X v10.0 and later.
`badProfileError`	-4008	Available in Mac OS X v10.0 and later.
`noHelpForItem`	-4009	Available in Mac OS X v10.0 and later.

Control Manager Reference

Framework: Carbon/Carbon.h

Declared in Controls.h
ControlDefinitions.h

Overview

Your application uses the Control Manager to create and manage controls. Controls are onscreen objects that the user can manipulate with the mouse. By manipulating controls, the user can take an immediate action or change settings to modify a future action. For example, a scroll bar control allows a user to immediately change the portion of the document that your application displays, whereas a pop-up menu control for connection speed might allow the user to change the rate by which your application handles subsequent data transmissions.

Virtually all applications need to implement controls, at least in the form of scroll bars for document windows. Other standard Mac OS controls include buttons, checkboxes, radio buttons, and pop-up menus. You can use the Control Manager to create and manage these controls, too.

In Mac OS X v10.2 and later, Control Manager controls may be implemented as HIViews. View-based controls offer additional flexibility and extensibility for developers. For more information, see the document *HIView Programming Guide*.

Important: Documentation for the data browser control is available separately in *Data Browser Programming Guide*.

Carbon supports most Control Manager functions, with the following changes:

- The C-style, lowercase versions of Control Manager function names are no longer supported. If your application uses any Control Manager lowercase function names, you must replace them with their uppercase equivalents.
- Custom control definition procedures (also known as CDEFs), must be compiled as PowerPC-native code, and can not be stored as resources. See the Carbon Porting Guide for more information.
- Your application must use the functions defined by the Control Manager to create and dispose of Control Manager data structures. For example, instead of directly creating and disposing of control records, applications must call the Control Manager functions `GetNewControl` and `DisposeControl`.
- With the availability of the Appearance Manager, you should not access the `PopupPrivateData` structure, but rather pass the `kControlPopupButtonMenuHandleTag` tag to `GetControlData` to obtain its contents.
- Your application must use Control Manager accessor functions to access Control Manager data structures.

- You are encouraged to adopt the standard Mac OS control definition procedures in your application. Applications that use the standard control definition procedures inherit the Mac OS human interface appearance. Applications that use custom control definition procedures will work correctly, but because custom definition procedures invoke their own drawing routines, Mac OS X can't draw these applications with the current appearance.

Functions by Task

Creating and Removing Controls

`CreateBevelButtonControl` (page 539)
Creates a bevel button control.

`CreateChasingArrowsControl` (page 540)
Creates a chasing arrows control.

`CreateCheckBoxControl` (page 541)
Creates a checkbox control.

`CreateCheckGroupBoxControl` (page 542)
Creates a group box control that has a check box as its title.

`CreateClockControl` (page 542)
Creates a clock control.

`CreateDisclosureButtonControl` (page 544)
Creates a new instance of the Disclosure Button Control.

`CreateDisclosureTriangleControl` (page 545)
Creates a disclosure triangle control.

`CreateEditUnicodeTextControl` (page 547)
Creates a new edit text control.

`CreateGroupBoxControl` (page 548)
Creates a group box control.

`CreateIconControl` (page 549)
Creates an icon control.

`CreateImageWellControl` (page 549)
Creates an image well control.

`CreateListBoxControl` (page 550)
Creates a list box control.

`CreateLittleArrowsControl` (page 552)
Creates a little arrows control.

`CreatePictureControl` (page 553)
Creates a picture control.

`CreatePlacardControl` (page 554)
Creates a placard control.

`CreatePopupArrowControl` (page 554)
Creates a pop-up arrow control.

`CreatePopupButtonControl` (page 555)
Creates a pop-up button control.

`CreatePopupGroupBoxControl` (page 556)
Creates a group box control that has a pop-up button as its title.

`CreateProgressBarControl` (page 558)
Creates a progress bar control.

`CreatePushButtonControl` (page 558)
Creates a push button control.

`CreatePushButtonWithIconControl` (page 559)
Creates a push button control containing an icon or other graphical content.

`CreateRadioButtonControl` (page 560)
Creates a radio button control.

`CreateRadioGroupControl` (page 561)
Creates a radio group control.

`CreateRelevanceBarControl` (page 561)
Creates a relevance bar control.

`CreateRoundButtonControl` (page 564)
Creates a new instance of the round button control.

`CreateScrollBarControl` (page 564)
Creates a scroll bar control.

`CreateSeparatorControl` (page 567)
Creates a separator control.

`CreateSliderControl` (page 567)
Creates a slider control.

`CreateStaticTextControl` (page 569)
Creates a new static text control.

`CreateTabsControl` (page 569)
Creates a tabs control.

`CreateUserPaneControl` (page 571)
Creates a user pane control.

`CreateWindowHeaderControl` (page 571)
Creates a window header control.

`DisposeControl` (page 573)
Decrements a control's reference count and destroys it if the reference count becomes 0.

`GetNewControl` (page 610)
Creates a control from a control resource.

`KillControls` (page 632)
Removes all of the controls from a window that you wish to keep.

`RegisterControlDefinition` (page 640)
Registers an old-style control definition.

`CreateCustomControl` (page 543) Deprecated in Mac OS X v10.5
Creates a custom control. (Deprecated. Register your custom subclass of the HIView class and create an instance of your class using `HIObjectCreate`.)

Embedding Controls

`AutoEmbedControl` (page 536)
Automatically embeds a control in the smallest appropriate embedder control.

`CountSubControls` (page 538)
Obtains the number of embedded controls within a control.

`CreateRootControl` (page 562)
Creates the root control for a specified window.

`DumpControlHierarchy` (page 582)
Writes a textual representation of the control hierarchy for a specified window into a file.

`EmbedControl` (page 583)
Embeds one control inside another.

`GetControlOwner` (page 600)
Returns the window to which a control is bound.

`GetIndexedSubControl` (page 608)
Obtains a handle to a specified embedded control.

`GetRootControl` (page 611)
Obtains a handle to a window's root control.

`GetSuperControl` (page 611)
Obtains a handle to an embedder control.

Displaying Controls

`DisableControl` (page 573)
Disables a control.

`Draw1Control` (page 580)
Draws a control and any embedded controls that are currently visible in the specified window.

`DrawControlInCurrentPort` (page 581)
Draws a control in the current graphics port.

`DrawControls` (page 581)
Draws all controls currently visible in the specified window.

`GetControlViewSize` (page 607)
Obtains the size of the content to which a control's size is proportioned.

`IsControlVisible` (page 631)
Determines whether a control is visible.

`SetControlViewSize` (page 662)
Informs the Control Manager of the size of the content to which a control's size is proportioned.

`SetUpControlBackground` (page 667)
Applies the proper background color for the given control to the current port.

`SetUpControlTextColor` (page 668)
Applies the proper text color for the given control to the current port.¬¬

`UpdateControls` (page 672)
Draws controls in the specified update region of a window.

HideControl (page 618)
Makes a control, and any latent embedded controls, invisible.

ShowControl (page 669)
Makes an invisible control, and any latent embedded controls, visible.

SetControlVisibility (page 663)
Sets the visibility of a control, and any embedded controls, and specifies whether it should be drawn.

Handling Events in Controls

FindControl (page 585)
Obtains the location of a mouse-down event in a control.

FindControlUnderMouse (page 586)
Obtains the location of a mouse-down event in a control.

GetControlAction (page 591)
Returns a pointer to the action function associated with a control structure.

GetControlClickActivation (page 593)
Gets the control's preferred behavior for responding to particular click.

GetControlCommandID (page 594)
Gets the command ID for a control.

HandleControlClick (page 613)
Responds to cursor movements in a control while the mouse button is down and returns the location of the next mouse-up event.

HandleControlContextualMenuClick (page 614)
Allows a control to display a contextual menu.

HandleControlDragReceive (page 615)
Tells a control to accept the data from a drag.

HandleControlDragTracking (page 615)
Tells a control to respond visually to a drag.

HandleControlSetCursor (page 617)
Requests that a control set the cursor based on the mouse location.

IsAutomaticControlDragTrackingEnabledForWindow (page 629)
Indicates whether automatic drag tracking is enabled for the specified window.

IsControlDragTrackingEnabled (page 630)
Indicates whether a control's drag track and receive support is enabled.

SetAutomaticControlDragTrackingEnabledForWindow (page 643)
Enables or disables automatic drag tracking for a window.

SetControlAction (page 649)
Sets the action function for a control.

SetControlCommandID (page 651)
Sets the command ID for a control.

SetControlDragTrackingEnabled (page 653)
Sets the drag tracking state for a control.

TestControl (page 670)
Obtains the control part in which a mouse-down event occurred.

`HandleControlKey` (page 616) Deprecated in Mac OS X v10.5
Sends a keyboard event to a control with keyboard focus. (Deprecated. For HIView-based controls, send the view a `kEventTextInputUnicodeForKeyEvent` event.)

Manipulating Controls

`ActivateControl` (page 535)
Activates a control and any latent embedded controls.

`DeactivateControl` (page 572)
Deactivates a control and any latent embedded controls.

`EnableControl` (page 584)
Enables a control.

`GetControlRegion` (page 605)
Obtains the region corresponding to a given control part.

`IsControlActive` (page 629)
Returns whether a control is active.

`IsControlEnabled` (page 631)
Indicates whether a control is enabled.

`IsControlHilited` (page 631)
Indicates whether or not the control is highlighted.

`MoveControl` (page 633)
Moves a control within its window.

`SizeControl` (page 670)
Changes the size of a control's rectangle.

Handling Keyboard Focus

`AdvanceKeyboardFocus` (page 535)
Advances the keyboard focus to the next focusable control in a window.

`ClearKeyboardFocus` (page 537)
Removes the keyboard focus for the currently focused control in a window.

`GetKeyboardFocus` (page 609)
Obtains a handle to the control with the current keyboard focus for a specified window.

`ReverseKeyboardFocus` (page 642)
Returns keyboard focus to the prior focusable control in a window.

`SetKeyboardFocus` (page 665)
Sets the current keyboard focus to a specified control part for a window.

Accessing Control Settings and Data

`ChangeControlPropertyAttributes` (page 537)
Changes a property attribute.

`CopyControlTitleAsCFString` (page 538)
Makes a copy of the control's title as a Core Foundation string.

`GetBestControlRect` (page 587)
Obtains a control's optimal size and text placement.

`GetControl32BitMaximum` (page 589)
Obtains the maximum setting of a control.

`GetControl32BitMinimum` (page 590)
Obtains the minimum setting of a control.

`GetControl32BitValue` (page 590)
Obtains the current setting of a control.

`GetControlBounds` (page 592)
Gets the bounds of a control.

`GetControlByID` (page 592)
Finds a control in a window by its unique ID.

`GetControlData` (page 594)
Obtains control-specific data.

`GetControlDataSize` (page 596)
Obtains the size of a control's tagged data.

`GetControlHilite` (page 597)
Gets the highlight status of a control.

`GetControlID` (page 598)
Gets the control ID for a control.

`GetControlKind` (page 598)
Returns the kind of the given control.

`GetControlProperty` (page 602)
Obtains a piece of data that has been previously associated with a control.

`GetControlPropertySize` (page 603)
Obtains the size of a piece of data that has previously been associated with a control.

`GetControlReference` (page 604)
Obtains a control's current reference value.

`GetImageWellContentInfo` (page 608)
Gets information about the content of an image well.

`GetControlPropertyAttributes` (page 603)
Gets the property attributes for a control.

`GetTabContentRect` (page 612)
Gets the content rectangle for a tab.

`IsValidControlHandle` (page 632)
Reports whether a given handle is a control handle.

`RemoveControlProperty` (page 641)
Removes a piece of data that has been previously associated with a control.

`SetControl32BitMaximum` (page 647)
Changes the maximum setting of a control and, if appropriate, redraws it accordingly.

`SetControl32BitMinimum` (page 648)
Changes the minimum setting of a control and, if appropriate, redraws it accordingly.

SetControl32BitValue (page 648)
Changes the current setting of a control and redraws it accordingly.

SetControlData (page 652)
Sets control-specific data.

SetControlBounds (page 650)
Sets the bounds of a control.

SetControlID (page 655)
Sets a control's ID.

SetControlTitleWithCFString (page 661)
Sets the title for a control to the specified Core Foundation string.

SetDisclosureTriangleLastValue (page 664)
Sets the last value of a disclosure triangle.

SetImageWellContentInfo (page 664)
Sets the content information for an image well.

SetImageWellTransform (page 665)
Sets an image well transform.

SetTabEnabled (page 666)
Enables and disables a tab control.

SetControlFontStyle (page 654)
Sets the font style for a control.

SetControlProperty (page 658)
Associates data with a control.

SetControlReference (page 659)
Changes a control's current reference value.

Manipulating Menus in Controls

The functions described in this section can only be called for pop-up button and pop-up group box controls, which can support pop-up menus that activate when the user presses the control with the mouse.

GetControlPopupMenuHandle (page 601)
Gets the menu handle for a pop-up control.

GetControlPopupMenuID (page 601)
Gets the menu ID of a pop-up menu.

SetControlPopupMenuHandle (page 657)
Sets the menu handle for a pop-up control.

SetControlPopupMenuID (page 657)
Sets the menu ID for a pop-up control

Manipulating Bevel Buttons

Bevel button controls have additional features that you can or should manipulate to display them properly. This section describes the functions you can use to manipulate these features.

`GetBevelButtonContentInfo` (page 588)
Gets the content information for a bevel button.

`GetBevelButtonMenuHandle` (page 588)
Gets the menu handle for a bevel button.

`GetBevelButtonMenuValue` (page 589)
Gets the value of a bevel button menu.

`SetBevelButtonContentInfo` (page 644)
Sets the content information for a bevel button.

`SetBevelButtonGraphicAlignment` (page 644)
Sets the alignment for a bevel button.

`SetBevelButtonMenuValue` (page 645)
Sets the value of a bevel button menu.

`SetBevelButtonTextAlignment` (page 645)
Sets the alignment of the text for a bevel button.

`SetBevelButtonTextPlacement` (page 646)
Sets the placement for bevel button text.

`SetBevelButtonTransform` (page 647)
Sets the transform for a bevel button.

Managing Control UPPs

`DisposeControlActionUPP` (page 574)
Disposes of a control action UPP.

`DisposeControlCNTLToCollectionUPP` (page 574)
Disposes of a CNLT to collection UPP.

`DisposeControlEditTextValidationUPP` (page 576)
Disposes of an edit text validation UPP.

`DisposeControlKeyFilterUPP` (page 576)
Disposes of a key filter UPP.

`DisposeControlUserPaneActivateUPP` (page 576)
Disposes of a user pane activate UPP.

`DisposeControlUserPaneBackgroundUPP` (page 577)
Disposes of a user pane background UPP.

`DisposeControlUserPaneDrawUPP` (page 577)
Disposes of a user pane draw UPP.

`DisposeControlUserPaneFocusUPP` (page 577)
Disposes of a user pane focus UPP.

`DisposeControlUserPaneHitTestUPP` (page 578)
Disposes of a user pane hit test UPP.

`DisposeControlUserPaneIdleUPP` (page 578)
Disposes of a user pane idle UPP.

`DisposeControlUserPaneKeyDownUPP` (page 578)
Disposes of a user pane key down UPP.

DisposeControlUserPaneTrackingUPP (page 579)
Disposes of a user pane tracking UPP.

DisposeEditUnicodePostUpdateUPP (page 579)
Disposes of an edit unicode post update UPP.

InvokeControlActionUPP (page 620)
Invokes a control action UPP.

InvokeControlCNTLToCollectionUPP (page 621)
Invokes a control-to-collection UPP.

InvokeControlEditTextValidationUPP (page 623)
Invokes a control edit text validation UPP.

InvokeControlKeyFilterUPP (page 624)
Invokes a control key filter UPP.

InvokeControlUserPaneActivateUPP (page 624)
Invokes a control user pane activate UPP.

InvokeControlUserPaneBackgroundUPP (page 625)
Invokes a user pane background UPP.

InvokeControlUserPaneDrawUPP (page 625)
Invokes a user pane draw UPP.

InvokeControlUserPaneFocusUPP (page 626)
Invokes a user pane focus UPP.

InvokeControlUserPaneHitTestUPP (page 626)
Invokes a user pane hit test UPP.

InvokeControlUserPaneIdleUPP (page 627)
Invokes a user pane idle UPP.

InvokeControlUserPaneKeyDownUPP (page 627)
Invokes a user pane key down UPP.

InvokeControlUserPaneTrackingUPP (page 628)
Invokes a user pane tracking UPP.

InvokeEditUnicodePostUpdateUPP (page 628)
Invokes a Unicode post update UPP.

NewControlActionUPP (page 636)
Creates a UPP for a control action callback function.

NewControlCNTLToCollectionUPP (page 636)
Creates a UPP for a control-to-collection callback function.

NewControlEditTextValidationUPP (page 637)
Creates a UPP for a control edit text validation callback function.

NewControlKeyFilterUPP (page 637)

NewControlUserPaneActivateUPP (page 638)

NewControlUserPaneBackgroundUPP (page 638)

NewControlUserPaneDrawUPP (page 638)

NewControlUserPaneFocusUPP (page 638)

NewControlUserPaneHitTestUPP (page 639)

NewControlUserPaneIdleUPP (page 639)

NewControlUserPaneKeyDownUPP (page 639)

NewControlUserPaneTrackingUPP (page 640)

NewEditUnicodePostUpdateUPP (page 640)

DisposeControlDefUPP (page 575) Deprecated in Mac OS X v10.5
: Disposes of a control definition UPP. (Deprecated. Use a custom HIView to draw a custom control.)

InvokeControlDefUPP (page 622) Deprecated in Mac OS X v10.5
: Invokes a control definition UPP. (Deprecated. Use a custom HIView to draw a custom control.)

NewControlDefUPP (page 637) Deprecated in Mac OS X v10.5
: Creates a UPP for a control definition callback function. (Deprecated. Use a custom HIView to draw a custom control.)

Obsolete Functions

These functions are outdated and are not recommended.

CreateScrollingTextBoxControl (page 566)
: Creates a scrolling text box control.

DisposeControlColorUPP (page 575)

GetControlFeatures (page 597)
: Obtains the features a control supports.

GetControlMaximum (page 599)
: Obtains a control's maximum setting. (Deprecated. Use GetControl32BitMaximum (page 589) instead.)

GetControlMinimum (page 600)
: Obtains a control's minimum setting. (Deprecated. Use GetControl32BitMinimum (page 590) instead.)

GetControlValue (page 606)
: Obtains a control's current setting. (Deprecated. Use GetControl32BitValue (page 590) instead.)

GetControlVariant (page 607)
: Returns the variation code specified in the control definition function for a particular control. (Deprecated. Use custom HIViews instead of custom CDEFs. See *HIView Programming Guide*.)

InvokeControlColorUPP (page 622)

NewControlColorUPP (page 636)

`SetControlColorProc` (page 650)
Associates a `ControlColorUPP` with a given Control, thereby allowing you to bypass the embedding hierarchy-based color setup of `SetUpControlBackground`/`SetUpControlTextColor` and replace it with your own.

`GetControlDataHandle` (page 595)
Obtains a handle to control-specific data. (Deprecated. Use custom HIViews instead of custom CDEFs. See *HIView Programming Guide*.)

`SetControlDataHandle` (page 653)
(Deprecated. Use custom HIViews instead of custom CDEFs. See *HIView Programming Guide*.)

`SetControlMaximum` (page 655)
Changes the maximum setting of a control and redraws its indicator or scroll box accordingly. (Deprecated. Use `SetControl32BitMaximum` (page 647) instead.)

`SetControlMinimum` (page 656)
Changes the minimum setting of a control and redraws its indicator or scroll box accordingly. (Deprecated. Use `SetControl32BitMinimum` (page 648) instead.)

`SetControlSupervisor` (page 659)
Routes mouse-down events to the embedder control.

`SetControlValue` (page 661)
Changes the current setting of a control and redraws it accordingly. (Deprecated. Use `SetControl32BitValue` (page 648) instead.)

`TrackControl` (page 671)
Responds to cursor movements in a control while the mouse button is down. (Deprecated. Use `HandleControlClick` (page 613) instead.)

`DragControl` (page 579)
Draws and moves an outline of a control or its indicator while the user drags it. (Deprecated. Use Drag Manager functions if you want drag-and-drop support for controls. See *Drag Manager Reference*.)

`HiliteControl` (page 619)
Changes the highlighting of a control.

`SendControlMessage` (page 642)
Sends a message to a control definition function. (Deprecated. For custom controls, use a custom HIView instead of a control definition function. See *HIView Programming Guide*.)

`GetControlTitle` (page 605) Deprecated in Mac OS X v10.5
Obtains the title of a control. (Deprecated. Use `HIViewCopyText` (page 2448) or `CopyControlTitleAsCFString` (page 538) instead.)

`NewControl` (page 634) Deprecated in Mac OS X v10.5
Creates a control based on parameter data. (Deprecated. Use the specific control creation function instead (for example, `CreateCheckBoxControl` (page 541)).)

`SetControlTitle` (page 660) Deprecated in Mac OS X v10.5
Changes the title of a control and redraws the control accordingly. (Deprecated. Use `HIViewSetText` (page 2488) or `SetControlTitleWithCFString` (page 661) instead.)

`CreateEditTextControl` (page 546) Deprecated in Mac OS X v10.4
Creates a new edit text control. (Deprecated. Use `CreateEditUnicodeTextControl` (page 547) instead.)

`IdleControls` (page 620) Deprecated in Mac OS X v10.4
Performs idle event processing. (Deprecated. You should remove all calls to `IdleControls` because it uses unnecessary processor time. System-supplied controls do not respond to `IdleControls` in Mac OS X.)

Functions

ActivateControl

Activates a control and any latent embedded controls.

```
OSErr ActivateControl (
   ControlRef inControl
);
```

Parameters

inControl

A handle to the control to activate. If you pass a window's root control, `ActivateControl` activates all controls in that window. For a description of this data type, see ControlRef (page 709).

Return Value

A result code. See "Control Manager Result Codes" (page 824).

Discussion

The `ActivateControl` function should be called instead of `HiliteControl` to activate a specified control and its latent embedded controls.

An embedded control is considered latent when it is deactivated or hidden due to its embedder control being deactivated or hidden. If you activate a latent embedded control whose embedder is deactivated, the embedded control becomes latent until the embedder is activated. However, if you deactivate a latent embedded control, it will not be activated when its embedder is activated.

If a control definition function supports activate events, it will receive a `kControlMsgActivate` message before redrawing itself in its active state.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

BSDLLCTest

Declared In

`Controls.h`

AdvanceKeyboardFocus

Advances the keyboard focus to the next focusable control in a window.

```
OSErr AdvanceKeyboardFocus (
   WindowRef inWindow
);
```

Parameters

inWindow

A pointer to the window for which to advance keyboard focus.

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Discussion
The `AdvanceKeyboardFocus` function skips over deactivated and hidden controls until it finds the next focusable control in the window. If it does not find a focusable item, it simply returns.

When `AdvanceKeyboardFocus` is called, the Control Manager calls your control definition function and passes `kControlMsgFocus` in its `message` parameter and `kControlFocusNextPart` in its `param` parameter. In response to this message, your control definition function should change keyboard focus to its next part, the entire control, or remove keyboard focus from the control, depending upon the circumstances. See `ControlDefProcPtr` (page 677) for a discussion of possible responses to this message.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

AutoEmbedControl

Automatically embeds a control in the smallest appropriate embedder control.

```
OSErr AutoEmbedControl (
   ControlRef inControl,
   WindowRef inWindow
);
```

Parameters

inControl
A handle to the control to be embedded.

inWindow
A pointer to the window in which to embed the control.

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Discussion
The Dialog Manager uses `AutoEmbedControl` (page 536) to position dialog items in an embedding hierarchy based on both visual containment and the item list resource order. As items are added to a dialog box during creation, controls that already exist in the window will be containers for new controls if they both visually contain the control and have set the `kControlSupportsEmbedding` feature bit. For this reason, you should place the largest embedder controls at the beginning of the item list resource. As an example, the Dialog Manager would embed radio buttons in a tab control if they visually "fit" inside the tab control, as long as the tab control was already created in a `'DITL'` resource and established as an embedder control. For more information on embedding hierarchies in dialog and alert boxes, see the function `EmbedControl` (page 583).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

ChangeControlPropertyAttributes

Changes a property attribute.

```
OSStatus ChangeControlPropertyAttributes (
   ControlRef control,
   OSType propertyCreator,
   OSType propertyTag,
   OptionBits attributesToSet,
   OptionBits attributesToClear
);
```

Parameters

control

The control whose property's attributes you want to change. For a description of this data type, see ControlRef (page 709).

propertyCreator

An OSType signature, usually the signature of your application. Do not use all lower case signatures, as these are reserved for use by Apple.

propertyTag

An OSType signature, defined by your application, defining the property whose attributes you want to change.

attributesToSet

A bit field indicating the attributes you want to set for this property.

attributesToClear

A bit field indicating the attributes you want to clear for this property.

Return Value

A result code. See "Control Manager Result Codes" (page 824).

Discussion

If you have associated control properties with a control (by calling `SetControlProperty` (page 658), you can also assign arbitrary attribute bits to the property. You can use these attributes to indicate information about the property data.

Currently, `kControlPropertyPersistent` is the only control property attribute that is defined.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Controls.h`

ClearKeyboardFocus

Removes the keyboard focus for the currently focused control in a window.

```
OSErr ClearKeyboardFocus (
   WindowRef inWindow
);
```

Parameters

inWindow

A pointer to the window in which to clear keyboard focus.

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Discussion
When the `ClearKeyboardFocus` function is called, the Control Manager calls your control definition function and passes `kControlMsgFocus` in its `message` parameter and `kControlFocusNoPart` in its `param` parameter. See `ControlDefProcPtr` (page 677) for a discussion of possible responses to this message.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

CopyControlTitleAsCFString

Makes a copy of the control's title as a Core Foundation string.

```
OSStatus CopyControlTitleAsCFString (
   ControlRef inControl,
   CFStringRef *outString
);
```

Parameters

inControl

The control whose title is to be copied. For a description of this data type, see ControlRef (page 709).

outString

A copy of the control's title.

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

CountSubControls

Obtains the number of embedded controls within a control.

```
OSErr CountSubControls (
   ControlRef inControl,
   UInt16 *outNumChildren
);
```

Parameters

inControl

The control whose embedded controls are to be counted. For a description of this data type, see ControlRef (page 709).

outNumChildren

On input, a pointer to an unsigned 16-bit integer value. On return, the value is set to the number of embedded subcontrols.

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Discussion
The `CountSubControls` function is useful for iterating over the control hierarchy. You can use the count produced to determine how many subcontrols there are and then call `GetIndexedSubControl` (page 608) to get each.

Availability
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code
HID Calibrator

Declared In
`Controls.h`

CreateBevelButtonControl

Creates a bevel button control.

```
OSStatus CreateBevelButtonControl (
   WindowRef window,
   const Rect *boundsRect,
   CFStringRef title,
   ControlBevelThickness thickness,
   ControlBevelButtonBehavior behavior,
   ControlButtonContentInfoPtr info,
   MenuID menuID,
   ControlBevelButtonMenuBehavior menuBehavior,
   ControlBevelButtonMenuPlacement menuPlacement,
   ControlRef *outControl
);
```

Parameters

window

The window that is to contain the control. This parameter may be `NULL` in Mac OS X v10.3 and later.

boundsRect

The bounds of the desired control in the window's local coordinates.

title
The title of the control.

thickness
The thickness of the button. For possible values, see "Control Bevel Thickness Constants" (page 753).

behavior
The behavior the button is to have. For possible values, see "Bevel Button Behavior Constants" (page 721).

info
A value of type `ControlButtonContentInfoPtr` for the content information.

menuID
The menu ID. This parameter may be 0 if you don't have a menu. Icon suite, picture, color icon, and `IconRef` are supported on Mac OS X v10.0 through Mac OS X v10.4. Values of type `CGImageRef` are supported in Mac OS X v10.4.

menuBehavior
The behavior of the menu. For possible values, see "Bevel Button Menu Constant" (page 727).

menuPlacement
The placement of the menu. For possible values, see "Control Bevel Button Menu Placement Constants" (page 753).

outControl
On return, `outControl` points to the new control. For a description of this data type, see ControlRef (page 709).

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`HIButtonViews.h`

CreateChasingArrowsControl

Creates a chasing arrows control.

```
OSStatus CreateChasingArrowsControl (
   WindowRef window,
   const Rect *boundsRect,
   ControlRef *outControl
);
```

Parameters

window
The window that is to contain the control. This parameter may be `NULL` in Mac OS X v10.3 and later.

boundsRect
The bounds of the desired control in the window's local coordinates.

outControl
On return, `outControl` points to the new control. For a description of this data type, see ControlRef (page 709).

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Discussion
This control automatically animates via an event loop timer.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`HIProgressViews.h`

CreateCheckBoxControl

Creates a checkbox control.

```
OSStatus CreateCheckBoxControl (
   WindowRef window,
   const Rect *boundsRect,
   CFStringRef title,
   SInt32 initialValue,
   Boolean autoToggle,
   ControlRef *outControl
);
```

Parameters

window
The window that is to contain the checkbox control. This parameter may be `NULL` in Mac OS X v10.3 and later.

boundsRect
The bounds of the desired checkbox in the window's local coordinates.

title
The title of the checkbox.

initialValue
The initial setting of the checkbox. Set to a non-zero value to indicate the checked state.

autoToggle
If set to `true`, clicking the checkbox will automatically toggle its state (checked or unchecked).

outControl
On return, `outControl` points to the new checkbox. For a description of this data type, see ControlRef (page 709).

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`HIButtonViews.h`

CreateCheckGroupBoxControl

Creates a group box control that has a check box as its title.

```
OSStatus CreateCheckGroupBoxControl (
   WindowRef window,
   const Rect *boundsRect,
   CFStringRef title,
   SInt32 initialValue,
   Boolean primary,
   Boolean autoToggle,
   ControlRef *outControl
);
```

Parameters

window

The window in which the control is to be placed. This parameter may be `NULL` in Mac OS X v10.3 and later.

boundsRect

The bounds of the control in the window's local coordinates.

title

The title of the control. The title is used as the title of the check box.

initialValue

The initial value of the check box.

primary

A Boolean whose value is `true` to create a primary group box or `false` to create a secondary group box.

autoToggle

A Boolean whose value is true to create an auto-toggling check box. Auto-toggling check box titles are only supported on Mac OS X; this parameter must be `false` when used with CarbonLib.

outControl

On return, the new control. For a description of this data type, see ControlRef (page 709).

Return Value

A result code. See "Control Manager Result Codes" (page 824).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`HIContainerViews.h`

CreateClockControl

Creates a clock control.

```
OSStatus CreateClockControl (
   WindowRef window,
   const Rect *boundsRect,
   ControlClockType clockType,
   ControlClockFlags clockFlags,
   ControlRef *outControl
);
```

Parameters

window

The window that is to contain the control. This parameter may be `NULL` in Mac OS X v10.3 and later.

boundsRect

The bounds of the desired control in the window's local coordinates.

clockType

The clock type. For possible values, see "Control Clock Type Constants" (page 753).

clockFlags

Clock options. For possible values, see "Clock Value Flag Constants" (page 734).

outControl

On return, `outControl` points to the new control. For a description of this data type, see ControlRef (page 709).

Return Value

A result code. See "Control Manager Result Codes" (page 824).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`HIClockView.h`

CreateCustomControl

Creates a custom control. (Deprecated in Mac OS X v10.5. Register your custom subclass of the HIView class and create an instance of your class using `HIObjectCreate`.)

```
OSStatus CreateCustomControl (
   WindowRef owningWindow,
   const Rect *contBounds,
   const ControlDefSpec *def,
   Collection initData,
   ControlRef *outControl
);
```

Parameters

owningWindow

The window that is to contain the control. This parameter may be `NULL` in Mac OS X v10.3 and later.

contBounds

The bounds of the new control in the window's local coordinates.

def

A pointer to the control definition function you want to associate with the new control.

initData
The initial state of the control. For additional information, see "Control Collection Tag Constants" (page 768).

outControl
On return, `outControl` points to the new control.

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Controls.h`

CreateDisclosureButtonControl

Creates a new instance of the Disclosure Button Control.

```
OSStatus CreateDisclosureButtonControl (
   WindowRef inWindow,
   const Rect *inBoundsRect,
   SInt32 inValue,
   Boolean inAutoToggles,
   ControlRef *outControl
);
```

Parameters

inWindow
The `WindowRef` in which to create the control. This parameter may be `NULL` in Mac OS X v10.3 and later.

inBoundsRect
The bounding rectangle for the control in the window's local coordinates. The height of the control is fixed and the control will be centered vertically within the rectangle you specify.

inValue
The initial value; either `kControlDisclosureButtonClosed` or `kControlDisclosureButtonDisclosed`.

inAutoToggles
A Boolean value indicating whether its value should change automatically after tracking the mouse.

outControl
On successful exit, this will contain the new control. For a description of this data type, see ControlRef (page 709).

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Discussion
`CreateDisclosureButtonControl` is preferred over `NewControl` (page 634) because it allows you to specify the exact set of parameters required to create the control without overloading parameter semantics. The initial minimum of the Disclosure Button will be `kControlDisclosureButtonClosed`, and the maximum will be `kControlDisclosureButtonDisclosed`.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDisclosureViews.h`

CreateDisclosureTriangleControl

Creates a disclosure triangle control.

```
OSStatus CreateDisclosureTriangleControl (
   WindowRef inWindow,
   const Rect *inBoundsRect,
   ControlDisclosureTriangleOrientation inOrientation,
   CFStringRef inTitle,
   SInt32 inInitialValue,
   Boolean inDrawTitle,
   Boolean inAutoToggles,
   ControlRef *outControl
);
```

Parameters

window
The window that is to contain the control. This parameter may be `NULL` in Mac OS X v10.3 and later.

inBoundsRect
The desired position, in the window's local coordinates, for the disclosure triangle.

inOrientation
The direction the disclosure triangle should point when it is "closed." Passing `kControlDisclosureTrianglePointDefault` is only legal as of Mac OS X and CarbonLib 1.5. For other possible values, see "Control Disclosure Triangle Orientation Constants" (page 753).

inTitle
The title for the disclosure triangle. The title is displayed only if the value of the `inDrawTitle` parameter is `true`. Displaying the title only works on Mac OS X.

inInitialValue
The initial state of the disclosure triangle. A value of `0` causes the disclosure triangle to be drawn initially in the "closed" state, and a value of `1` causes the triangle to be drawn initially in the "open" state.

inDrawTitle
A Boolean whose value is true if the disclosure triangle should draw its title next to the widget. Displaying the title only works on Mac OS X.

inAutoToggles
A Boolean whose value is `true` to enable auto toggling; otherwise, `false`. When auto toggling is enabled, the disclosure triangle automatically changes from "open" to "closed" and from "closed" to "open" when it is clicked.

outControl

On return, `outControl` points to the new control. For a description of this data type, see ControlRef (page 709).

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Discussion
A disclosure triangle is a small control that gives the user a way to toggle the visibility of information or other user interface. When information is in a hidden state, a disclosure triangle is considered "closed" and should point to the right (or sometimes to the left). When the user clicks it, a disclosure triangle rotates downwards into the "open" state. The application should respond by revealing the appropriate information or interface.

On Mac OS X, a root control is created for the window if one does not already exist. If a root control exists for the window, the disclosure triangle control is embedded in it.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDisclosureViews.h`

CreateEditTextControl

Creates a new edit text control. (Deprecated in Mac OS X v10.4. Use `CreateEditUnicodeTextControl` (page 547) instead.)

```
OSStatus CreateEditTextControl (
   WindowRef window,
   const Rect *boundsRect,
   CFStringRef text,
   Boolean isPassword,
   Boolean useInlineInput,
   const ControlFontStyleRec *style,
   ControlRef *outControl
);
```

Parameters

window

The window in which the control is to be placed. May be `NULL` in Mac OS X v10.3 and later.

boundsRect

The bounds of the control in the window's local coordinates.

text

The text of the control. May be `NULL`.

isPassword

A Boolean indicating whether the field is to be used as a password field. Passing `false` indicates that the field is to display entered text normally. Passing `true` means that the field is to be used as a password field; any text typed into the field is displayed as bullets.

useInlineInput

A Boolean indicating whether the control is to accept inline input. Pass `true` to accept inline input; otherwise pass `false`.

style
The control's font style, size, color, and so on. May be `NULL`.

outControl
On return, the new control.

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`ControlDefinitions.h`

CreateEditUnicodeTextControl

Creates a new edit text control.

```
OSStatus CreateEditUnicodeTextControl (
   WindowRef window,
   const Rect *boundsRect,
   CFStringRef text,
   Boolean isPassword,
   const ControlFontStyleRec *style,
   ControlRef *outControl
);
```

Parameters

window
The window in which the control is to be placed. This parameter may be `NULL` in Mac OS X v10.3 and later.

boundsRect
The bounds of the control in the window's local coordinates.

text
The text of the control. May be `NULL`.

isPassword
A Boolean indicating whether the field is to be used as a password field. Passing `false` indicates that text entered in the field is to be displayed normally. Passing `true` means that the field is to be used as a password field; any text typed into the field is displayed as bullets.

style
The control's font style, size, color, and so on. May be `NULL`.

outControl
On return, the new control. For a description of this data type, see ControlRef (page 709).

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Discussion
This function is the preferred way of creating edit text controls. Use it instead of the `CreateEditTextControl` (page 546) function. The resulting control handles Unicode text and draws its text using anti-aliasing. Controls created by `CreateEditTextControl` do not handle Unicode text and are not drawn with anti-aliasing.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`HITextViews.h`

CreateGroupBoxControl

Creates a group box control.

```
OSStatus CreateGroupBoxControl (
   WindowRef window,
   const Rect *boundsRect,
   CFStringRef title,
   Boolean primary,
   ControlRef *outControl
);
```

Parameters

window
The window that is to contain the control. This parameter may be `NULL` in Mac OS X v10.3 and later.

boundsRect
The bounds of the desired control in the window's local coordinates.

title
The title of the control. This parameter can be `NULL` if you don't want the control to have a title.

primary
A Boolean whose value is `true` to create a primary group box or `false` to create a secondary group box. Secondary group boxes are intended to be contained within primary group boxes and have a slightly different appearance.

outControl
On return, `outControl` points to the new control. For a description of this data type, see ControlRef (page 709).

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`HIContainerViews.h`

CreateIconControl

Creates an icon control.

```
OSStatus CreateIconControl (
   WindowRef inWindow,
   const Rect *inBoundsRect,
   const ControlButtonContentInfo *inIconContent,
   Boolean inDontTrack,
   ControlRef *outControl
);
```

Parameters

inWindow

The window in which the control is to be placed. This parameter may be `NULL` in Mac OS X v10.3 and later.

inBoundsRect

The bounds of the control in the window's local coordinates.

inIconContent

The descriptor for the icon you want the control to display. Mac OS X and CarbonLib 1.5 (and beyond) support all of the icon content types. Prior to CarbonLib 1.5, the only content types that are properly respected are `kControlContentIconSuiteRes`, `kControlContentCIconRes`, and `kControlContentICONRes`.

inDontTrack

A Boolean whose value is `true` to indicate that the control should not be highlighted when it is clicked; `false` means that the control should be highlighted and the mouse tracked when the control is clicked.

outControl

On return, the new control. For a description of this data type, see ControlRef (page 709).

Return Value

A result code. See "Control Manager Result Codes" (page 824).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`HIImageViews.h`

CreateImageWellControl

Creates an image well control.

```
OSStatus CreateImageWellControl (
   WindowRef window,
   const Rect *boundsRect,
   const ControlButtonContentInfo *info,
   ControlRef *outControl
);
```

Parameters

window

The window that is to contain the control. This parameter may be `NULL` in Mac OS X v10.3 and later.

boundsRect

The bounds of the control in the window's local coordinates.

info

The image that is to be displayed in the image well. The image can be an icon suite, picture, color icon, or an `IconRef` in Mac OS X v10.0 and later. It can be also be a `CGImageRef` in Mac OS X v10.4 and later.

outControl

On return, `outControl` points to the new control. For a description of this data type, see ControlRef (page 709).

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Discussion
An image well control is a control that displays an image inside a frame (or "well"). The user can drag other images onto the well.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`HIImageViews.h`

CreateListBoxControl

Creates a list box control.

```
OSStatus CreateListBoxControl (
   WindowRef window,
   const Rect *boundsRect,
   Boolean autoSize,
   SInt16 numRows,
   SInt16 numColumns,
   Boolean horizScroll,
   Boolean vertScroll,
   SInt16 cellHeight,
   SInt16 cellWidth,
   Boolean hasGrowSpace,
   const ListDefSpec *listDef,
   ControlRef *outControl
);
```

Parameters

window

The window that is to contain the control. This parameter may be `NULL` in Mac OS X v10.3 and later.

boundsRect

The bounds of the control in the window's local coordinates.

autoSize

A Boolean whose value is `true` to enable auto-sizing; otherwise, `false`. When auto-sizing is enabled, the control automatically resizes itself as necessary to ensure that the height of the control is an exact multiple of the cell height.

numRows

The number of rows the control is to have.

numColumns

The number of columns the control is to have.

horizScroll

A Boolean whose value is `true` if the control is to have a horizontal scroll bar; otherwise, `false`.

vertScroll

A Boolean whose value is `true` if the control is to have a vertical scroll bar; otherwise, `false`.

cellHeight

The height of cells in the control.

cellWidth

The width of cells in the control.

hasGrowSpace

A Boolean whose value is `true` to indicate that the control is drawn so that there is room for a size box; otherwise, `false`.

listDef

A pointer to the list definition function you want to associate with the new control. This parameter may be `NULL` if you want to use the standard list definition function, which only displays text.

outControl

On return, `outControl` points to the new control. For a description of this data type, see ControlRef (page 709).

Return Value

A result code. See "Control Manager Result Codes" (page 824).

Discussion
The list is created with default values, and uses the standard LDEF (0) if you don't specify a custom list definition function in the `listDef` parameter. You can set the LDEF to use by using `kControlListBoxLDEFTag`. You can change the list by getting the list handle. To get the list handle, call `GetControlData` (page 594) and pass the `kControlListBoxListHandletag` constant.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ControlDefinitions.h`

CreateLittleArrowsControl

Creates a little arrows control.

```
OSStatus CreateLittleArrowsControl (
   WindowRef window,
   const Rect *boundsRect,
   SInt32 value,
   SInt32 minimum,
   SInt32 maximum,
   SInt32 increment,
   ControlRef *outControl
);
```

Parameters

window
The window that is to contain the control. This parameter may be `NULL` in Mac OS X v10.3 and later.

boundsRect
The bounds of the control in the window's local coordinates.

value
The initial value of the control.

minimum
The minimum value the control can have.

maximum
The maximum value the control can have.

increment
The amount to increment each time an arrow is clicked.

outControl
On return, `outControl` points to the new control. For a description of this data type, see ControlRef (page 709).

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Discussion
This control implements the little up and down arrows seen in the Date & Time system preferences panel. To change the value of this control, you need to create a control action proc. The following sample code creates the control and sets the action proc:

```
CreateLittleArrowsControl(&rect, minimum, maximum, increment,  value);
SetControlAction(Arrows, LittleArrowActionProc);
```

Here is sample code for the action proc:

```
void LittleArrowActionProc(ControlRef cref, ControlPartCode part)  {
    SInt32 val = GetControl32BitValue(cref);
    SInt32 s = 0;
   GetControlData(cref, 0, kControlLittleArrowsIncrementValueTag, sizeof(SInt32),
 &s, nil;
    switch (part) {
        case kControlUpButtonPart:
            SetControl32BitValue(cref, val+s);
            break;
        case kControlDownButtonPart:
            SetControl32BitValue(cref, val-s);
            break;
        };
    };
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`HILittleArrows.h`

CreatePictureControl

Creates a picture control.

```
OSStatus CreatePictureControl (
   WindowRef window,
   const Rect *boundsRect,
   const ControlButtonContentInfo *content,
   Boolean dontTrack,
   ControlRef *outControl
);
```

Parameters

window
The window in which the control is to be placed. This parameter may be `NULL` in Mac OS X v10.3 and later.

boundsRect
The bounds of the control in the window's local coordinates.

content
The descriptor for the picture you want the control to display. Only picture content is supported. You can change the picture by calling `SetControlData` (page 652) and passing the `kControlPictureHandleTag` constant.

dontTrack
A Boolean whose value is `true` to indicate that the control should not be highlighted when it is clicked; `false` means that the control should be highlighted and the mouse tracked when the control is clicked.

outControl
On return, the new control.

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ControlDefinitions.h`

CreatePlacardControl

Creates a placard control.

```
OSStatus CreatePlacardControl (
   WindowRef window,
   const Rect *boundsRect,
   ControlRef *outControl
);
```

Parameters

window
The window that is to contain the control. This parameter may be `NULL` in Mac OS X v10.3 and later.

boundsRect
The bounding box of the control in the window's local coordinates.

outControl
On return, the new control. For a description of this data type, see ControlRef (page 709).

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`HIContainerViews.h`

CreatePopupArrowControl

Creates a pop-up arrow control.

```
OSStatus CreatePopupArrowControl (
   WindowRef window,
   const Rect *boundsRect,
   ControlPopupArrowOrientation orientation,
   ControlPopupArrowSize size,
   ControlRef *outControl
);
```

Parameters

window

The window that is to contain the control. This parameter may be `NULL` in Mac OS X v10.3 and later.

boundsRect

The bounds of the control in the window's local coordinates.

orientation

The orientation of the control.

size

The size of the control.

outControl

On return, the new control. For a description of this data type, see ControlRef (page 709).

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`HIPopupButton.h`

CreatePopupButtonControl

Creates a pop-up button control.

```
OSStatus CreatePopupButtonControl (
   WindowRef window,
   const Rect *boundsRect,
   CFStringRef title,
   MenuID menuID,
   Boolean variableWidth,
   SInt16 titleWidth,
   SInt16 titleJustification,
   Style titleStyle,
   ControlRef *outControl
);
```

Parameters

window

The window in which the control is to be placed. This parameter may be `NULL` in Mac OS X v10.3 and later.

boundsRect

The bounds of the control in the window's local coordinates.

title
The title of the control.

menuID
The ID of a menu that should be used by the control. A menu with this ID should be inserted into the menubar with `InsertMenu(menu, kInsertHierarchicalMenu)`. You can also pass `-12345` to have the control delay its acquisition of a menu; in this case, you can build the menu and later provide it to the control with `SetControlData` and `kControlPopupButtonMenuRefTag` or `kControlPopupButtonOwnedMenuRefTag`.

variableWidth
A Boolean whose value indicates whether the width of the control is allowed to vary according to the width of the selected menu item text (`true`), or should remain fixed to the original control bounds width (`false`).

titleWidth
The width of the title.

titleJustification
The justification of the title. Use a TextEdit justification constant (`teFlushDefault`, `teCenter`, `teFlushRight`, or `teFlushLeft`).

titleStyle
A QuickDraw style bitfield indicating the font style of the title.

outControl
On return, the new control. For a description of this data type, see ControlRef (page 709).

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`HIPopupButton.h`

CreatePopupGroupBoxControl

Creates a group box control that has a pop-up button as its title.

```
OSStatus CreatePopupGroupBoxControl (
   WindowRef window,
   const Rect *boundsRect,
   CFStringRef title,
   Boolean primary,
   MenuID menuID,
   Boolean variableWidth,
   SInt16 titleWidth,
   SInt16 titleJustification,
   Style titleStyle,
   ControlRef *outControl
);
```

Parameters

window

The window that is to contain the control. This parameter may be `NULL` in Mac OS X v10.3 and later.

boundsRect

The bounds of the control in the window's local coordinates.

title

The title of the control. The title is used as the title of the pop-up button.

primary

A Boolean whose value is `true` to create a primary group box or `false` to create a secondary group box.

menuID

The menu ID of the menu that is to be displayed by the pop-up button. A menu with this ID should be inserted into the menubar with `InsertMenu(menu, kInsertHierarchicalMenu)`. You can also pass `-12345` to have the control delay its acquisition of a menu; in this case, you can build the menu and later provide it to the control with `SetControlData` and `kControlPopupButtonMenuRefTag` or `kControlPopupButtonOwnedMenuRefTag`.

variableWidth

A Boolean whose value is `true` if the pop-up button is to have a variable-width title or `false` if the pop-up button is to have a fixed-width title. Fixed-width titles are only supported by Mac OS X; this parameter must be `true` when used with CarbonLib.

titleWidth

The width in pixels of the pop-up button title.

titleJustification

The justification of the pop-up button title. Use a TextEdit justification constant (`teFlushDefault`, `teCenter`, `teFlushRight`, or `teFlushLeft`).

titleStyle

The QuickDraw text style of the pop-up button title.

outControl

On return, `outControl` points to the new control. For a description of this data type, see ControlRef (page 709).

Return Value

A result code. See "Control Manager Result Codes" (page 824).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
HIContainerViews.h

CreateProgressBarControl

Creates a progress bar control.

```
OSStatus CreateProgressBarControl (
   WindowRef window,
   const Rect *boundsRect,
   SInt32 value,
   SInt32 minimum,
   SInt32 maximum,
   Boolean indeterminate,
   ControlRef *outControl
);
```

Parameters

window
The window that is to contain the control. This parameter may be `NULL` in Mac OS X v10.3 and later.

boundsRect
The bounds of the control in the window's local coordinates.

value
The initial value of the control.

minimum
The minimum value of the control.

maximum
The maximum value of the control.

indeterminate
A Boolean whose value is `true` if you want the control to display a rotating barber pole effect to indicate that something is happening (an indeterminate progress bar) or `false` if you want to display a determinate progress bar that uses the values of the `minimum` and `maximum` parameters to show progress from minimum to maximum.

outControl
On return, `outControl` points to the new control. For a description of this data type, see ControlRef (page 709).

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
HIProgressViews.h

CreatePushButtonControl

Creates a push button control.

```
OSStatus CreatePushButtonControl (
   WindowRef window,
   const Rect *boundsRect,
   CFStringRef title,
   ControlRef *outControl
);
```

Parameters

window

The window in which the control is to be placed. This parameter may be `NULL` in Mac OS X v10.3 and later.

boundsRect

The bounds of the control in the window's local coordinates.

title

The control title. May be `NULL`.

outControl

On return, the new control. For a description of this data type, see ControlRef (page 709).

Return Value

A result code. See "Control Manager Result Codes" (page 824).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`HIButtonViews.h`

CreatePushButtonWithIconControl

Creates a push button control containing an icon or other graphical content.

```
OSStatus CreatePushButtonWithIconControl (
   WindowRef window,
   const Rect *boundsRect,
   CFStringRef title,
   ControlButtonContentInfo *icon,
   ControlPushButtonIconAlignment iconAlignment,
   ControlRef *outControl
);
```

Parameters

window

The window in which the control is to be placed. This parameter may be `NULL` in Mac OS X v10.3 and later.

boundsRect

The bounds of the control, in local coordinates of the window.

title

The control title. May be `NULL`.

icon

The control graphic content. The value of this parameter can be `kControlContentCIconRes` in Mac OS X v10.0 and later. It can also be `kControlContentCGImageRef` in Mac OS X v10.4 and later.

iconAlignment
The alignment of the control graphic content.For possible values, see "Control Push Button Icon Alignment Constants" (page 753).

outControl
On return, the new control. For a description of this data type, see ControlRef (page 709).

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`HIButtonViews.h`

CreateRadioButtonControl

Creates a radio button control.

```
OSStatus CreateRadioButtonControl (
   WindowRef window,
   const Rect *boundsRect,
   CFStringRef title,
   SInt32 initialValue,
   Boolean autoToggle,
   ControlRef *outControl
);
```

Parameters

window
The window in which the control is to be placed. This parameter may be `NULL` in Mac OS X v10.3 and later.

boundsRect
The bounds of the control in the window's local coordinates.

title
The control title. May be `NULL`.

initialValue
The initial value of the control. Should be `0` (off), `1` (on), or `2` (mixed). The control is automatically given a minimum value of 0 and a maximum value of 2.

autoToggle
A Boolean whose value indicates whether this control should have auto-toggle behavior. If `true`, the control automatically toggles between on and off states when clicked. This parameter should be `false` if the control is embedded into a radio group control; in that case, the radio group handles setting the correct control value in response to a click.

outControl
On return, the new control.

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
HIButtonViews.h

CreateRadioGroupControl

Creates a radio group control.

```
OSStatus CreateRadioGroupControl (
   WindowRef window,
   const Rect *boundsRect,
   ControlRef *outControl
);
```

Parameters

window
The window that is to contain the control. This parameter may be NULL in Mac OS X v10.3 and later.

boundsRect
The bounds of the control in the window's local coordinates.

outControl
On return, outControl points to the new control. For a description of this data type, see ControlRef (page 709).

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Discussion
This control implements a radio group. It is an embedding control and can therefore only be used when a control hierarchy is established for its owning window. You should only embed radio buttons within it. As radio buttons are embedded into it, the group sets up its value, min, and max to represent the number of embedded items. The current value of the control is the index of the sub-control that is the current "on" radio button. To get the current radio button control handle, you can use the Control Manager call GetIndexedSubControl (page 608), passing in the value of the radio group.

Note that when creating radio buttons for use in a radio group control, you should not use the auto-toggle version of the radio button. The radio group control handles toggling the radio button values itself; auto-toggle radio buttons do not work properly in a radio group control on Mac OS 9.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
HIButtonViews.h

CreateRelevanceBarControl

Creates a relevance bar control.

```
OSStatus CreateRelevanceBarControl (
   WindowRef window,
   const Rect *boundsRect,
   SInt32 value,
   SInt32 minimum,
   SInt32 maximum,
   ControlRef *outControl
);
```

Parameters

window

The window that is to contain the control. This parameter may be `NULL` in Mac OS X v10.3 and later.

boundsRect

The bounds of the control in the window's local coordinates.

value

The initial value of the control.

minimum

The minimum value of the control.

maximum

The maximum value of the control.

outControl

On return, `outControl` points to the new control. For a description of this data type, see ControlRef (page 709).

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`HIRelevanceBar.h`

CreateRootControl

Creates the root control for a specified window.

```
OSErr CreateRootControl (
   WindowRef inWindow,
   ControlRef *outControl
);
```

Parameters

inWindow

A pointer to the window in which you wish to create a root control.

outControl

On input, a pointer to a `ControlHandle` value. On return, the `ControlHandle` value is set to a handle to the root control.

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Discussion
Establishing an embedding hierarchy can be accomplished in two steps: creating a root control and embedding controls within it.

To embed controls in a window, you must create a root control for that window. The root control is the container for all other window controls. You create the root control in one of two ways—by calling the `CreateRootControl` function or by setting the appropriate dialog flag. The root control can be retrieved by calling the function `GetRootControl` (page 611).

The `CreateRootControl` function creates the root control for a window if no other controls are present. If there are any controls in the window prior to calling `CreateRootControl`, an error is returned and the root control is not created. Note that the minimum, maximum, and initial settings for a root control are reserved and should not be changed.

The root control is implemented as a user pane control. You can attach any application-defined user pane functions to the root control to perform actions such as hit testing, drawing, handling keyboard focus, erasing to the correct background, and processing idle and keyboard events.

Once you have created a root control, newly created controls will automatically be embedded in the root control when you call `NewControl` (page 634) or `GetNewControl` (page 610). You can specify that a specific control be embedded into another by calling `EmbedControl` (page 583).

By acting on an embedder control, you can move, disable, or hide groups of items. For example, you can use a blank user pane control as the embedder control for all items in a particular "page" of a tab control. After creating as many user panes as you have tabs, you can hide one and show the next when a tab is clicked. All the controls embedded in the user pane will be hidden and shown automatically when the user pane is hidden and shown.

In addition to calling `CreateRootControl`, you can establish an embedding hierarchy in a dialog box by either setting the feature bit `kDialogFlagsUseControlHierarchy` in the extended dialog resource or passing it in the `inFlags` parameter of the Dialog Manager function `NewFeaturesDialog`. An embedding hierarchy can be created in an alert box by setting the `kAlertFlagsUseControlHierarchy` bit in the extended alert resource. It is important to note that a preexisting alert or dialog item will become a control if it is in an alert or dialog box that now uses an embedding hierarchy.

The embedding hierarchy enforces drawing order by drawing the embedding control before its embedded controls. Using an embedding hierarchy also enforces orderly hit-testing, since it performs an "inside-out" hit test to determine the most deeply nested control that is hit by the mouse. An embedding hierarchy is also necessary for controls to make use of keyboard focus, the default focusing order for which is a linear progression that uses the order the controls were added to the window. For more details on keyboard focus, see "Handling Keyboard Focus".

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
QTCarbonShell

Declared In
`Controls.h`

CreateRoundButtonControl

Creates a new instance of the round button control.

```
OSStatus CreateRoundButtonControl (
   WindowRef inWindow,
   const Rect *inBoundsRect,
   ControlRoundButtonSize inSize,
   ControlButtonContentInfo *inContent,
   ControlRef *outControl
);
```

Parameters

inWindow

The `WindowRef` in which to create the control.

inBoundsRect

The bounds of the control in the window's local coordinates. The height and width of the control are fixed (specified by the `ControlRoundButtonSize` parameter) and the control will be centered within the rectangle you specify.

inSize

The button size; either `kControlRoundButtonNormalSize` or `kControlRoundButtonLargeSize`.

inContent

Any optional content displayed in the button. In Mac OS X v10.0 and later, `kControlContentIconRef` is supported.

outControl

On return, the new control. For a description of this data type, see ControlRef (page 709).

Return Value

A result code. See "Control Manager Result Codes" (page 824).

Discussion

`CreateRoundButtonControl` is preferred over `NewControl` (page 634) because it allows you to specify the exact set of parameters required to create the control without overloading parameter semantics.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`HIButtonViews.h`

CreateScrollBarControl

Creates a scroll bar control.

```
OSStatus CreateScrollBarControl (
   WindowRef window,
   const Rect *boundsRect,
   SInt32 value,
   SInt32 minimum,
   SInt32 maximum,
   SInt32 viewSize,
   Boolean liveTracking,
   ControlActionUPP liveTrackingProc,
   ControlRef *outControl
);
```

Parameters

window

The window in which the control is to be placed. This parameter may be `NULL` in Mac OS X v10.3 and later.

boundsRect

The bounds of the control in the window's local coordinates.

value

The initial value of the control.

minimum

The minimum value of the control.

maximum

The maximum value of the control.

viewSize

The size of the visible area of the scroll bar content. If this parameter is a non-zero value, this function creates a proportional scroll bar thumb; a value of `0` causes a non-proportional scroll bar thumb to be created.

liveTracking

A Boolean indicating whether or not live tracking is enabled for this scroll bar. If set to `true` and a valid `liveTrackingProc` is also passed in, the callback is called repeatedly as the thumb is moved during tracking. If set to `false`, a semi-transparent thumb called a "ghost thumb" draws and no live tracking occurs.

liveTrackingProc

If the value of the `liveTracking` parameter is `true`, a `ControlActionUPP` callback is to be called as the control live tracks. This callback is called repeatedly as the scroll thumb is moved during tracking.

outControl

On return, the new control. For a description of this data type, see ControlRef (page 709).

Return Value

A result code. See "Control Manager Result Codes" (page 824).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`HIScrollView.h`

CreateScrollingTextBoxControl

Creates a scrolling text box control.

```
OSStatus CreateScrollingTextBoxControl (
   WindowRef window,
   const Rect *boundsRect,
   SInt16 contentResID,
   Boolean autoScroll,
   UInt32 delayBeforeAutoScroll,
   UInt32 delayBetweenAutoScroll,
   UInt16 autoScrollAmount,
   ControlRef *outControl
);
```

Parameters

window
The window that is to contain the control. This parameter may be `NULL` in Mac OS X v10.3 and later.

boundsRect
The bounds of the control in the window's local coordinates.

contentResID
The resource ID of '`TEXT`' and (optionally) '`style`' resources whose contents are to be displayed.

autoScroll
A Boolean whose value is `true` to enable automatic scrolling; otherwise, `false`.

delayBeforeAutoScroll
The number of ticks to wait before scrolling automatically. This parameter is ignored and can be set to `0` if the value of the `autoScroll` parameter is `false`.

delayBetweenAutoScroll
The number of ticks to wait between automatic scrolls. This parameter is ignored and can be set to `0` if the value of the `autoScroll` parameter is `false`.

autoScrollAmount
The number of pixels to scroll. This parameter is ignored and can be set to `0` if the value of the `autoScroll` parameter is `false`.

outControl
On return, `outControl` points to the newly-created control.

Return Value

A result code. See "Control Manager Result Codes" (page 824).

Discussion

This control implements a scrolling box of text that cannot be edited. This is useful for credits in about boxes.

The standard version of this control has a scroll bar, but the autoscrolling variant does not. The autoscrolling variant needs two pieces of information to work: delay (in ticks) before the scrolling starts, and time (in ticks) between scrolls. This control scrolls one pixel at a time if created by `NewControl` (page 634), unless changed by calling `SetControlData` (page 652).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`ControlDefinitions.h`

CreateSeparatorControl

Creates a separator control.

```
OSStatus CreateSeparatorControl (
   WindowRef window,
   const Rect *boundsRect,
   ControlRef *outControl
);
```

Parameters

window

The window that is to contain the control. This parameter may be `NULL` in Mac OS X v10.3 and later.

boundsRect

The bounds of the control in the window's local coordinates.

outControl

On return, `outControl` points to the new control. For a description of this data type, see ControlRef (page 709).

Return Value

A result code. See "Control Manager Result Codes" (page 824).

Discussion

The horizontal or vertical orientation of a separator line is determined automatically based on the relative height and width of its control bounds.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`HISeparator.h`

CreateSliderControl

Creates a slider control.

```
OSStatus CreateSliderControl (
   WindowRef window,
   const Rect *boundsRect,
   SInt32 value,
   SInt32 minimum,
   SInt32 maximum,
   ControlSliderOrientation orientation,
   UInt16 numTickMarks,
   Boolean liveTracking,
   ControlActionUPP liveTrackingProc,
   ControlRef *outControl
);
```

Parameters

window

The window that is to contain the control. This parameter may be `NULL` in Mac OS X v10.3 and later.

boundsRect
The bounds of the control in the window's local coordinates.

value
The initial value of the control.

minimum
The minimum value of the control.

maximum
The maximum value of the control.

orientation
The orientation of the control. For possible values, see "Control Slider Orientation Constants" (page 754).

numTickMarks
The number of tick marks the slider control is to have.

liveTracking
A Boolean whose value is `true` to enable live tracking for the control; otherwise, `false`.

liveTrackingProc
If the value of the `liveTracking` parameter is `true`, a `ControlActionUPP` callback is to be called as the control live tracks. This callback is called repeatedly as the slider is moved during tracking.

outControl
On return, `outControl` points to the new control. For a description of this data type, see ControlRef (page 709).

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Discussion
Mac OS X has a "Scroll to here" option in the General pane of System Preferences that allows users to click in the page up or page down regions of a slider and have the indicator jump directly to the clicked position, which alters the value of the slider and moves any associated content appropriately. As long as the mouse button is held down, the click is treated as though the user clicked the indicator.

If you want the sliders in your application to work with the "Scroll to here" option, you must do the following:

1. Create live-tracking sliders, not sliders that show a "ghost" thumb when you click on it. You can request live-tracking sliders by passing `true` in the `liveTracking` parameter to `CreateSliderControl`. If you create sliders with `NewControl` (page 634), use the `kControlSliderLiveFeedback` variant.

2. Write an appropriate ControlActionProc and associate it with your slider by calling `SetControlAction` (page 649). This allows your application to update its content appropriately when the live-tracking slider is clicked.

3. When calling`HandleControlClick` (page 613) or `TrackControl` (page 671) TrackControl, pass -1 in the action proc parameter. This is a request for the Control Manager to use the action proc you associated with your control in step 2. If you rely on the standard window event handler to do your control tracking, this step is handled for you automatically.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
HISlider.h

CreateStaticTextControl

Creates a new static text control.

```
OSStatus CreateStaticTextControl (
   WindowRef window,
   const Rect *boundsRect,
   CFStringRef text,
   const ControlFontStyleRec *style,
   ControlRef *outControl
);
```

Parameters

window
The window in which the control is to be placed. This parameter may be NULL in Mac OS X v10.3 and later.

boundsRect
The bounds of the control in the window's local coordinates.

text
The text of the control. May be NULL.

style
The control's font style, size, color, and so on. May be NULL.

outControl
On return, the new control. For a description of this data type, see ControlRef (page 709).

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
HITextViews.h

CreateTabsControl

Creates a tabs control.

```
OSStatus CreateTabsControl (
   WindowRef window,
   const Rect *boundsRect,
   ControlTabSize size,
   ControlTabDirection direction,
   UInt16 numTabs,
   const ControlTabEntry *tabArray,
   ControlRef *outControl
);
```

Parameters

window

The window in which the control is to be placed. This parameter may be `NULL` in Mac OS X v10.3 and later.

boundsRect

The bounds of the control in the window's local coordinates.

size

The control tab size. See "`Control Tab Size Constants`" (page 754) for possible values.

direction

The control tab direction. See "`Control Tab Direction Constants`" (page 754) for possible values.

numTabs

The initial number of tabs.

tabArray

Information about each tab. There must be the same number of entries as specified by the `numTabs` parameter.

outControl

On return, the new control. For a description of this data type, see ControlRef (page 709).

Return Value

A result code. See "Control Manager Result Codes" (page 824).

Discussion

If you want to customize the accessibility information provided for individual tabs of a tabs control, such as by handling various `kEventClassAccessibility` Carbon Events and by calling `HIObjectSetAuxiliaryAccessibilityAttribute`, you need to know how to build or interpret AXUIElement reference that represent individual tabs. The AXUIElement representing an individual tab must be constructed using the tab control's `ControlRef` and the `UInt64` identifier of the one-based index of the tab to which the element refers. A `UInt64` identifier of 0 represents the tabs control as a whole. You cannot interpret or create tab control elements whose identifiers are greater than the count of tabs in the tabs control.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`HITabbedView.h`

CreateUserPaneControl

Creates a user pane control.

```
OSStatus CreateUserPaneControl (
   WindowRef window,
   const Rect *boundsRect,
   UInt32 features,
   ControlRef *outControl
);
```

Parameters

window

The window in which the control is to be placed. This parameter may be `NULL` in Mac OS X v10.3 and later.

boundsRect

The bounds of the control in the window's local coordinates.

features

The user pane features with which the user pane is to be created. For possible constants, see "Control Features Constants" (page 741).

outControl

On return, the new control. For a description of this data type, see ControlRef (page 709).

Return Value

A result code. See "Control Manager Result Codes" (page 824).

Discussion

User panes have two primary purposes: to allow easy implementation of a custom control by the developer, and to provide a generic container for embedding other controls.

In Carbon, with the advent of Carbon-event-based controls, you may find it easier to write a new control from scratch than to customize a user pane control. The set of callbacks provided by the user pane will not be extended to support new Control Manager features; instead, you should just write a real control.User panes do not, by default, support embedding. If you try to embed a control into a user pane, you will get the `errControlIsNotEmbedder`. You can make a user pane support embedding by passing the `kControlSupportsEmbedding` flag in the `features` parameter when you create the control.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

HID Calibrator

Declared In

`HIContainerViews.h`

CreateWindowHeaderControl

Creates a window header control.

```
OSStatus CreateWindowHeaderControl (
   WindowRef window,
   const Rect *boundsRect,
   Boolean isListHeader,
   ControlRef *outControl
);
```

Parameters

window

The window in which the control is to be placed. This parameter may be `NULL` in Mac OS X v10.3 and later.

boundsRect

The bounds of the control in the window's local coordinates.

isListHeader

A Boolean whose value is `true` if the control should have an appropriate appearance to be the header of a list; otherwise, `false`.

outControl

On return, the new control. For a description of this data type, see ControlRef (page 709).

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`HIContainerViews.h`

DeactivateControl

Deactivates a control and any latent embedded controls.

```
OSErr DeactivateControl (
   ControlRef inControl
);
```

Parameters

inControl

A handle to the control to deactivate. If you pass a window's root control, `DeactivateControl` deactivates all controls in that window.

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Discussion
The `DeactivateControl` function should be called instead of `HiliteControl` to deactivate a specified control and its latent embedded controls.

An embedded control is considered latent when it is deactivated or hidden due to its embedder control being deactivated or hidden. If you activate a latent embedded control whose embedder is deactivated, the embedded control becomes latent until the embedder is activated. However, if you deactivate a latent embedded control, it will not be activated when its embedder is activated.

If a control definition function supports activate events, it will receive a `kControlMsgActivate` message before redrawing itself in its inactive state.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
BSDLLCTest

Declared In
`Controls.h`

DisableControl

Disables a control.

```
OSStatus DisableControl (
   ControlRef inControl
);
```

Parameters

inControl
The control to disable. For a description of this data type, see ControlRef (page 709).

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
QTCarbonShell

Declared In
`Controls.h`

DisposeControl

Decrements a control's reference count and destroys it if the reference count becomes 0.

```
void DisposeControl (
   ControlRef theControl
);
```

Parameters

theControl
The control you want to dispose of. For a description of this data type, see ControlRef (page 709).

Discussion
The `DisposeControl` function reduces the control's reference count and, if the reference count becomes 0, releases the memory occupied by the control structure and any data structures associated with the control. Before destroying the control, DisposeControl removes the control (and any embedded controls it may possess) from the screen and deletes the control from the window's control list.

To destroy all of the controls from a window you want to keep, use the function `KillControls` (page 632). If an embedding hierarchy is present, passing the root control to the `DisposeControl` function is the effectively the same as calling `KillControls` (page 632). In that situation, `DisposeControl` disposes of the controls embedded within a control before disposing of the container control.

You should use `DisposeControl` when you want to retain the window but remove one of its controls. The Window Manager functions `CloseWindow` and `DisposeWindow` automatically remove all controls associated with the window and release the memory the controls occupy.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
QTCarbonShell

Declared In
`Controls.h`

DisposeControlActionUPP

Disposes of a control action UPP.

```
void DisposeControlActionUPP (
   ControlActionUPP userUPP
);
```

Parameters
userUPP
The UPP that is to be disposed of.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Controls.h`

DisposeControlCNTLToCollectionUPP

Disposes of a CNLT to collection UPP.

```
void DisposeControlCNTLToCollectionUPP (
   ControlCNTLToCollectionUPP userUPP
);
```

Parameters

userUPP

The UPP that is to be disposed of.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
Controls.h

DisposeControlColorUPP

Not recommended

```
void DisposeControlColorUPP (
   ControlColorUPP userUPP
);
```

Carbon Porting Notes

Instead of specifying a callback to redraw your background, you should make the background a control and then embed your other controls within it.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
Controls.h

DisposeControlDefUPP

Disposes of a control definition UPP. (Deprecated in Mac OS X v10.5. Use a custom HIView to draw a custom control.)

```
void DisposeControlDefUPP (
   ControlDefUPP userUPP
);
```

Parameters

userUPP

The UPP that is to be disposed of.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Controls.h`

DisposeControlEditTextValidationUPP

Disposes of an edit text validation UPP.

```
void DisposeControlEditTextValidationUPP (
   ControlEditTextValidationUPP userUPP
);
```

Parameters

userUPP
The UPP that is to be disposed of.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HITextViews.h`

DisposeControlKeyFilterUPP

Disposes of a key filter UPP.

```
void DisposeControlKeyFilterUPP (
   ControlKeyFilterUPP userUPP
);
```

Parameters

userUPP
The UPP that is to be disposed of.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Controls.h`

DisposeControlUserPaneActivateUPP

Disposes of a user pane activate UPP.

```
void DisposeControlUserPaneActivateUPP (
   ControlUserPaneActivateUPP userUPP
);
```

Parameters

userUPP
The UPP that is to be disposed of.

Availability
Available in Mac OS X v10.0 and later.

Declared In
HIContainerViews.h

DisposeControlUserPaneBackgroundUPP

Disposes of a user pane background UPP.

```
void DisposeControlUserPaneBackgroundUPP (
   ControlUserPaneBackgroundUPP userUPP
);
```

Parameters

userUPP
The UPP that is to be disposed of.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
HIContainerViews.h

DisposeControlUserPaneDrawUPP

Disposes of a user pane draw UPP.

```
void DisposeControlUserPaneDrawUPP (
   ControlUserPaneDrawUPP userUPP
);
```

Parameters

userUPP
The UPP that is to be disposed of.

Availability
Available in Mac OS X v10.0 and later.

Declared In
HIContainerViews.h

DisposeControlUserPaneFocusUPP

Disposes of a user pane focus UPP.

```
void DisposeControlUserPaneFocusUPP (
   ControlUserPaneFocusUPP userUPP
);
```

Parameters

userUPP
The UPP that is to be disposed of.

Availability
Available in Mac OS X v10.0 and later.

Declared In
HIContainerViews.h

DisposeControlUserPaneHitTestUPP

Disposes of a user pane hit test UPP.

```
void DisposeControlUserPaneHitTestUPP (
   ControlUserPaneHitTestUPP userUPP
);
```

Parameters

userUPP
The UPP that is to be disposed of.

Availability
Available in Mac OS X v10.0 and later.

Declared In
HIContainerViews.h

DisposeControlUserPaneIdleUPP

Disposes of a user pane idle UPP.

```
void DisposeControlUserPaneIdleUPP (
   ControlUserPaneIdleUPP userUPP
);
```

Parameters

userUPP
The UPP that is to be disposed of.

Availability
Available in Mac OS X v10.0 and later.

Declared In
HIContainerViews.h

DisposeControlUserPaneKeyDownUPP

Disposes of a user pane key down UPP.

```
void DisposeControlUserPaneKeyDownUPP (
   ControlUserPaneKeyDownUPP userUPP
);
```

Parameters

userUPP
The UPP that is to be disposed of.

Availability
Available in Mac OS X v10.0 and later.

Declared In
HIContainerViews.h

DisposeControlUserPaneTrackingUPP

Disposes of a user pane tracking UPP.

```
void DisposeControlUserPaneTrackingUPP (
   ControlUserPaneTrackingUPP userUPP
);
```

Parameters

userUPP
The UPP that is to be disposed of.

Availability
Available in Mac OS X v10.0 and later.

Declared In
HIContainerViews.h

DisposeEditUnicodePostUpdateUPP

Disposes of an edit unicode post update UPP.

```
void DisposeEditUnicodePostUpdateUPP (
   EditUnicodePostUpdateUPP userUPP
);
```

Parameters

userUPP
The UPP that is to be disposed of.

Availability
Available in Mac OS X v10.0 and later.

Declared In
HITextViews.h

DragControl

Draws and moves an outline of a control or its indicator while the user drags it. (**Deprecated.** Use Drag Manager functions if you want drag-and-drop support for controls. See *Drag Manager Reference*.)

Not recommended

```
void DragControl (
   ControlRef theControl,
   Point startPoint,
   const Rect *limitRect,
   const Rect *slopRect,
   DragConstraint axis
);
```

Parameters

theControl

A handle to the control to drag. For a description of this data type, see ControlRef (page 709).

startPoint

The location of the cursor at the time the mouse button was first pressed, in global coordinates. Your application retrieves this point from the `where` field of the event structure.

limitRect

A pointer to a rectangle—whose coordinates should normally coincide with or be contained in the window's content region—delimiting the area in which the user can drag the control's outline.

slopRect

A pointer to a rectangle that allows some extra space for the user to move the mouse while still constraining the control within the rectangle specified in the `limitRect` parameter.

axis

The axis along which the user may drag the control's outline. Specify the `axis` using one of the following values: `noConstraint` (no constraint), `hAxisOnly` (drag along horizontal axis only), `vAxisOnly` (drag along vertical axis only).

Discussion

The `DragControl` function moves a dotted outline of a control, such as a scroll box, around the screen, following the movements of the cursor until the user releases the mouse button. When the user releases the mouse button, `DragControl` moves the control to the new location.

The function `TrackControl` (page 671) automatically calls the `DragControl` function as appropriate; when you use `TrackControl`, you don't need to call `DragControl`.

Before tracking the cursor, `DragControl` calls the control definition function. If you define your own control definition function, you can specify custom dragging behavior.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Controls.h`

Draw1Control

Draws a control and any embedded controls that are currently visible in the specified window.

```
void Draw1Control (
   ControlRef theControl
);
```

Parameters

theControl

A handle to the control to draw. For a description of this data type, see ControlRef (page 709).

Discussion

Although you should generally use the functionUpdateControls (page 672) to update controls, you can use the DrawOneControl function to update a single control. If an embedding hierarchy exists and the control passed in has embedded controls, DrawOneControl draws the control and embedded controls. If the root control for a window is passed in, the result is the same as if DrawControls was called.

If you are using compositing mode, you generally do not need to call Draw1Control. If you call Draw1Control in compositing mode, keep in mind that it draws the specified control as well as all other controls that intersect the control.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

Controls.h

DrawControlInCurrentPort

Draws a control in the current graphics port.

```
void DrawControlInCurrentPort (
   ControlRef inControl
);
```

Parameters

inControl

A handle to the control to draw. For a description of this data type, see ControlRef (page 709).

Discussion

Typically, controls are automatically drawn in their owner's graphics port with the functions DrawControls (page 581), Draw1Control (page 580), and UpdateControls (page 672). DrawControlInCurrentPort permits easy offscreen control drawing and printing. All standard system controls support this function.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

Controls.h

DrawControls

Draws all controls currently visible in the specified window.

```
void DrawControls (
   WindowRef theWindow
);
```

Parameters

theWindow

A pointer to a window whose controls you want to display.

Discussion

Because the `UpdateControls` function redraws only those controls that need updating, your application should generally use it instead of `DrawControls` when you receive an update event for a window that contains controls. You should typically call either `DrawControls` or `UpdateControls` after calling the Window Manager function `BeginUpdate` and before calling `EndUpdate`.

While the Dialog Manager automatically draws and updates controls in alert boxes and dialog boxes, Window Manager functions such as `SelectWindow`, `ShowWindow`, and `BringToFront` do not automatically update the window's controls.

When the Appearance Manager is not available, the `DrawControls` function draws all controls currently visible in the specified window in reverse order of creation; thus, in case of overlapping controls, the control created first appears frontmost in the window. If you only wish to draw controls in need of update, call `UpdateControls` (page 672) instead.

Note that `DrawControls` generally should not be called if you are using compositing mode.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

CarbonSketch

HID Config Save

HID Explorer

Declared In

`Controls.h`

DumpControlHierarchy

Writes a textual representation of the control hierarchy for a specified window into a file.

```
OSErr DumpControlHierarchy (
   WindowRef inWindow,
   const FSSpec *inDumpFile
);
```

Parameters

inWindow

A pointer to the window whose control hierarchy you wish to examine.

inDumpFile

A pointer to a file specification in which to place a text description of the window's control hierarchy.

Return Value

A result code. See "Control Manager Result Codes" (page 824).

Discussion
The `DumpControlHierarchy` function places a text listing of the current control hierarchy for the window specified into the specified file, overwriting any existing file. If the specified window does not contain a control hierarchy, `DumpControlHierarchy` notes this in the text file. This function is useful for debugging embedding-related problems.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`HIToolboxDebugging.h`

EmbedControl

Embeds one control inside another.

```
OSErr EmbedControl (
   ControlRef inControl,
   ControlRef inContainer
);
```

Parameters

inControl
The control that is to be embedded. For a description of this data type, see ControlRef (page 709).

inContainer
The control in which the control specified by `inControl` is to be is to embedded. For a description of this data type, see ControlRef (page 709).

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Discussion
Establishing an embedding hierarchy can be accomplished in two steps: creating a root control and embedding controls within it.

To embed controls in a window, you must create a root control for that window. The root control is the container for all other window controls. You create the root control in one of two ways—by calling the `CreateRootControl` (page 562) function or by setting the appropriate dialog flag. The root control can be retrieved by calling `GetRootControl` (page 611).

The root control is implemented as a user pane control. You can attach any application-defined user pane functions to the root control to perform actions such as hit testing, drawing, handling keyboard focus, erasing to the correct background, and processing idle and keyboard events.

Once you have created a root control, newly created controls will automatically be embedded in the root control when you call `NewControl` (page 634) or `GetNewControl` (page 610). You can specify that a specific control be embedded into another by calling `EmbedControl`.

Note that an embedding hierarchy must be established before your application calls the `EmbedControl` function. If the specified control does not support embedding or there is no root control in the owning window, an error is returned. Prior to Mac OS X, if the control you wish to embed is in a different window from the embedder control, an error is returned. On Mac OS X, however, you can use `EmbedControl` to

move a control from one window to another. On Mac OS X v.10.0 and v.10.1, you can move all controls except for the edit text and unicode edit text controls. Support for the edit text controls is available in Mac OS X v.10.2 and later.

By acting on an embedder control, you can move, disable, or hide groups of items. For example, you can use a blank user pane control as the embedder control for all items in a particular "page" of a tab control. After creating as many user panes as you have tabs, you can hide one and show the next when a tab is clicked. All the controls embedded in the user pane will be hidden and shown automatically when the user pane is hidden and shown.

In addition to calling `CreateRootControl`, you can establish an embedding hierarchy in a dialog box by either setting the feature bit `kDialogFlagsUseControlHierarchy` in the extended dialog resource or passing it in the `inFlags` parameter of the Dialog Manager function `NewFeaturesDialog`. An embedding hierarchy can be created in an alert box by setting the `kAlertFlagsUseControlHierarchy` bit in the extended alert resource. It is important to note that a preexisting alert or dialog item will become a control if it is in an alert or dialog box that now uses an embedding hierarchy.

The embedding hierarchy enforces drawing order by drawing the embedding control before its embedded controls. Using an embedding hierarchy also enforces orderly hit-testing, since it performs an "inside-out" hit test to determine the most deeply nested control that is hit by the mouse. An embedding hierarchy is also necessary for controls to make use of keyboard focus, the default focusing order for which is a linear progression that uses the order the controls were added to the window. For more details on keyboard focus, see "Handling Keyboard Focus".

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
QTCarbonShell

Declared In
`Controls.h`

EnableControl

Enables a control.

```
OSStatus EnableControl (
   ControlRef inControl
);
```

Parameters

theControl
The control that is to be enabled.

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
QTCarbonShell

Declared In
`Controls.h`

FindControl

Obtains the location of a mouse-down event in a control.

```
ControlPartCode FindControl (
   Point testPoint,
   WindowRef theWindow,
   ControlRef *theControl
);
```

Parameters

`testPoint`

A point, specified in coordinates local to the window, where the mouse-down event occurred. Before calling `FindControl`, use the `GlobalToLocal` function to convert the point stored in the `where` field of the event structure (which describes the location of the mouse-down event) to coordinates local to the window.

`theWindow`

A pointer to the window in which the mouse-down event occurred. Pass the window pointer returned by the `FindWindow` function.

`theControl`

A pointer to a control handle. On output, `FindControl` returns a handle to the control in which the mouse-down event occurred or `NULL` if the point was not over a control. For a description of this data type, see ControlRef (page 709).

Return Value
The control part code of the control in which the mouse-down event occurred; see "Control Meta Part Code Constants" (page 790), "Control Part Code Constants " (page 748), and" Control State Part Code Constants" (page 751). For a description of this data type, see ControlPartCode (page 708).

Discussion
The `FindControl` function is not recommended when the Appearance Manager is available. When the Appearance Manager is available, you should call `FindControlUnderMouse` (page 586) to determine the location of a mouse-down event in a control. `FindControlUnderMouse` will return a handle to the control even if no part was hit and can determine whether a mouse-down event has occurred even if the control is deactivated, while `FindControl` does not.

If the Appearance Manager is not available, then, when a mouse-down event occurs, your application can call `FindControl` after using the Window Manager function `FindWindow` to ascertain that a mouse-down event has occurred in the content region of a window containing controls.

When the user presses the mouse button while the cursor is in a visible, active control, `FindControl` returns as its function result a part code identifying the control's part the function also returns a handle to the control in the parameter `theControl`.

If the mouse-down event occurs in an invisible or inactive control, or if it occurs outside a control, `FindControl` sets the value referenced through `theControl` to `NULL` and returns 0 as its function result.

The `FindControl` function also returns `NULL` in the value referenced through the parameter `theControl` and 0 as its function result if the window is invisible or if it doesn't contain the given point. (However, `FindWindow` won't return a window pointer to an invisible window or to one that doesn't contain the point where the mouse-down event occurred. As long as you call `FindWindow` before `FindControl`, this situation won't arise.)

After using `FindControl` to determine that a mouse-down event has occurred in a control, you typically call the function `TrackControl` (page 671) to follow and respond to the cursor movements in that control, and then to determine in which part of the control the mouse-up event occurs.

The pop-up control definition function does not define part codes for pop-up menus. Instead, your application should store the handles for your pop-up menus when you create them. Your application should then test the handles you store against the handles returned by `FindControl` before responding to users' choices in pop-up menus.

The Dialog Manager automatically calls `FindControl` and `TrackControl` for mouse-down events inside controls of alert boxes and dialog boxes.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

FindControlUnderMouse

Obtains the location of a mouse-down event in a control.

```
ControlRef FindControlUnderMouse (
   Point inWhere,
   WindowRef inWindow,
   ControlPartCode *outPart
);
```

Parameters

inWhere
A point, specified in coordinates local to the window, where the mouse-down event occurred. Before calling `FindControlUnderMouse`, use the QuickDraw `GlobalToLocal` function to convert the point stored in the `where` field of the event structure (which describes the location of the mouse-down event) to coordinates local to the window.

inWindow
A pointer to the window in which the mouse-down event occurred.

outPart
On input, a pointer to a signed 16-bit integer value. On return, the value is set to the part code of the control part that was selected; see "Control Part Code Constants " (page 748), "Control Part Code Constants " (page 748), and " Control State Part Code Constants" (page 751).

Return Value
A handle to the control that was selected. If the mouse-down event did not occur over a control part, `FindControlUnderMouse` returns `NULL`. For a description of this data type, see ControlRef (page 709).

Discussion
You should call the `FindControlUnderMouse` function instead of `FindControl` (page 585) to determine whether a mouse-down event occurred in a control, particularly if an embedding hierarchy is present. `FindControlUnderMouse` will return a handle to the control even if no part was hit and can determine whether a mouse-down event has occurred even if the control is deactivated, while `FindControl` does not.

When a mouse-down event occurs, your application should call `FindControlUnderMouse` after using the Window Manager function `FindWindow` to ascertain that a mouse-down event has occurred in the content region of a window containing controls.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
CarbonSketch

Declared In
`Controls.h`

GetBestControlRect

Obtains a control's optimal size and text placement.

```
OSErr GetBestControlRect (
   ControlRef inControl,
   Rect *outRect,
   SInt16 *outBaseLineOffset
);
```

Parameters

inControl
A handle to the control to be examined.

outRect
On input, a pointer to an empty rectangle (0, 0, 0, 0). On return, the rectangle is set to the optimal size for the control. If the control doesn't support getting an optimal size rectangle, the control's bounding rectangle is passed back.

outBaseLineOffset
On input, a pointer to a signed 16-bit integer value. On return, the value is set to the offset from the bottom of control to the base of the text (usually a negative value). If the control doesn't support optimal sizing or has no text, 0 is passed back.

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Discussion
You can call the `GetBestControlRect` function to automatically position and size controls in accordance with human interface guidelines. This function is particularly helpful in determining the correct placement of control text whose length is not known until run-time. For example, the `StandardAlert` function uses `GetBestControlRect` to automatically size and position buttons in a newly created alert box.

Availability
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
`Controls.h`

GetBevelButtonContentInfo

Gets the content information for a bevel button.

```
OSErr GetBevelButtonContentInfo (
   ControlRef inButton,
   ControlButtonContentInfoPtr outContent
);
```

Parameters

inButton
The control reference for the button to query.

outContent
A value of type `ControlButtonContentInfoPtr` for the bevel button's content information.

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`HIButtonViews.h`

GetBevelButtonMenuHandle

Gets the menu handle for a bevel button.

```
OSErr GetBevelButtonMenuHandle (
   ControlRef inButton,
   MenuHandle *outHandle
);
```

Parameters

inButton
The control reference for the button to query.

outHandle
A pointer to the menu handle.

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
HIButtonViews.h

GetBevelButtonMenuValue

Gets the value of a bevel button menu.

```
OSErr GetBevelButtonMenuValue (
   ControlRef inButton,
   MenuItemIndex *outValue
);
```

Parameters

inButton
The control reference for the button to query.

outValue
A pointer to the value of the bevel button menu.

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
HIButtonViews.h

GetControl32BitMaximum

Obtains the maximum setting of a control.

```
SInt32 GetControl32BitMaximum (
   ControlRef theControl
);
```

Parameters

theControl
A handle to the control whose maximum setting you wish to obtain. For a description of this data type, see ControlRef (page 709).

Return Value
The maximum setting of the control.

Discussion
Your application may use the `GetControl32BitMaximum` function to obtain a 32-bit value previously set with the function `SetControl32BitMaximum` (page 647).

If your application uses a 32-bit control maximum value, it should not attempt to obtain this value by calling the pre–Mac OS 8.5 function `GetControlMaximum` because the 16-bit value that is returned does not accurately reflect the current 32-bit control value.

Availability
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code
HID Calibrator
HID Explorer

Declared In
`Controls.h`

GetControl32BitMinimum

Obtains the minimum setting of a control.

```
SInt32 GetControl32BitMinimum (
   ControlRef theControl
);
```

Parameters

theControl
A handle to the control whose minimum setting you wish to obtain. For a description of this data type, see ControlRef (page 709).

Return Value
The minimum setting of the control.

Discussion
Your application may use the `GetControl32BitMinimum` function to obtain a 32-bit value previously set with the function `SetControl32BitMinimum` (page 648).

If your application uses a 32-bit control minimum value, it should not attempt to obtain this value by calling the pre–Mac OS 8.5 function `GetControlMinimum` because the 16-bit value that is returned does not accurately reflect the current 32-bit control value.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
HID Calibrator
HID Explorer

Declared In
`Controls.h`

GetControl32BitValue

Obtains the current setting of a control.

```
SInt32 GetControl32BitValue (
   ControlRef theControl
);
```

Parameters

theControl

A handle to the control whose current setting you wish to obtain.

Return Value

The current setting of the control.

Discussion

Your application may use the `GetControl32BitValue` function to obtain a 32-bit value previously set with the function `SetControl32BitValue` (page 648).

If your application uses a 32-bit control value, it should not attempt to obtain this value by calling the pre–Mac OS 8.5 function `GetControlValue` because the 16-bit value that is returned does not accurately reflect the current 32-bit control value.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

HID Calibrator

HID Explorer

Declared In

`Controls.h`

GetControlAction

Returns a pointer to the action function associated with a control structure.

```
ControlActionUPP GetControlAction (
   ControlRef theControl
);
```

Parameters

theControl

A handle to a control.

Return Value

The action function associated with the control. The action function is called by the `HandleControlClick` and `TrackControl` functions if you set the `InAction` parameter to `(ControlActionUPP)-1`. See `ControlActionProcPtr` (page 675) for an example of an action function.

Discussion

The action function returned by the `GetControlAction` function defines an action to take in response to a mouse button being held down while the cursor is in the control. An action function is usually specified in the `InAction` parameter of the functions `HandleControlClick` (page 613) and `TrackControl` (page 671). You can use the function `SetControlAction` (page 649) to change the action function.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
`Controls.h`

GetControlBounds

Gets the bounds of a control.

```
Rect * GetControlBounds (
   ControlRef control,
   Rect *bounds
);
```

Parameters

control

The control to query. For a description of this data type, see ControlRef (page 709).

bounds

On input, a pointer to a QuickDraw rectangle. On output, the rectangle contains the bounds of the control in local coordinates.

Return Value
A pointer to the rectangle passed in the `bounds` parameter.

Discussion
When called in a composited window, this function returns the view's frame, which is equivalent to calling `HIViewGetFrame`.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
CarbonSketch
HID Explorer

Declared In
`Controls.h`

GetControlByID

Finds a control in a window by its unique ID.

```
OSStatus GetControlByID (
   WindowRef inWindow,
   const ControlID *inID,
   ControlRef *outControl
);
```

Parameters

inWindow

The window to query.

inID

The control ID.

outControl

A pointer to a value of type `ControlRef` that, on output, is filled in with the control reference for the control specified by inID. For a description of this data type, see ControlRef (page 709).

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Discussion
As of Mac OS X v10.3, this function is superseded by the `HIViewFindByID` function, which is preferred over the `GetControlByID` function. The first parameter to the `HIViewFindByID` function is a view and not a window, so you can start the search at any point in the hierarchy.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
BSDLLCTest
CarbonSketch
HID Config Save
HID Explorer
QTCarbonShell

Declared In
`Controls.h`

GetControlClickActivation

Gets the control's preferred behavior for responding to particular click.

```
OSStatus GetControlClickActivation (
   ControlRef inControl,
   Point inWhere,
   EventModifiers inModifiers,
   ClickActivationResult *outResult
);
```

Parameters

inControl

inWhere

The location at which the control was clicked.

inModifiers

Information from the `modifiers` field of the event structure specifying the state of the modifier keys and the mouse button at the time the event was posted. .

outResult

A pointer to a value of type `ClickActivationResult` containing the result. For possible values, see "Click Activation Constants" (page 780).

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Discussion
Some complex controls, such as Data Browser, require proper sequencing of window activation and click processing. In some cases, the control might want the window to be left inactive yet still handle the click, or vice- versa. This function lets a control client ask the control how it wants to behave for a particular click.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

GetControlCommandID

Gets the command ID for a control.

```
OSStatus GetControlCommandID (
   ControlRef inControl,
   UInt32 *outCommandID
);
```

Parameters

inControl

outCommandID

A pointer to the command ID.

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

GetControlData

Obtains control-specific data.

```
OSErr GetControlData (
   ControlRef inControl,
   ControlPartCode inPart,
   ResType inTagName,
   Size inBufferSize,
   void *inBuffer,
   Size *outActualSize
);
```

Parameters

inControl

A handle to the control to be examined.

inPart

"Control Meta Part Code Constants" (page 790)The part code of the control part from which data is to be obtained; see , "Control Part Code Constants " (page 748), and " Control State Part Code Constants" (page 751). Passing `kControlEntireControl` indicates that either the control has no parts or the data is not tied to any specific part of the control. For a description of this data type, see ControlPartCode (page 708).

inTagName

A constant representing the control-specific data you wish to obtain see the data tag constants in the "Control Manager Constants" (page 719) section.

inBufferSize

The size (in bytes) of the data pointed to by the `inBuffer` parameter. For variable-length control data, pass the value returned in the `outMaxSize` parameter of `GetControlDataSize` (page 596) in the `inBufferSize` parameter. The number of bytes must match the actual data size.

inBuffer

On input, a pointer to a buffer allocated by your application. On return, the buffer contains a copy of the control-specific data. If you pass `NULL` on input, it is equivalent to calling `GetControlDataSize` (page 596). The actual size of the control-specific data will be returned in the `outActualSize` parameter. For variable-length data, the number of bytes must match the actual data size.

outActualSize

On input, a pointer to a `Size` value. On return, the value is set to the actual size of the data. You can pass `NULL` if you don't care about this value.

Return Value
A result code. See "Control Manager Result Codes" (page 824). The result code `errDataNotSupported` indicates that the `inTagName` parameter is not valid.

Discussion
The `GetControlData` function will only copy the amount of data specified in the `inBufferSize` parameter, but will tell you the actual size of the buffer so you will know if the data was truncated.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
BSDLLCTest
CarbonCocoa_PictureCursor
HID Explorer
QTCarbonShell

Declared In
`Controls.h`

GetControlDataHandle

Obtains a handle to control-specific data. (**Deprecated.** Use custom HIViews instead of custom CDEFs. See *HIView Programming Guide*.)

```
Handle GetControlDataHandle (
   ControlRef control
);
```

Return Value
A handle to control-specific data.

Discussion
The control data handle is for control-specific data used by a control's implementation. The control data handle is set by calling `SetControlDataHandle` (page 653).

In general, you should not attempt to interpret the contents of this handle if you did not implement the control yourself. For controls that are provided by the operating system, the format of the data handle may change from one release of the operating system to the next.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

GetControlDataSize

Obtains the size of a control's tagged data.

```
OSErr GetControlDataSize (
   ControlRef inControl,
   ControlPartCode inPart,
   ResType inTagName,
   Size *outMaxSize
);
```

Parameters

inControl
A handle to the control to be examined.For a description of this data type, see ControlRef (page 709).

inPart
The part code of the control part with which the data is associated; see"Control Meta Part Code Constants" (page 790), "Control Part Code Constants " (page 748), and " Control State Part Code Constants" (page 751). Passing `kControlEntireControl` indicates that either the control has no parts or the data is not tied to any specific part of the control.

inTagName
A constant representing the control-specific data whose size is to be obtained see the data tag constants in the "Control Manager Constants" (page 719) section.

outMaxSize
On input, a pointer to a `Size` value. On return, the value is set to the size (in bytes) of the control's tagged data. This value should be passed to `SetControlData` (page 652) and `GetControlData` (page 594) to allocate a sufficiently large buffer for variable-length data.

Return Value
A result code. See "Control Manager Result Codes" (page 824). The result code `errDataNotSupported` indicates that the `inTagName` parameter is not valid.

Discussion
Pass the value returned in the `outMaxSize` parameter of `GetControlDataSize` in the `inBufferSize` parameter of `SetControlData` (page 652) and `GetControlData` (page 594) to allocate an adequate buffer for variable-length data.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

GetControlFeatures

Obtains the features a control supports.

Not recommended

```
OSErr GetControlFeatures (
   ControlRef inControl,
   UInt32 *outFeatures
);
```

Parameters

inControl
A handle to the control to be examined. For a description of this data type, see ControlRef (page 709).

outFeatures
On input, a pointer to an unsigned 32-bit integer value. On return, the value contains a bit field specifying the features the control supports. For a list of the features a control may support, see `ControlDefProcPtr` (page 677).

Return Value
A result code. See "Control Manager Result Codes" (page 824). The result code `errMsgNotSupported` indicates that the control does not support Appearance-compliant features.

Discussion
The `GetControlFeatures` function obtains the Appearance-compliant features a control definition function supports, in response to a `kControlMsgGetFeatures` message.

Carbon Porting Notes

Some feature bits may not be relevant when using Carbon event-based messages.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

GetControlHilite

Gets the highlight status of a control.

```
UInt16 GetControlHilite (
   ControlRef control
);
```

Parameters

control

The control to query. For a description of this data type, see ControlRef (page 709).

Return Value

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
Controls.h

GetControlID

Gets the control ID for a control.

```
OSStatus GetControlID (
   ControlRef inControl,
   ControlID *outID
);
```

Parameters

inControl

The control to query. For a description of this data type, see ControlRef (page 709).

outID

A pointer to a value of type `ControlID` that, on return, contains the control ID of the control specified by `inControl`.

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
CarbonSketch

Declared In
Controls.h

GetControlKind

Returns the kind of the given control.

```
OSStatus GetControlKind (
   ControlRef inControl,
   ControlKind *outControlKind
);
```

Parameters

inControl

The control to query. For a description of this data type, see ControlRef (page 709).

outControlKind

On successful exit, this will contain the control signature and kind. See `ControlDefinitions.h` for the kinds of each system control. For a description of this data type, see ControlKind (page 707).

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Discussion
`GetControlKind` allows you to query the kind of any control.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
HID Calibrator

Declared In
`Controls.h`

GetControlMaximum

Obtains a control's maximum setting. (**Deprecated.** Use `GetControl32BitMaximum` (page 589) instead.)

Not recommended

```
SInt16 GetControlMaximum (
   ControlRef theControl
);
```

Parameters

theControl

A handle to the control whose maximum value you wish to determine. For a description of this data type, see ControlRef (page 709).

Return Value
The specified control's maximum setting.

Discussion
When you create a control, you specify an initial maximum setting either in the control resource or in the `max` parameter of the function `NewControl` (page 634). You can change the maximum setting by using the function `SetControlMaximum` (page 655).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
CarbonSketch

Declared In
`Controls.h`

GetControlMinimum

Obtains a control's minimum setting. (**Deprecated.** Use `GetControl32BitMinimum` (page 590) instead.)

Not recommended

```
SInt16 GetControlMinimum (
   ControlRef theControl
);
```

Parameters

theControl
A handle to the control whose minimum value you wish to determine.

Return Value
The specified control's minimum setting.

Discussion
When you create a control, you specify an initial minimum setting either in the control resource or in the `min` parameter of the function `NewControl` (page 634). You can change the minimum setting by using the function `SetControlMinimum` (page 656).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
CarbonSketch

Declared In
`Controls.h`

GetControlOwner

Returns the window to which a control is bound.

```
WindowRef GetControlOwner (
   ControlRef control
);
```

Parameters

control
The control to query. For a description of this data type, see ControlRef (page 709).

Return Value
The window reference to which the control is bound, or `NULL` if the control is not bound to a window.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
CarbonSketch

Declared In
`Controls.h`

GetControlPopupMenuHandle

Gets the menu handle for a pop-up control.

```
MenuRef GetControlPopupMenuHandle (
   ControlRef control
);
```

Parameters

control
The pop-up control to query.

Return Value
See the Menu Manager documentation for a description of the `MenuRef` data type.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
BSDLLCTest

Declared In
`Controls.h`

GetControlPopupMenuID

Gets the menu ID of a pop-up menu.

```
short GetControlPopupMenuID (
   ControlRef control
);
```

Parameters

control
The pop-up control to query.

Return Value
The menu ID.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
Controls.h

GetControlProperty

Obtains a piece of data that has been previously associated with a control.

```
OSStatus GetControlProperty (
   ControlRef control,
   OSType propertyCreator,
   OSType propertyTag,
   ByteCount bufferSize,
   ByteCount *actualSize,
   void *propertyBuffer
);
```

Parameters

control
A handle to the control whose associated data you wish to obtain.

propertyCreator
Your program's signature, as registered through Apple Developer Technical Support. If your program is of a type that would not normally have a signature (for example, a plug-in), you should still register and use a signature in this case, even though your program's file may not have the same creator code as the signature that you register. The 'macs' property signature is reserved for the system and should not be used.

propertyTag
The application-defined code identifying the data.

bufferSize
A value specifying the size of the data to be obtained. If the size of the data is unknown, use the function GetControlPropertySize (page 603) to get the data's size. If the size specified in the bufferSize parameter does not match the actual size of the property, GetControlProperty only retrieves data up to the size specified or up to the actual size of the property, whichever is smaller, and an error is returned.

actualSize
On input, a pointer to an unsigned 32-bit integer. On return, this value is set to the actual size of the associated data. You may pass NULL for the actualSize parameter if you are not interested in this information.

propertyBuffer
On input, a pointer to a buffer. On return, this buffer contains a copy of the data that is associated with the specified control.

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Discussion
You may use the function GetControlProperty to obtain a copy of data previously set by your application with the function SetControlProperty (page 658).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
CarbonCocoa_PictureCursor
HID Calibrator
HID Explorer

Declared In
`Controls.h`

GetControlPropertyAttributes

Gets the property attributes for a control.

```
OSStatus GetControlPropertyAttributes (
   ControlRef control,
   OSType propertyCreator,
   OSType propertyTag,
   OptionBits *attributes
);
```

Parameters

control
The control to query. For a description of this data type, see ControlRef (page 709).

propertyCreator
The OSType signature, usually the signature of your application, for the property creator of the attributes that are to be obtained.

propertyTag
The OSType signature for the property tag for the attributes that are to be obtained.

attributes
A pointer to a value of type `UInt32` that, on return, contains the attributes of the control specified by `control`.

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

GetControlPropertySize

Obtains the size of a piece of data that has previously been associated with a control.

```
OSStatus GetControlPropertySize (
   ControlRef control,
   OSType propertyCreator,
   OSType propertyTag,
   ByteCount *size
);
```

Parameters

control

A handle to the control whose associated data you wish to examine. For a description of this data type, see ControlRef (page 709).

propertyCreator

Your program's signature, as registered through Apple Developer Technical Support. If your program is of a type that would not normally have a signature (for example, a plug-in), you should still register and use a signature in this case, even though your program's file may not have the same creator code as the signature that you register. The `'macs'` property signature is reserved for the system and should not be used.

propertyTag

The application-defined code identifying the data.

size

On input, a pointer to an unsigned 32-bit integer. On return, this value is set to the actual size of the data.

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Discussion
If you want to retrieve a piece of associated data with the function `GetControlProperty` (page 602), you will typically need to use the `GetControlPropertySize` function beforehand to determine the size of the associated data.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

GetControlReference

Obtains a control's current reference value.

```
SRefCon GetControlReference (
   ControlRef theControl
);
```

Parameters

theControl

A handle to the control whose current reference value you wish to determine.

Return Value
The current reference value for the specified control.

Discussion
When you create a control, you specify an initial reference value, either in the control resource or in the `refCon` parameter of the function `NewControl` (page 634). The reference value is stored in the `contrlRfCon` field of the control structure. You can use this field for any purpose, and you can use the function `SetControlReference` (page 659) to change this value.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

GetControlRegion

Obtains the region corresponding to a given control part.

```
OSStatus GetControlRegion (
   ControlRef inControl,
   ControlPartCode inPart,
   RgnHandle outRegion
);
```

Parameters

inControl
A handle to the control whose part region you want to obtain.

inPart
A constant identifying the control part for which a region is to be obtained. You may specify the `kControlStructureMetaPart` and `kControlContentMetaPart` control part codes, as well as the standard control part codes. See "Control Meta Part Code Constants" (page 790), "Control Part Code Constants " (page 748), and " Control State Part Code Constants" (page 751) for descriptions of possible values.

outRegion
On input, a value of type `RgnHandle`. On return, `GetControlRegion` sets the region to contain the actual dimensions and position of the control part, in local coordinates.

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

GetControlTitle

Obtains the title of a control. (Deprecated in Mac OS X v10.5. Use `HIViewCopyText` (page 2448) or `CopyControlTitleAsCFString` (page 538) instead.)

```
void GetControlTitle (
   ControlRef theControl,
   Str255 title
);
```

Parameters

theControl

A handle to the control whose title you want to determine.

title

On input, a pascal string. On output, the title of the control.

Discussion

The `GetControlTitle` function produces the title of the specified control, which is stored in the `contrlTitle` field of the control structure.

When you create a control, you specify an initial title either in the control resource or in the `title` parameter of the function `NewControl` (page 634). You can change the title by using `SetControlTitle` (page 660).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Controls.h`

GetControlValue

Obtains a control's current setting. (**Deprecated.** Use `GetControl32BitValue` (page 590) instead.)

Not recommended

```
SInt16 GetControlValue (
   ControlRef theControl
);
```

Parameters

theControl

On input, a handle to a control.

Return Value

The current setting of the control.

Discussion

When you create a control, you specify an initial setting either in the control resource or in the `value` parameter of the function `NewControl` (page 634). You can change the setting by calling `SetControlValue` (page 661).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

BSDLLCTest

CarbonSketch
ictbSample

Declared In
`Controls.h`

GetControlVariant

Returns the variation code specified in the control definition function for a particular control. (**Deprecated.** Use custom HIViews instead of custom CDEFs. See *HIView Programming Guide*.)

Not recommended

```
ControlVariant GetControlVariant (
   ControlRef theControl
);
```

Parameters

`theControl`
A handle to the control whose variation code you wish to determine.

Return Value
The variation code for the specified control see the control definition IDs in the "Control Manager Constants" (page 719) section for descriptions of control variation codes. For a description of this data type, see ControlVariant (page 712).

Discussion
A control definition function can use a variation code to describe variations of the same basic control. For example, all pop-up arrows share the same basic control definition function, which is stored in a resource of type `'CDEF'` and has a resource ID of 12. The standard pop-up arrow is large and points to the right; it has a control definition ID of 192. A variation of this is a large, left-pointing arrow, which has a control definition ID of 193. Still another variation, in which the arrow points up, has a control definition ID of 194.

Carbon Porting Notes

Use only if you are using message-based custom controls (CDEFs).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

GetControlViewSize

Obtains the size of the content to which a control's size is proportioned.

```
SInt32 GetControlViewSize (
   ControlRef theControl
);
```

Parameters

theControl

A value of type `ControlHandle`. Pass a handle to the control whose view size you wish to obtain.

Return Value

A value equal to the current size of the content being displayed, expressed in terms of the same units of measurement as are used for the minimum, maximum, and current settings of the control.

Discussion

Your application should call the `GetControlViewSize` function to obtain the current view size of a control. This value is used by the scrollbar control to support proportional scroll boxes.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Controls.h`

GetImageWellContentInfo

Gets information about the content of an image well.

```
OSErr GetImageWellContentInfo (
   ControlRef inButton,
   ControlButtonContentInfoPtr outContent
);
```

Parameters

inButton

The control reference to query.

outContent

On return, the value type `ControlButtonContentInfoPtr` for the control specified by `inButton`.

Return Value

A result code. See "Control Manager Result Codes" (page 824).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`HIImageViews.h`

GetIndexedSubControl

Obtains a handle to a specified embedded control.

```
OSErr GetIndexedSubControl (
   ControlRef inControl,
   UInt16 inIndex,
   ControlRef *outSubControl
);
```

Parameters

inControl

The control from which an embedded control handle is to be obtained. For a description of this data type, see ControlRef (page 709).

inIndex

A one-based index—an integer between 1 and the value returned in the `outNumChildren` parameter of `CountSubControls` (page 538)—specifying the control you wish to access.

outSubControl

On input, a pointer to a `ControlHandle` value. On return, the `ControlHandle` value is set to a handle to the embedded control.

Return Value

A result code. See "Control Manager Result Codes" (page 824). If the index passed in is invalid, the `paramErr` result code is returned.

Discussion

The `GetIndexedSubControl` function is useful for iterating over the control hierarchy. Also, the value of a radio group control is the index of its currently selected embedded radio button control. So, passing the current value of a radio group control into `GetIndexedSubControl` will give you a handle to the currently selected radio button control.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

HID Calibrator

Declared In

`Controls.h`

GetKeyboardFocus

Obtains a handle to the control with the current keyboard focus for a specified window.

```
OSErr GetKeyboardFocus (
   WindowRef inWindow,
   ControlRef *outControl
);
```

Parameters

inWindow

A pointer to the window for which to obtain keyboard focus.

outControl

On input, a pointer to a `ControlHandle` value. On return, the `ControlHandle` value is set to a handle to the control that currently has keyboard focus. Produces `NULL` if no control has focus.

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Discussion
The `GetKeyboardFocus` function returns the handle of the control with current keyboard focus within a specified window.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

GetNewControl

Creates a control from a control resource.

```
ControlRef GetNewControl (
   SInt16 resourceID,
   WindowRef owningWindow
);
```

Parameters

resourceID
The resource ID of the control you wish to create.

owningWindow
A pointer to the window in which to place the control.

Return Value
A handle to the control created from the specified control resource. If `GetNewControl` can't read the control resource from the resource file, it returns `NULL`. For a description of this data type, see ControlRef (page 709).

Discussion
The `GetNewControl` function creates a control structure from the information in the specified control resource, adds the control structure to the control list for the specified window, and returns as its function result a handle to the control. You use this handle when referring to the control in most other Control Manager functions. After making a copy of the control resource, `GetNewControl` releases the memory occupied by the original control resource before returning.

The control resource specifies the rectangle for the control, its initial setting, its visibility state, its maximum and minimum settings, its control definition ID, a reference value, and its title (if any). After you use `GetNewControl` to create the control, you can change the control characteristics with other Control Manager functions.

If the control resource specifies that the control should be visible, the Control Manager draws the control. If the control resource specifies that the control should initially be invisible, you can use the function `ShowControl` (page 669) to make the control visible.

When an embedding hierarchy is established within a window, `GetNewControl` automatically embeds the newly created control in the root control of the owning window.

Availability
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
`Controls.h`

GetRootControl

Obtains a handle to a window's root control.

```
OSErr GetRootControl (
   WindowRef inWindow,
   ControlRef *outControl
);
```

Parameters

inWindow

A pointer to the window to be examined.

outControl

Pass a pointer to a `ControlHandle` value. On return, the `ControlHandle` value is set to a handle to the root control.

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Discussion
You can call `GetRootControl` to determine whether or not a root control (and therefore an embedding hierarchy) exists within a specified window. Once you have the root control's handle, you can pass it to functions such as `DisposeControl` (page 573), `ActivateControl` (page 535),and `DeactivateControl` (page 572) to apply specified actions to the entire embedding hierarchy.

Note that the minimum, maximum, and initial settings for a root control are reserved and should not be changed.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
QTCarbonShell

Declared In
`Controls.h`

GetSuperControl

Obtains a handle to an embedder control.

```
OSErr GetSuperControl (
   ControlRef inControl,
   ControlRef *outParent
);
```

Parameters

inControl

A handle to an embedded control. For a description of this data type, see ControlRef (page 709).

outParent

A pointer to a `ControlHandle` value. On return, the `ControlHandle` value is set to a handle to the embedder control. For a description of this data type, see ControlRef (page 709).

Return Value

A result code. See "Control Manager Result Codes" (page 824).

Discussion

The `GetSuperControl` function gets a handle to the parent control of the control passed in.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

QTCarbonShell

Declared In

`Controls.h`

GetTabContentRect

Gets the content rectangle for a tab.

```
OSErr GetTabContentRect (
   ControlRef inTabControl,
   Rect *outContentRect
);
```

Parameters

inTabControl

The tab control reference to query.

outContentRect

On return, the value of this parameter is a pointer to the content rectangle for the tab specified by `inTabControl`.

Return Value

A result code. See "Control Manager Result Codes" (page 824).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`HITabbedView.h`

HandleControlClick

Responds to cursor movements in a control while the mouse button is down and returns the location of the next mouse-up event.

```
ControlPartCode HandleControlClick (
   ControlRef inControl,
   Point inWhere,
   EventModifiers inModifiers,
   ControlActionUPP inAction
);
```

Parameters

inControl

A handle to the control in which the mouse-down event occurred. Pass the control handle returned by `FindControl` or `FindControlUnderMouse`.

inWhere

A point, specified in local coordinates, where the mouse-down event occurred. Supply the same point you passed to `FindControl` or `FindControlUnderMouse`.

inModifiers

Information from the `modifiers` field of the event structure specifying the state of the modifier keys and the mouse button at the time the event was posted.

inAction

A universal procedure pointer to an action function defining what action your application takes while the user holds down the mouse button. See `ControlActionProcPtr` (page 675) for a description of such an action function. The value of the `inAction` parameter can be a valid `procPtr`, `NULL`, or -1. A value of -1 indicates that the control should either perform auto tracking, or if it is incapable of doing so, do nothing (like `NULL`). For custom controls, what you pass in this parameter depends on how you define the control. If the part index is greater than 128, the pointer must be of type `DragGrayRegionUPP` unless the control supports live feedback, in which case it should be a `ControlActionUPP`.

Return Value

Returns a value of type `ControlPartCode` identifying the control's part see "Control Meta Part Code Constants" (page 790), "Control Part Code Constants " (page 748), and" Control State Part Code Constants" (page 751). For a description of this data type, see ControlPartCode (page 708).

Discussion

Call the `HandleControlClick` function after a call to `FindControl` (page 585) or `FindControlUnderMouse` (page 586). The `HandleControlClick` function should be called instead of `TrackControl` (page 671) to follow the user's cursor movements in a control and provide visual feedback until the user releases the mouse button. Unlike `TrackControl`, `HandleControlClick` allows modifier keys to be passed in so that the control may use these if the control (such as a list box or editable text field) is set up to handle its own tracking.

The visual feedback given by `HandleControlClick` depends on the control part in which the mouse-down event occurs. When highlighting is appropriate, for example, `HandleControlClick` highlights the control part (and removes the highlighting when the user releases the mouse button). When the user holds down the mouse button while the cursor is in an indicator (such as the scroll box of a scroll bar) and moves the mouse, `HandleControlClick` responds by dragging a dotted outline or a ghost image of the indicator. If the user releases the mouse button when the cursor is in an indicator such as the scroll box, `HandleControlClick` calls the control definition function to reposition the indicator.

While the user holds down the mouse button with the cursor in one of the standard controls, `HandleControlClick` performs the following actions, depending on the value you pass in the parameter `inAction`.

- If you pass `NULL` in the `inAction` parameter, `HandleControlClick` uses no action function and therefore performs no additional actions beyond highlighting the control or dragging the indicator. This is appropriate for push buttons, checkboxes, radio buttons, and the scroll box of a scroll bar.
- If you pass a pointer to an action function in the `inAction` parameter, it must define some action that your application repeats as long as the user holds down the mouse button. This is appropriate for the scroll arrows and gray areas of a scroll bar.
- If you pass `(ControlActionUPP)-1L` in the `inAction` parameter, `HandleControlClick` calls the control action function associated with the control. This is appropriate when you are tracking the cursor in a pop-up menu. You can call `GetControlAction` (page 591) to get a pointer to the control action function that is associated with the control, and you can call `SetControlAction` (page 649) to set the control action function that is associated with the control.

For `'CDEF'` resources that implement custom dragging, you usually call `HandleControlClick`, which returns 0 regardless of the user's changes of the control setting. To avoid this, you should use another method to determine whether the user has changed the control setting, for instance, comparing the control's value before and after your call to `HandleControlClick`.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
CarbonSketch

Declared In
`Controls.h`

HandleControlContextualMenuClick

Allows a control to display a contextual menu.

```
OSStatus HandleControlContextualMenuClick (
   ControlRef inControl,
   Point inWhere,
   Boolean *menuDisplayed
);
```

Parameters

inControl

inWhere

The location that was clicked.

menuDisplayed

Pointer to a Boolean whose value is true if the control displayed a contextual menu; otherwise, `false`.

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

HandleControlDragReceive

Tells a control to accept the data from a drag.

```
OSStatus HandleControlDragReceive (
   ControlRef inControl,
   DragReference inDrag
);
```

Parameters

inControl

inDrag

The drag reference that was dropped on the control.

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Discussion
Call this function when the user drops a drag on a control in your window to give the control an opportunity to take any interesting data from the drag. Before calling this function, you must call `SetControlDragTrackingEnabled` (page 653) to enable drag and drop support for the control.Note that this function should not be called in a composited window. Instead, the `SetAutomaticControlDragTrackingEnabledForWindow` API should be used to enable automatic control drag tracking.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

HandleControlDragTracking

Tells a control to respond visually to a drag.

```
OSStatus HandleControlDragTracking (
   ControlRef inControl,
   DragTrackingMessage inMessage,
   DragReference inDrag,
   Boolean *outLikesDrag
);
```

Parameters

inControl

inMessage

A drag message indicating the state of the drag above the control. The meaning of the value you pass in must be relative to the control, not the whole window. For when the drag first enters the control, you should pass `kDragTrackingEnterControl`. While the drag stays within the control, pass `kDragTrackingInControl`. When the drag leaves the control, pass `kDragTrackingLeaveControl`.

inDrag

The drag reference that is over the control.

outLikesDrag

On output, a pointer to a Boolean whose value is `true` if the control can accept the data in the drag reference or `false` if the control cannot accept the data. If the value is `false`,there is no need to call `HandleControlDragReceive` (page 615) when the user drops the dragged object onto the control because the control cannot accept the data.

Return Value

A result code. See "Control Manager Result Codes" (page 824).

Discussion

Call this function when a drag is above a control in your window and you want to give that control a chance to draw appropriately in response to the drag. Before calling this function, you must call `SetControlDragTrackingEnabled` (page 653) to enable drag and drop support for the control.Note that this function should not be called in a composited window. Instead, the `SetAutomaticControlDragTrackingEnabledForWindow` API should be used to enable automatic control drag tracking.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Controls.h`

HandleControlKey

Sends a keyboard event to a control with keyboard focus. (Deprecated in Mac OS X v10.5. For HIView-based controls, send the view a `kEventTextInputUnicodeForKeyEvent` event.)

```
ControlPartCode HandleControlKey (
   ControlRef inControl,
   SInt16 inKeyCode,
   SInt16 inCharCode,
   EventModifiers inModifiers
);
```

Parameters

inControl

A handle to the control that currently has keyboard focus.

inKeyCode

The virtual key code, derived from the event structure. This value represents the key pressed or released by the user. It is always the same for a specific physical key on a particular keyboard regardless of which modifier keys were also pressed.

inCharCode

A character, derived from the event structure. The value that is generated depends on the virtual key code, the state of the modifier keys, and the current `'KCHR'` resource.

inModifiers

Information from the `modifiers` field of the event structure specifying the state of the modifier keys and the mouse button at the time the event was posted.

Return Value

The part code that was hit during the keyboard event; see "Control Meta Part Code Constants" (page 790), "Control Part Code Constants " (page 748), and" Control State Part Code Constants" (page 751). For a description of this data type, see ControlPartCode (page 708).

Discussion

If you have determined that a keyboard event has occurred in a given window, before calling the `HandleControlKey` function, call `GetKeyboardFocus` (page 609) to get the handle to the control that currently has keyboard focus. The `HandleControlKey` function passes the values specified in its `inKeyCode`, `inCharCode`, and `inModifiers` parameters to control definition functions that set the `kControlSupportsFocus` feature bit.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Controls.h`

HandleControlSetCursor

Requests that a control set the cursor based on the mouse location.

```
OSStatus HandleControlSetCursor (
   ControlRef control,
   Point localPoint,
   EventModifiers modifiers,
   Boolean *cursorWasSet
);
```

Parameters

inControl

.

localPoint

The location of the mouse.

modifiers

Information from the `modifiers` field of the event structure specifying the state of the modifier keys and the mouse button at the time the event was posted.

cursorWasSet

Out output, a pointer to a Boolean whose value is `true` if the cursor was set; otherwise, `false`.

Return Value

A result code. See "Control Manager Result Codes" (page 824).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Controls.h`

HideControl

Makes a control, and any latent embedded controls, invisible.

```
void HideControl (
   ControlRef theControl
);
```

Parameters

theControl

A handle to the control to hide.

Discussion

The `HideControl` function makes the specified control invisible. This can be useful, for example, before adjusting a control's size and location. It also adds the control's rectangle to the window's update region, so that anything else that was previously obscured by the control will reappear on the screen. If the control is already invisible, `HideControl` has no effect.

When hiding groups of controls, the state of an embedded control that is hidden or deactivated is preserved so that when the embedder control is shown or activated, the embedded control appears in the same state as the embedder. If the specified control has embedded controls, `HideControl` makes the embedded controls invisible as well.

An embedded control is considered latent when it is deactivated or hidden due to its embedder control being deactivated or hidden. If you call `HideControl` on a latent embedded control, it would not be displayed the next time `ShowControl` (page 669) was called on its embedder control.

To make the control visible again, call `ShowControl` (page 669).

You can also call `SetControlVisibility` (page 663) to hide or show a control without causing it to redraw.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

HiliteControl

Changes the highlighting of a control.

Not recommended

```
void HiliteControl (
   ControlRef theControl,
   ControlPartCode hiliteState
);
```

Parameters

theControl

A handle to the control. For a description of this data type, see ControlRef (page 709).

hiliteState

A value from 0 to 255 that specifies the highlighting state of the control. The value of 0 signifies an active control with no highlighting. A value from 1 through 253 signifies a part code designating the part of the (active) control to highlight. Values 254 and 255 signify that the control is to be made disabled or inactive, respectively, and drawn accordingly. For a description of part code constants, see "Control Part Code Constants " (page 748), "Control Part Code Constants " (page 748), and " Control State Part Code Constants" (page 751).

Discussion
If the Appearance Manager is available, you should call the functions `ActivateControl` (page 535) and `DeactivateControl` (page 572) instead of `HiliteControl` to activate or deactivate a control. This is important if the control is in an embedding hierarchy, since calling these functions will ensure that any latent embedded controls will be activated and deactivated correctly.

If the Appearance Manager is not available, then when you need to make a control inactive (such as when its window is not frontmost) or in any other way change the highlighting of a control, you can use the `HiliteControl` function.

The `HiliteControl` function calls the control definition function to redraw the control with the highlighting specified in the `hiliteState` parameter. The `HiliteControl` function uses the value in this parameter to change the value of the `contrlHilite` field of the control structure.

Except for scroll bars, which you should hide using `HideControl` (page 618), you should use `HiliteControl` to make all controls inactive when their windows are not frontmost. The function `TrackControl` (page 671) automatically uses the `HiliteControl` function as appropriate; when you use `TrackControl`, you don't need to call `HiliteControl`.

Carbon Porting Notes

If you are activating or deactivating a control, you should use `ActivateControl` or `DeactivateControl` instead. Otherwise okay to use.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Controls.h`

IdleControls

Performs idle event processing. (Deprecated in Mac OS X v10.4. You should remove all calls to `IdleControls` because it uses unnecessary processor time. System-supplied controls do not respond to `IdleControls` in Mac OS X.)

Not recommended

```
void IdleControls (
   WindowRef inWindow
);
```

Parameters

inWindow

A pointer to a window containing controls that support idle events.

Discussion

Your application should call the `IdleControls` function to give idle time to any controls that want the `kControlMsgIdle` message. `IdleControls` calls the control with an idle event so the control can do idle-time processing. You should call `IdleControls` at least once in your event loop. See `ControlDefProcPtr` (page 677) for more details on how a control definition function should handle idle processing.

Special Considerations

Idle events are not recommended. If you have a custom control that needs time to perform tasks (such as animation), use Carbon Event timers instead. See *Carbon Event Manager Programming Guide* for more details.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`Controls.h`

InvokeControlActionUPP

Invokes a control action UPP.

```
void InvokeControlActionUPP (
   ControlRef theControl,
   ControlPartCode partCode,
   ControlActionUPP userUPP
);
```

Parameters

theControl

The control for which the control action UPP is to be invoked. For a description of this data type, see ControlRef (page 709).

partCode

The part code for which the control action UPP is to be invoked. For possible values, see "Control Meta Part Code Constants" (page 790), "Control Part Code Constants " (page 748), and " Control State Part Code Constants" (page 751).

userUPP

The UPP that is to be invoked.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Controls.h`

InvokeControlCNTLToCollectionUPP

Invokes a control-to-collection UPP.

```
OSStatus InvokeControlCNTLToCollectionUPP (
   const Rect *bounds,
   SInt16 value,
   Boolean visible,
   SInt16 max,
   SInt16 min,
   SInt16 procID,
   SRefCon refCon,
   ConstStr255Param title,
   Collection collection,
   ControlCNTLToCollectionUPP userUPP
);
```

Parameters

bounds

The bounds of the control.

value

The value of the control.

visible

A Boolean whose value is `true` if the control is visible; otherwise, `false`.

max

The maximum value of the control.

min

The minimum value of the control.

procID
The proc ID.

refCon
The refcon.

title
The title of the control.

collection
The collection.

userUPP
The UPP that is to be invoked.

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

InvokeControlColorUPP

Not recommended

```
OSStatus InvokeControlColorUPP (
   ControlRef inControl,
   SInt16 inMessage,
   SInt16 inDrawDepth,
   Boolean inDrawInColor,
   ControlColorUPP userUPP
);
```

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Carbon Porting Notes

Instead of specifying a callback to redraw your background, you should make the background a control and then embed your other controls within it.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

InvokeControlDefUPP

Invokes a control definition UPP. (Deprecated in Mac OS X v10.5. Use a custom HIView to draw a custom control.)

```
SInt32 InvokeControlDefUPP (
   SInt16 varCode,
   ControlRef theControl,
   ControlDefProcMessage message,
   SInt32 param,
   ControlDefUPP userUPP
);
```

Parameters

varCode

The variation code.

theControl

The control. For a description of this data type, see ControlRef (page 709).

message

The message.

param

The maximum value of the control.

userUPP

The UPP that is to be invoked.

Return Value

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

Controls.h

InvokeControlEditTextValidationUPP

Invokes a control edit text validation UPP.

```
void InvokeControlEditTextValidationUPP (
   ControlRef control,
   ControlEditTextValidationUPP userUPP
);
```

Parameters

theControl

The control. For a description of this data type, see ControlRef (page 709).

userUPP

The UPP that is to be invoked.

Availability

Available in Mac OS X v10.0 and later.

Declared In

HITextViews.h

InvokeControlKeyFilterUPP

Invokes a control key filter UPP.

```
ControlKeyFilterResult InvokeControlKeyFilterUPP (
   ControlRef theControl,
   SInt16 *keyCode,
   SInt16 *charCode,
   EventModifiers *modifiers,
   ControlKeyFilterUPP userUPP
);
```

Parameters

theControl
The control. For a description of this data type, see ControlRef (page 709).

keyCode
The key code.

charCode
The character code.

modifiers
Information from the `modifiers` field of the event structure specifying the state of the modifier keys and the mouse button at the time the event was posted. .

userUPP
The UPP that is to be invoked.

Return Value

For a description of this data type, see ControlKeyFilterResult (page 707).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Controls.h`

InvokeControlUserPaneActivateUPP

Invokes a control user pane activate UPP.

```
void InvokeControlUserPaneActivateUPP (
   ControlRef control,
   Boolean activating,
   ControlUserPaneActivateUPP userUPP
);
```

Parameters

control
The control.

activating
A Boolean whose value is true if the user pane is being activated; otherwise, `false`.

userUPP
The UPP that is to be invoked.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HIContainerViews.h`

InvokeControlUserPaneBackgroundUPP

Invokes a user pane background UPP.

```
void InvokeControlUserPaneBackgroundUPP (
   ControlRef control,
   ControlBackgroundPtr info,
   ControlUserPaneBackgroundUPP userUPP
);
```

Parameters

control
The control.

info
A pointer to information such as the depth and type of the drawing device. For a description of the `ControlBackgroundPtr` data type, see ControlBackgroundRec (page 698).

userUPP
The UPP that is to be activated.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`HIContainerViews.h`

InvokeControlUserPaneDrawUPP

Invokes a user pane draw UPP.

```
void InvokeControlUserPaneDrawUPP (
   ControlRef control,
   ControlPartCode part,
   ControlUserPaneDrawUPP userUPP
);
```

Parameters

control
The control.

userUPP
The part.

userUPP
The UPP that is to be activated.

Availability
Available in Mac OS X v10.0 and later.

Declared In
HIContainerViews.h

InvokeControlUserPaneFocusUPP

Invokes a user pane focus UPP.

```
ControlPartCode InvokeControlUserPaneFocusUPP (
   ControlRef control,
   ControlFocusPart action,
   ControlUserPaneFocusUPP userUPP
);
```

Parameters

control
The control.

action
The action.

userUPP
The UPP that is to be activated.

Return Value
For a description of this data type, see ControlPartCode (page 708).

Availability
Available in Mac OS X v10.0 and later.

Declared In
HIContainerViews.h

InvokeControlUserPaneHitTestUPP

Invokes a user pane hit test UPP.

```
ControlPartCode InvokeControlUserPaneHitTestUPP (
   ControlRef control,
   Point where,
   ControlUserPaneHitTestUPP userUPP
);
```

Parameters

control
The control.

where
The location.

userUPP
The UPP that is to be activated.

Return Value
See ControlPartCode (page 708) for a description of the `ControlPartCode` data type.

Availability
Available in Mac OS X v10.0 and later.

Declared In
HIContainerViews.h

InvokeControlUserPaneIdleUPP

Invokes a user pane idle UPP.

```
void InvokeControlUserPaneIdleUPP (
   ControlRef control,
   ControlUserPaneIdleUPP userUPP
);
```

Parameters

control
The control.

userUPP
The UPP that is to be activated.

Availability
Available in Mac OS X v10.0 and later.

Declared In
HIContainerViews.h

InvokeControlUserPaneKeyDownUPP

Invokes a user pane key down UPP.

```
ControlPartCode InvokeControlUserPaneKeyDownUPP (
   ControlRef control,
   SInt16 keyCode,
   SInt16 charCode,
   SInt16 modifiers,
   ControlUserPaneKeyDownUPP userUPP
);
```

Parameters

control
The control.

keyCode
The key code.

charCode
The character code.

modifiers
The modifiers.

userUPP
The UPP that is to be activated.

Return Value
For a description of this data type, see ControlPartCode (page 708).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HIContainerViews.h`

InvokeControlUserPaneTrackingUPP

Invokes a user pane tracking UPP.

```
ControlPartCode InvokeControlUserPaneTrackingUPP (
   ControlRef control,
   Point startPt,
   ControlActionUPP actionProc,
   ControlUserPaneTrackingUPP userUPP
);
```

Parameters

control
The control.

startPt
The starting point.

actionProc
The action proc.

userUPP
The UPP that is to be activated.

Return Value
For a description of this data type, see ControlPartCode (page 708).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HIContainerViews.h`

InvokeEditUnicodePostUpdateUPP

Invokes a Unicode post update UPP.

```
Boolean InvokeEditUnicodePostUpdateUPP (
   UniCharArrayHandle uniText,
   UniCharCount uniTextLength,
   UniCharArrayOffset iStartOffset,
   UniCharArrayOffset iEndOffset,
   void *refcon,
   EditUnicodePostUpdateUPP userUPP
);
```

Parameters

uniText
The UPP that is to be activated.

uniTextLength
The length of text in `Unitext` parameter.

iStartOffset
The starting offset.

iEndOffset
The ending offset.

refcon
The refcon.

userUPP
The UPP that is to be activated.

Return Value

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HITextViews.h`

IsAutomaticControlDragTrackingEnabledForWindow

Indicates whether automatic drag tracking is enabled for the specified window.

```
OSStatus IsAutomaticControlDragTrackingEnabledForWindow (
   WindowRef inWindow,
   Boolean *outTracks
);
```

Parameters

inWindow

outTracks
On output, a pointer to a Boolean whose value is `true` if the Control Manager's automatic drag tracking is enabled for the window; otherwise, `false`.

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Discussion
For more information on automatic drag tracking, see `SetAutomaticControlDragTrackingEnabledForWindow` (page 643).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

IsControlActive

Returns whether a control is active.

```
Boolean IsControlActive (
   ControlRef inControl
);
```

Parameters

inControl

A handle to the control to be examined.

Return Value

A Boolean whose value is `true` if the control is active; otherwise, `false`.

Discussion

If you wish to determine whether a control is active, you should call `IsControlActive` instead of testing the `contrlHilite` field of the control structure.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Controls.h`

IsControlDragTrackingEnabled

Indicates whether a control's drag track and receive support is enabled.

```
OSStatus IsControlDragTrackingEnabled (
   ControlRef inControl,
   Boolean *outTracks
);
```

Parameters

inControl

.

outTracks

On output, a pointer to a Boolean whose value is `true` if automatic drag tracking and receive support is enabled for the control; otherwise, `false`.

Return Value

A result code. See "Control Manager Result Codes" (page 824).

Discussion

Call this function to determine whether drag and drop support is enabled for a control. Some controls don't support drag and drop; these controls don't track or receive drags even if the `outTracks` parameter indicates a value of `true`.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Controls.h`

IsControlEnabled

Indicates whether a control is enabled.

```
Boolean IsControlEnabled (
   ControlRef inControl
);
```

Parameters

inControl
The control that is to be queried.

Return Value

A Boolean whose value is `true` if the control is enabled; otherwise, `false`.

Availability

Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In

`Controls.h`

IsControlHilited

Indicates whether or not the control is highlighted.

```
Boolean IsControlHilited (
   ControlRef control
);
```

Parameters

control
The control that is to be queried.

Return Value

A Boolean whose value is `true` if the control is highlighted; otherwise, `false`.

Availability

Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In

`Controls.h`

IsControlVisible

Determines whether a control is visible.

```
Boolean IsControlVisible (
   ControlRef inControl
);
```

Parameters

inControl
A handle to the control to be examined.

Return Value
If `true`, the control is visible. If `false`, the control is hidden.

Discussion
If you want to determine whether a control is visible, call `IsControlVisible`. Note that this function indicates the actual user visibility; if the control is marked visible, but its owning window or view is hidden, `isControlVisible` returns `false`. (In compositing mode, if a window is hidden, its root view is also marked as hidden. Similarly, any subviews of a hidden view are considered hidden.) A control's latent visibility (its visibility ignoring the visibility of its parents) can be determined by calling the HIView function `HIViewIsLatentlyVisible`.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

IsValidControlHandle

Reports whether a given handle is a control handle.

```
Boolean IsValidControlHandle (
   ControlRef theControl
);
```

Parameters
`theControl`
A value of type `ControlHandle`. Pass the handle to be examined.

Return Value
`true` if the specified handle is a valid control handle; otherwise, `false`.

Discussion
The `IsValidControlHandle` function confirms whether a given handle is a valid control handle, but it does not check the validity of the data contained in the control itself.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

KillControls

Removes all of the controls from a window that you wish to keep.

```
void KillControls (
   WindowRef theWindow
);
```

Parameters

theWindow

A pointer to the window whose controls you wish to remove.

Discussion

The `KillControls` function disposes of all controls associated with the specified window. To remove just one control, use `DisposeControl` (page 573). If an embedding hierarchy is present, `KillControls` disposes of the controls embedded within a control before disposing of the container control.

You should use `KillControls` when you wish to retain the window but dispose of its controls. The Window Manager functions `CloseWindow` and `DisposeWindow` automatically remove all controls associated with the window and release the memory the controls occupy.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Controls.h`

MoveControl

Moves a control within its window.

```
void MoveControl (
   ControlRef theControl,
   SInt16 h,
   SInt16 v
);
```

Parameters

theControl

A handle to the control you wish to move.

h

The horizontal coordinate (local to the control's window) of the new location of the upper-left corner of the control's rectangle.

v

The vertical coordinate (local to the control's window) of the new location of the upper-left corner of the control's rectangle.

Discussion

The `MoveControl` function moves the control to the new location specified by the `h` and `v` parameters, using them to change the rectangle specified in the `contrlRect` field of the control structure. When the control is visible, `MoveControl` first hides it and then redraws it at its new location.

For example, if the user resizes a document window that contains a scroll bar, your application can use `MoveControl` to move the scroll bar to its new location.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code
QTCarbonShell

Declared In
`Controls.h`

NewControl

Creates a control based on parameter data. (Deprecated in Mac OS X v10.5. Use the specific control creation function instead (for example, `CreateCheckBoxControl` (page 541)).)

```
ControlRef NewControl (
   WindowRef owningWindow,
   const Rect *boundsRect,
   ConstStr255Param controlTitle,
   Boolean initiallyVisible,
   SInt16 initialValue,
   SInt16 minimumValue,
   SInt16 maximumValue,
   SInt16 procID,
   SRefCon controlReference
);
```

Parameters

owningWindow

A pointer to the window in which you want to place the control. All coordinates pertaining to the control are interpreted in this window's local coordinate system.

boundsRect

A pointer to a rectangle, specified in the given window's local coordinates, that encloses the control and thus determines its size and location. When specifying this rectangle, you should follow the guidelines presented in "Dialog Box Layout", in Mac OS 8 Human Interface Guidelines, for control placement and alignment.

controlTitle

The title string, used for push buttons, checkboxes, radio buttons, and pop-up menus. When specifying a multiple-line title, separate the lines with the ASCII character code 0x0D (carriage return). For controls that don't use titles, pass an empty string.

initiallyVisible

A Boolean value specifying the visible/invisible state for the control. If you pass `true` in this parameter, `NewControl` draws the control immediately, without using your window's standard updating mechanism. If you pass `false`, you must later use `ShowControl` (page 669) to display the control.

initialValue

The initial setting for the control. For sliders and scrollbars, pass the appropriate integer value. For checkboxes and radio buttons, pass the constant indicating the current setting (as defined in "Checkbox Value Constants" (page 732) and "Radio Button Value Constants" (page 806). For plain buttons that do not retain a setting, pass 0.

minimumValue

The minimum setting for the control. For sliders and scrollbars, pass the appropriate minimum integer value. For checkboxes and radio buttons, pass 0 (or the equivalent constant from "Checkbox Value Constants" (page 732) or "Radio Button Value Constants" (page 806)). For plain buttons that do not retain a setting, pass 0.

maximumValue

The maximum setting for the control. For sliders and scrollbars, pass the appropriate maximum integer value. For scroll bars, if the maximum value is equal to the minimum value, the control definition function automatically disables the scroll bar. For checkboxes and radio buttons, pass 1 (or the equivalent constant defined in "Checkbox Value Constants" (page 732) or "Radio Button Value Constants" (page 806)). For plain buttons that do not retain a setting, pass 0.

procID

The control definition ID. If the control definition function isn't in memory, it is read in. On Mac OS X, if you do not pass a valid procID (that is, if it does not correspond to a CDEF resource), NewControl will not create a control and will simply return NULL. On Mac OS 9 and earlier, passing an invalid procID will cause NewControl to create a pushbutton control.

controlReference

The control's reference value, which is set and used only by your application.

Return Value

A handle to the control described in its parameters. If `NewControl` runs out of memory or fails, it returns `NULL`. For a description of this data type, see ControlRef (page 709).

Discussion

The `NewControl` function creates a control structure from the information you specify in its parameters, adds the control structure to the control list for the specified window, and returns as its function result a handle to the control. You use this handle when referring to the control in most other Control Manager functions. Generally, you should use the function `GetNewControl` (page 610) instead of `NewControl`, because `GetNewControl` is a resource-based control-creation function that allows you to localize your application without recompiling.

When an embedding hierarchy is established within a window, `NewControl` automatically embeds the newly created control in the root control of the owning window.

Carbon Porting Notes

Carbon does not support custom control definitions stored in `'CDEF'` resources. If you want to specify a custom control definition for `NewControl`, you must compile your definition function directly in your application and then register the function by calling `RegisterControlDefinition` (page 640). When `NewControl` gets a `procID` value that doesn't recognize, it checks a special mapping table to find the pointer that is registered for the resource ID embedded in the `procID` parameter. It then calls that function to implement your control.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Controls.h`

NewControlActionUPP

Creates a UPP for a control action callback function.

```
ControlActionUPP NewControlActionUPP (
   ControlActionProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your control action callback function. See `ControlActionProcPtr` (page 675) for information about defining this function.

Return Value

A UPP to your control action callback function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Controls.h`

NewControlCNTLToCollectionUPP

Creates a UPP for a control-to-collection callback function.

```
ControlCNTLToCollectionUPP NewControlCNTLToCollectionUPP (
   ControlCNTLToCollectionProcPtr userRoutine
);
```

Return Value

A UPP to your control-to-collection callback function.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Controls.h`

NewControlColorUPP

Not recommended

```
ControlColorUPP NewControlColorUPP (
   ControlColorProcPtr userRoutine
);
```

Carbon Porting Notes

Instead of specifying a callback to redraw your background, you should make the background a control and then embed your other controls within it.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
Controls.h

NewControlDefUPP

Creates a UPP for a control definition callback function. (Deprecated in Mac OS X v10.5. Use a custom HIView to draw a custom control.)

```
ControlDefUPP NewControlDefUPP (
   ControlDefProcPtr userRoutine
);
```

Return Value
A UPP to your control definition callback function.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
Controls.h

NewControlEditTextValidationUPP

Creates a UPP for a control edit text validation callback function.

```
ControlEditTextValidationUPP NewControlEditTextValidationUPP (
   ControlEditTextValidationProcPtr userRoutine
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.

Declared In
HITextViews.h

NewControlKeyFilterUPP

```
ControlKeyFilterUPP NewControlKeyFilterUPP (
   ControlKeyFilterProcPtr userRoutine
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.

Declared In
Controls.h

NewControlUserPaneActivateUPP

```
ControlUserPaneActivateUPP NewControlUserPaneActivateUPP (
   ControlUserPaneActivateProcPtr userRoutine
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.

Declared In
HIContainerViews.h

NewControlUserPaneBackgroundUPP

```
ControlUserPaneBackgroundUPP NewControlUserPaneBackgroundUPP (
   ControlUserPaneBackgroundProcPtr userRoutine
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
HIContainerViews.h

NewControlUserPaneDrawUPP

```
ControlUserPaneDrawUPP NewControlUserPaneDrawUPP (
   ControlUserPaneDrawProcPtr userRoutine
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.

Declared In
HIContainerViews.h

NewControlUserPaneFocusUPP

```
ControlUserPaneFocusUPP NewControlUserPaneFocusUPP (
   ControlUserPaneFocusProcPtr userRoutine
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.

Declared In
HIContainerViews.h

NewControlUserPaneHitTestUPP

```
ControlUserPaneHitTestUPP NewControlUserPaneHitTestUPP (
   ControlUserPaneHitTestProcPtr userRoutine
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.

Declared In
HIContainerViews.h

NewControlUserPaneIdleUPP

```
ControlUserPaneIdleUPP NewControlUserPaneIdleUPP (
   ControlUserPaneIdleProcPtr userRoutine
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.

Declared In
HIContainerViews.h

NewControlUserPaneKeyDownUPP

```
ControlUserPaneKeyDownUPP NewControlUserPaneKeyDownUPP (
   ControlUserPaneKeyDownProcPtr userRoutine
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.

Declared In
HIContainerViews.h

NewControlUserPaneTrackingUPP

```
ControlUserPaneTrackingUPP NewControlUserPaneTrackingUPP (
   ControlUserPaneTrackingProcPtr userRoutine
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HIContainerViews.h`

NewEditUnicodePostUpdateUPP

```
EditUnicodePostUpdateUPP NewEditUnicodePostUpdateUPP (
   EditUnicodePostUpdateProcPtr userRoutine
);
```

Return Value

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HITextViews.h`

RegisterControlDefinition

Registers an old-style control definition.

```
OSStatus RegisterControlDefinition (
   SInt16 inCDEFResID,
   const ControlDefSpec *inControlDef,
   ControlCNTLToCollectionUPP inConversionProc
);
```

Parameters

CDEFResID

The virtual resource ID you want to assign to the control definition.

def

A pointer to the control definition function you want to register. Pass `NULL` if you want to unregister a given CDEF proc ID.

conversionProc

A UPP to a callback function to place initialization data (such as the bounds, min and max values, and so on) into a collection.

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Discussion
Mac OS X does not allow you to store custom control definitions in resources. However, some older functions such as `GetNewControl` (page 610) expect a resource ID when creating controls. To work around this, you can use `RegisterControlDefinition` to register "virtual" resource IDs for your control definition functions.

Since custom control definitions receive initialization data as a collection in the `param` parameter, you must provide a callback to properly package this information. See "Control Collection Tag Constants" (page 768) for a list of tags you can apply to the collection. If you do not supply a conversion callback, the Control Manager sends an empty collection to your control definition.

To unregister a control definition, pass `NULL` in the `inDefSpec` parameter for a given CDEF proc ID.

In Mac OS X v10.2 and later, you should consider reimplementing your custom control code using custom HIViews. See I*ntroducing HIView* for more information.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

RemoveControlProperty

Removes a piece of data that has been previously associated with a control.

```
OSStatus RemoveControlProperty (
   ControlRef control,
   OSType propertyCreator,
   OSType propertyTag
);
```

Parameters

control
A handle to the control whose associated data you wish to remove.

propertyCreator
Your program's signature, as registered through Apple Developer Technical Support. If your program is of a type that would not normally have a signature (for example, a plug-in), you should still register and use a signature in this case, even though your program's file may not have the same creator code as the signature that you register. The `'macs'` property signature is reserved for the system and should not be used.

propertyTag
The application-defined code identifying the associated data.

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Discussion
Your application may dissociate data it has previously set with the `SetControlProperty` (page 658) function by calling the `RemoveControlProperty` function.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

ReverseKeyboardFocus

Returns keyboard focus to the prior focusable control in a window.

```
OSErr ReverseKeyboardFocus (
   WindowRef inWindow
);
```

Parameters

inWindow

A pointer to the window for which to reverse keyboard focus.

Return Value

A result code. See "Control Manager Result Codes" (page 824).

Discussion

The `ReverseKeyboardFocus` function reverses the progression of keyboard focus, skipping over deactivated and hidden controls until it finds the previous control to receive keyboard focus in the window.

When `ReverseKeyboardFocus` is called, the Control Manager calls your control definition function and passes `kControlMsgFocus` in its `message` parameter and `kControlFocusPrevPart` in its `param` parameter. In response to this message, your control definition function should change keyboard focus to its previous part, the entire control, or remove keyboard focus from the control, depending upon the circumstances. See `ControlDefProcPtr` (page 677) for a discussion of possible responses to this message.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Controls.h`

SendControlMessage

Sends a message to a control definition function. (**Deprecated.** For custom controls, use a custom HIView instead of a control definition function. See *HIView Programming Guide*.)

Not recommended

```
SInt32 SendControlMessage (
   ControlRef inControl,
   SInt16 inMessage,
   void *inParam
);
```

Parameters

inControl

A handle to the control that is to receive a low-level message. For a description of this data type, see ControlRef (page 709).

inMessage

A bit field representing the message(s) you wish to send; see `ControlDefProcPtr` (page 677).

inParam

The message-dependent data passed in the `param` parameter of the control definition function.

Return Value
Varying data, depending upon the message sent in the `inMessage` parameter.

Discussion
Your application does not normally need to call the `SendControlMessage` function. If you have a special need to call a control definition function directly, call `SendControlMessage` to access and manipulate the control's attributes.

Before calling `SendControlMessage`, you should determine whether the control supports the specific message you wish to send by calling `GetControlFeatures` (page 597) and examining the feature bit field returned. If there are no feature bits returned that correspond to the message you wish to send (for messages 0 through 12), you can assume that all system controls support that message.

Carbon Porting Notes

Don't send messages to standard system control definitions.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

SetAutomaticControlDragTrackingEnabledForWindow

Enables or disables automatic drag tracking for a window.

```
OSStatus SetAutomaticControlDragTrackingEnabledForWindow (
   WindowRef inWindow,
   Boolean inTracks
);
```

Parameters

inWindow

inTracks

A Boolean whose value is `true` to enable the Control Manager's automatic drag tracking support or `false` to disable automatic drag tracking support.

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Discussion
By default, your application is responsible for installing drag tracking and receive handlers on a given window. The Control Manager, however, has support for automatically tracking and receiving drags over controls that you can enable by calling this function with the `inTracks` parameter set to `true`.

The Control Manager automatic drag tracking detects the control the drag is over and calls `HandleControlDragTracking` (page 615) and `HandleControlDragReceive` (page 615) appropriately. By default, the Control Manager's automatic drag tracking is disabled.

Earlier versions of system software enabled automatic drag tracking by default, but as of Mac OS X v10.1.3, Mac OS 9.2, and CarbonLib 1.4, you must call this function to enable automatic drag tracking.

Availability
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code
CarbonCocoa_PictureCursor

Declared In
`Controls.h`

SetBevelButtonContentInfo

Sets the content information for a bevel button.

```
OSErr SetBevelButtonContentInfo (
   ControlRef inButton,
   ControlButtonContentInfoPtr inContent
);
```

Parameters

inButton
The control reference for the bevel button whose content information is to be set.

inContent
A value of type `ControlButtonContentInfoPtr` for the content information that is to be set.

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`HIButtonViews.h`

SetBevelButtonGraphicAlignment

Sets the alignment for a bevel button.

```
OSErr SetBevelButtonGraphicAlignment (
   ControlRef inButton,
   ControlButtonGraphicAlignment inAlign,
   SInt16 inHOffset,
   SInt16 inVOffset
);
```

Parameters

inButton
The control reference for the bevel button that is to be aligned.

inAlign
The alignment that is to be set. For possible values, see "Bevel Button Graphic Alignment Constants" (page 726).

inHOffset
The horizontal offset, in pixels, that is to be applied to the alignment specified by the `inAlign` parameter.

inVOffset
The vertical offset, in pixels, that is to be applied to the alignment specified by the `inAlign` parameter.

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`HIButtonViews.h`

SetBevelButtonMenuValue

Sets the value of a bevel button menu.

```
OSErr SetBevelButtonMenuValue (
   ControlRef inButton,
   MenuItemIndex inValue
);
```

Parameters

inButton
The control reference for the bevel button whose menu value is to be set.

inValue
The value that is to be set.

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`HIButtonViews.h`

SetBevelButtonTextAlignment

Sets the alignment of the text for a bevel button.

```
OSErr SetBevelButtonTextAlignment (
   ControlRef inButton,
   ControlButtonTextAlignment inAlign,
   SInt16 inHOffset
);
```

Parameters

inButton

The control reference for the bevel button whose text is to be aligned.

inAlign

The alignment that is to be set. For possible values, see"Bevel Button Text Alignment Constants " (page 729).

inHOffset

The horizontal offset, in pixels, that is to be applied to the alignment specified by the `inAlign` parameter.

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`HIButtonViews.h`

SetBevelButtonTextPlacement

Sets the placement for bevel button text.

```
OSErr SetBevelButtonTextPlacement (
   ControlRef inButton,
   ControlButtonTextPlacement inWhere
);
```

Parameters

inButton

The control reference for the bevel button whose text is to be placed.

inWhere

The placement that is to be set. For possible values, see "Bevel Button Text Placement Constants " (page 730).

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`HIButtonViews.h`

SetBevelButtonTransform

Sets the transform for a bevel button.

```
OSErr SetBevelButtonTransform (
   ControlRef inButton,
   IconTransformType transform
);
```

Parameters

inButton

The control reference for the bevel button whose text is to be placed.

transform

The transform that is to be set. For possible values, see the `IconTransformType` enumeration described in the *Icon Services and Utilities Reference*.

Return Value

A result code. See "Control Manager Result Codes" (page 824).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`HIButtonViews.h`

SetControl32BitMaximum

Changes the maximum setting of a control and, if appropriate, redraws it accordingly.

```
void SetControl32BitMaximum (
   ControlRef theControl,
   SInt32 newMaximum
);
```

Parameters

theControl

A handle to the control whose maximum setting you wish to change. For a description of this data type, see ControlRef (page 709).

newMaximum

The new maximum setting of the control. In general, to avoid unpredictable behavior, do not set the maximum control value lower than the current minimum value.

Discussion

Your application may use the `SetControl32BitMaximum` function to set a 32-bit value as the maximum setting for a control.

If your application uses a 32-bit control maximum value, it should not attempt to obtain this value by calling the pre–Mac OS 8.5 function `GetControlMaximum` because the 16-bit value that is returned does not accurately reflect the current 32-bit control value. Instead, use the function `GetControl32BitMaximum` (page 589).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code
HID Calibrator
HID Explorer

Declared In
`Controls.h`

SetControl32BitMinimum

Changes the minimum setting of a control and, if appropriate, redraws it accordingly.

```
void SetControl32BitMinimum (
   ControlRef theControl,
   SInt32 newMinimum
);
```

Parameters

theControl
A handle to the control whose minimum setting you wish to change. For a description of this data type, see ControlRef (page 709).

newMinimum
A value specifying the new minimum setting of the control. In general, to avoid unpredictable behavior, do not set the minimum control value higher than the current maximum value.

Discussion
Your application may use the `SetControl32BitMinimum` function to set a 32-bit value as the minimum setting for a control.

If your application uses a 32-bit control minimum value, it should not attempt to obtain this value by calling the pre–Mac OS 8.5 function `GetControlMinimum` because the 16-bit value that is returned does not accurately reflect the current 32-bit control value. Instead, use the function `GetControl32BitMinimum` (page 590).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
HID Calibrator
HID Explorer

Declared In
`Controls.h`

SetControl32BitValue

Changes the current setting of a control and redraws it accordingly.

```
void SetControl32BitValue (
   ControlRef theControl,
   SInt32 newValue
);
```

Parameters

theControl

A handle to the control whose current setting you wish to change. For a description of this data type, see ControlRef (page 709).

newValue

A value specifying the new setting of the control. If the specified value is less than the minimum setting for the control, `SetControl32BitValue` sets the current setting of the control to its minimum setting. If the specified value is greater than the maximum setting, `SetControl32BitValue` sets the control to its maximum.

Discussion

Your application may use the `SetControl32BitValue` function to set a 32-bit value as the current setting for a control.

If your application uses a 32-bit control value, it should not attempt to obtain this value by calling the pre–Mac OS 8.5 function `GetControlValue` because the 16-bit value that is returned does not accurately reflect the current 32-bit control value. Instead, use the function `GetControl32BitValue` (page 590).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

CarbonSketch

HID Calibrator

HID Explorer

QTCarbonShell

Declared In

`Controls.h`

SetControlAction

Sets the action function for a control.

```
void SetControlAction (
   ControlRef theControl,
   ControlActionUPP actionProc
);
```

Parameters

theControl

A handle to the control whose action function is to be changed.

actionProc

A universal procedure pointer to an action function defining what action your application takes while the user holds down the mouse button. See `ControlActionProcPtr` (page 675) for a description of an action function.

Discussion
The `SetControlAction` function associates the action function specified by `actionProc` with the control specified by `theControl`. If the cursor is in the specified control, the `HandleControlClick` (page 613) and `TrackControl` (page 671) functions call the specified action function when the user holds down the mouse button. You must provide the action function, and it must define some action to perform repeatedly as long as the user holds down the mouse button. `HandleControlUnderClick` and `TrackControl` always highlight and drag the control as appropriate.

`SetControlAction` should be used to set the application-defined action function for providing live feedback for standard system scroll bar controls.

Note that the action function associated with a control is used by `TrackControl` only if you set the action function to `TrackControl` to `Pointer(-1)`. Also, an action function can be specified in the `actionProc` parameter to `TrackControl`, so you don't have to call `SetControlAction` to change it.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

SetControlBounds

Sets the bounds of a control.

```
void SetControlBounds (
   ControlRef control,
   const Rect *bounds
);
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
CarbonSketch

Declared In
`Controls.h`

SetControlColorProc

Associates a `ControlColorUPP` with a given Control, thereby allowing you to bypass the embedding hierarchy-based color setup of `SetUpControlBackground`/`SetUpControlTextColor` and replace it with your own.

Not recommended

```
OSStatus SetControlColorProc (
   ControlRef inControl,
   ControlColorUPP inProc
);
```

Parameters

inControl

The ControlRef with whom the color proc should be associated. For a description of this data type, see ControlRef (page 709).

inProc

The color proc to associate with the `ControlRef`. If you pass `NULL`, the `ControlRef` will be dissociated from any previously installed color proc. For a description of the `ControlColorUPP` data type,

Return Value

A result code. See "Control Manager Result Codes" (page 824). An OSStatus code indicating success or failure. The most likely error is a controlHandleInvalidErr resulting from a bad ControlRef.

Discussion

Before an embedded Control can erase, it calls `SetUpControlBackground` (page 667) to have its background color set up by any parent controls. Similarly, any Control that draws text calls `SetUpControlTextColor` (page 668) to have the appropriate text color set up. This allows certain controls (such as tabs and placards) to offer special backgrounds and text colors for any child controls. By default, the set up functions only move up the Control Manager embedding hierarchy looking for a parent which has a special background.

This is fine in a plain vanilla embedding case, but many application frameworks find it troublesome; if there are interesting views between two controls in the embedding hierarchy, the framework needs to be in charge of the background and text colors, otherwise drawing defects will occur.

You can only associate a single color proc with a given `ControlRef`.

Carbon Porting Notes

Instead of specifying a callback to redraw your background, you should make the background a control and then embed your other controls within it.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Controls.h`

SetControlCommandID

Sets the command ID for a control.

```
OSStatus SetControlCommandID (
   ControlRef inControl,
   UInt32 inCommandID
);
```

Parameters

inControl

The control to set.

inCommandID

The command ID that is to be set.

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

SetControlData

Sets control-specific data.

```
OSErr SetControlData (
   ControlRef inControl,
   ControlPartCode inPart,
   ResType inTagName,
   Size inSize,
   const void *inData
);
```

Parameters

inControl

A handle to the control for which data is to be set.

inPart

The part code of the control part for which data is to be set; see "Control Meta Part Code Constants" (page 790), "Control Part Code Constants " (page 748), and " Control State Part Code Constants" (page 751). Passing `kControlEntireControl` indicates that either the control has no parts or the data is not tied to any specific part of the control.

inTagName

A constant representing the control-specific data you wish to set see the data tag constants in the "Control Manager Constants" (page 719) section.

inSize

The size (in bytes) of the data pointed to by the `inData` parameter. For variable-length control data, pass the value returned in the `outMaxSize` parameter of `GetControlDataSize` (page 596) in the `inSize` parameter. The number of bytes must match the actual data size.

inData

A pointer to a buffer allocated by your application. This buffer contains the data that you are sending to the control. After calling `SetControlData`, your application is responsible for disposing of this buffer, if necessary, as information is copied by control.

Return Value
A result code. See "Control Manager Result Codes" (page 824). The result code `errDataNotSupported` indicates that the `inTagName` parameter is not valid.

Discussion
The `SetControlData` function sets control-specific data represented by the value in the `inTagName` parameter to the data pointed to by the `inData` parameter. `SetControlData` could be used, for example, to switch a progress indicator from a determinate to indeterminate state. For a list of the control attributes that can be set, see the data tag constants in the "Control Manager Constants" (page 719) section.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
BSDLLCTest
CarbonSketch
HID Config Save
HID Explorer
QTCarbonShell

Declared In
`Controls.h`

SetControlDataHandle

(**Deprecated.** Use custom HIViews instead of custom CDEFs. See *HIView Programming Guide*.)

Not recommended

```
void SetControlDataHandle (
   ControlRef control,
   Handle dataHandle
);
```

Carbon Porting Notes

Only useful for message-based custom controls (CDEFs).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

SetControlDragTrackingEnabled

Sets the drag tracking state for a control.

```
OSStatus SetControlDragTrackingEnabled (
   ControlRef inControl,
   Boolean inTracks
);
```

Parameters

inControl

The control for which the drag tracking state is to be set.

inTracks

A Boolean whose value is `true` if you want the control to track and receive drags or `false` if want to disable support for drag and drop.

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Discussion
Call this function to enable a control's support for drag and drop. If you don't enable drag and drop support, the control won't track drags.

Some controls don't support drag and drop; these controls won't track or receive drags even if you call this function with the `inTracks` parameter set to `true`.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
CarbonCocoa_PictureCursor

Declared In
`Controls.h`

SetControlFontStyle

Sets the font style for a control.

```
OSErr SetControlFontStyle (
   ControlRef inControl,
   const ControlFontStyleRec *inStyle
);
```

Parameters

inControl

A handle to the control whose font style is to be set. For a description of this data type, see ControlRef (page 709).

inStyle

A pointer to a `ControlFontStyleRec` (page 704) structure. If the `flags` field is cleared, the control uses the system font unless the control variant `kControlUsesOwningWindowsFontVariant` has been specified (control uses window font).

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Discussion
The `SetControlFontStyle` function sets the font style for a given control. To specify the font for controls in a dialog box, it is generally easier to use the dialog font table resource. `SetControlFontStyle` allows you to override a control's default font (system or window font, depending upon whether the control variant `kControlUsesOwningWindowsFontVariant` has been specified). Once you have set a control's font with this function, you can cause the control to revert to its default font by passing a control font style structure with a cleared `flags` field in the `inStyle` parameter.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
BSDLLCTest

Declared In
`Controls.h`

SetControlID

Sets a control's ID.

```
OSStatus SetControlID (
   ControlRef inControl,
   const ControlID *inID
);
```

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
HID Calibrator

Declared In
`Controls.h`

SetControlMaximum

Changes the maximum setting of a control and redraws its indicator or scroll box accordingly. (**Deprecated.** Use `SetControl32BitMaximum` (page 647) instead.)

Not recommended

```
void SetControlMaximum (
   ControlRef theControl,
   SInt16 newMaximum
);
```

Parameters

theControl

A handle to the control whose maximum setting you wish to change.

newMaximum

The new maximum setting.

Discussion

The `SetControlMaximum` function changes the maximum value of a control to the value specified by the `newMaximum` parameter and redraws its indicator or scroll box to reflect its new range.

When you set the maximum setting of a scroll bar equal to its minimum setting, the control definition function makes the scroll bar inactive. When you make the maximum setting exceed the minimum, the control definition function makes the scroll bar active again.

When you create a control, you specify an initial maximum setting either in the control resource or in the `max` parameter of the function `NewControl` (page 634). To determine a control's current maximum setting, use the function `GetControlMaximum` (page 599).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

BSDLLCTest

CarbonSketch

HID Explorer

Declared In

`Controls.h`

SetControlMinimum

Changes the minimum setting of a control and redraws its indicator or scroll box accordingly. (**Deprecated.** Use `SetControl32BitMinimum` (page 648) instead.)

Not recommended

```
void SetControlMinimum (
   ControlRef theControl,
   SInt16 newMinimum
);
```

Parameters

theControl

A handle to the control whose minimum setting you wish to change. For a description of this data type, see ControlRef (page 709).

newMinimum

The new minimum setting.

Discussion
The `SetControlMinimum` function changes the control's minimum value to the value specified by the `newMinimum` parameter and redraws its indicator or scroll box to reflect its new range.

When you create a control, you specify an initial minimum setting either in the control resource or in the `min` parameter of the `NewControl` (page 634) function. To obtain a control's current minimum setting, use the function `GetControlMinimum` (page 600).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
CarbonSketch

Declared In
`Controls.h`

SetControlPopupMenuHandle

Sets the menu handle for a pop-up control.

```
void SetControlPopupMenuHandle (
   ControlRef control,
   MenuRef popupMenu
);
```

Parameters

control
The pop-up control.

popupMenu
The menu handle to set.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

SetControlPopupMenuID

Sets the menu ID for a pop-up control

```
void SetControlPopupMenuID (
   ControlRef control,
   short menuID
);
```

Parameters

control
The pop-up control.

menuID
The menu ID to set.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

SetControlProperty

Associates data with a control.

```
OSStatus SetControlProperty (
   ControlRef control,
   OSType propertyCreator,
   OSType propertyTag,
   ByteCount propertySize,
   const void *propertyData
);
```

Parameters

control
A handle to the control with which you wish to associate data. For a description of this data type, see ControlRef (page 709).

propertyCreator
Your program's signature, as registered through Apple Developer Technical Support. If your program is of a type that would not normally have a signature (for example, a plug-in), you should still register and use a signature in this case, even though your program's file may not have the same creator code as the signature that you register. The `'macs'` property signature is reserved for the system and should not be used.

propertyTag
A value identifying the data. You define the tag your application uses to identify the data.

propertySize
A value specifying the size of the data.

propertyData
On input, a pointer to data of any type. Pass a pointer to a buffer containing the data to be associated; this buffer should be at least as large as the value specified in the `propertySize` parameter.

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Discussion
Your application may use the `SetControlProperty` function to associate any type of data with a control.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
CarbonCocoa_PictureCursor
HID Calibrator

HID Explorer

Declared In
Controls.h

SetControlReference

Changes a control's current reference value.

```
void SetControlReference (
   ControlRef theControl,
   SRefCon data
);
```

Parameters

theControl

A handle to the control whose reference value you want to change. For a description of this data type, see ControlRef (page 709).

data

The new reference value for the control.

Discussion
The SetControlReference function sets the control's reference value to the value you specify in the data parameter.

When you create a control, you specify an initial reference value, either in the control resource or in the refCon parameter of the function NewControl (page 634). Call GetControlReference (page 604) to obtain the current value. You can use this value for any purpose.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
Controls.h

SetControlSupervisor

Routes mouse-down events to the embedder control.

Not recommended

```
OSErr SetControlSupervisor (
   ControlRef inControl,
   ControlRef inBoss
);
```

Parameters

inControl

A handle to an embedded control. For a description of this data type, see ControlRef (page 709).

inBoss

A handle to the embedder control to which mouse-down events are to be routed. For a description of this data type, see ControlRef (page 709).

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Discussion
The `SetControlSupervisor` function allows an embedder control to respond to mouse-down events occurring in its embedded controls.

An example of a standard control that uses this function is the radio group control. Mouse-down events in the embedded controls of a radio group are intercepted by the group control. (The embedded controls in this case must support radio behavior if a mouse-down event occurs in an embedded control within a radio group control that does not support radio behavior, the control tracks normally and the group is not involved.) The group handles all interactions and switches the embedded control's value on and off. If the value of the radio group changes, `TrackControl` (page 671) or `HandleControlClick` (page 613) will return the `kControlRadioGroupPart` part code. If the user tracks off the radio button or clicks the current radio button, `kControlNoPart` is returned.

Carbon Porting Notes

If you are using the Carbon Event Manager, send the event to the next higher control in the containment hierarchy instead.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

SetControlTitle

Changes the title of a control and redraws the control accordingly. (Deprecated in Mac OS X v10.5. Use `HIViewSetText` (page 2488) or `SetControlTitleWithCFString` (page 661) instead.)

Not recommended

```
void SetControlTitle (
   ControlRef theControl,
   ConstStr255Param title
);
```

Parameters

theControl

A handle to a control, the title of which you want to change.

title

The new title for the control.

Discussion
The `SetControlTitle` function changes the `contrlTitle` field of the control structure to the given string and redraws the control, using the system font for the control title.

The Control Manager allows multiple lines of text in the titles of buttons, checkboxes, and radio buttons. When specifying a multiple-line title, separate the lines with the ASCII character code 0x0D (carriage return). If the control is a button, each line is horizontally centered, and the font leading is inserted between lines. (The height of each line is equal to the distance from the ascent line to the descent line plus the leading of the font used. Be sure to make the total height of the rectangle greater than the number of lines times this height.) If the control is a checkbox or a radio button, the text is justified as appropriate for the user's current script system, and the checkbox or button is vertically centered within its rectangle.

When you create a control, you specify an initial title either in the control resource or in the `title` parameter of the function `NewControl` (page 634). To determine a control's current title, use the function `GetControlTitle` (page 605).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Related Sample Code
CarbonSketch

Declared In
`Controls.h`

SetControlTitleWithCFString

Sets the title for a control to the specified Core Foundation string.

```
OSStatus SetControlTitleWithCFString (
   ControlRef inControl,
   CFStringRef inString
);
```

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
CarbonCocoa_PictureCursor
HID Explorer

Declared In
`Controls.h`

SetControlValue

Changes the current setting of a control and redraws it accordingly. (**Deprecated.** Use `SetControl32BitValue` (page 648) instead.)

Not recommended

```
void SetControlValue (
   ControlRef theControl,
   SInt16 newValue
);
```

Parameters

theControl

A handle to the control whose current setting you wish to change. For a description of this data type, see ControlRef (page 709).

newValue

The new setting for the control.

Discussion

For controls whose values the user can set, you can use the `SetControlValue` function to change the value to the specified value and redraw the control to reflect the new setting. For checkboxes and radio buttons, the value 1 fills the control with the appropriate mark, and 0 removes the mark. For Mac OS 8 checkboxes and radio buttons, 2 represents a mixed state; see "Checkbox Value Constants" (page 732) and "Radio Button Value Constants" (page 806). For scroll bars, `SetControlValue` redraws the scroll box where appropriate.

If the specified value is less than the minimum setting for the control, `SetControlValue` sets the control to its minimum setting; if the value is greater than the maximum setting, `SetControlValue` sets the control to its maximum.

When you create a control, you specify an initial setting either in the control resource or in the `value` parameter of the function `NewControl` (page 634). To determine a control's current setting before changing it in response to a user's click in that control, use the function `GetControlValue` (page 606).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

BSDLLCTest

CarbonSketch

HID Config Save

HID Explorer

ictbSample

Declared In

`Controls.h`

SetControlViewSize

Informs the Control Manager of the size of the content to which a control's size is proportioned.

```
void SetControlViewSize (
   ControlRef theControl,
   SInt32 newViewSize
);
```

Parameters

theControl

A handle to the control whose view size is to be set. For a description of this data type, see ControlRef (page 709).

newViewSize

A value 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 control.

Discussion

Your application should call the `SetControlViewSize` function to support proportional scroll boxes. If the user selects the systemwide Appearance preference for proportional scroll boxes and your application doesn't call `SetControlViewSize`, it will still have the traditional square scroll boxes.

To support a proportional scroll box, simply pass the size of the view area—in terms of whatever units the scroll bar uses—to `SetControlViewSize`. The system automatically handles resizing the scroll box, once your application supplies this information.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Controls.h`

SetControlVisibility

Sets the visibility of a control, and any embedded controls, and specifies whether it should be drawn.

```
OSErr SetControlVisibility (
   ControlRef inControl,
   Boolean inIsVisible,
   Boolean inDoDraw
);
```

Parameters

inControl

A handle to the control whose visibility is to be set.

inIsVisible

A Boolean value indicating whether the control is visible or invisible. If you set this value to `true`, the control will be visible. If `false`, the control will be invisible. If you wish to show a control (and latent embedded subcontrols) but do not want to cause screen drawing, pass `true` for this parameter and false in the `inDoDraw` parameter.

inDoDraw

A Boolean value indicating whether the control should be drawn or erased. If `true`, the control's display on the screen should be updated (drawn or erased) based on the value passed in the `inIsVisible` parameter. If `false`, the display will not be updated.

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Discussion
You should call the `SetControlVisibility` function instead of setting the `contrlVis` field of the control structure to set the visibility of a control and specify whether it will be drawn. If the control has embedded controls, `SetControlVisibility` allows you to set their visibility and specify whether or not they will be drawn. If you wish to show a control but do not want it to be drawn onscreen, pass `true` in the `inIsVisible` parameter and false in the `inDoDraw` parameter.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

SetDisclosureTriangleLastValue

Sets the last value of a disclosure triangle.

```
OSErr SetDisclosureTriangleLastValue (
   HIViewRef inDisclosureTriangle,
   SInt16 inValue
);
```

Parameters

`inDisclosureTriangle`
The control reference for the disclosure triangle whose last value is to be set.

`inValue`
The value to set. For possible values, see "Disclosure Triangle Constants" (page 772).

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDisclosureViews.h`

SetImageWellContentInfo

Sets the content information for an image well.

```
OSErr SetImageWellContentInfo (
   ControlRef inButton,
   ControlButtonContentInfoPtr inContent
);
```

Parameters

inButton

The control reference for the image well whose content information is to be set.

inContent

The content to set.

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`HIImageViews.h`

SetImageWellTransform

Sets an image well transform.

```
OSErr SetImageWellTransform (
   ControlRef inButton,
   IconTransformType inTransform
);
```

Parameters

inButton

The control reference for the image well.

inTransform

The transform to set. For possible values, see the `IconTransformType` enumeration described in the *Icon Services and Utilities Reference*.

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Discussion
An transform is a visual appearance modification that is to be made when drawing the control's content.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`HIImageViews.h`

SetKeyboardFocus

Sets the current keyboard focus to a specified control part for a window.

```
OSErr SetKeyboardFocus (
   WindowRef inWindow,
   ControlRef inControl,
   ControlFocusPart inPart
);
```

Parameters

inWindow

A pointer to the window containing the control that is to receive keyboard focus.

inControl

A handle to the control that is to receive keyboard focus.

inPart

A part code specifying the part of a control to receive keyboard focus. To clear a control's keyboard focus, pass `kControlFocusNoPart`. For a description of this data type, see ControlFocusPart (page 703)

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Discussion
A control with keyboard focus receives keyboard events. The Dialog Manager tests to see which control has keyboard focus when a keyboard event is processed and sends the event to that control. If no control has keyboard focus, the keyboard event is discarded. A control retains keyboard focus if it is hidden or deactivated.

Keyboard focus is only available if an embedding hierarchy has been established in the focusable control's window. The default focusing order is based on the order in which controls are added to the window. For more details on embedding hierarchies, see `EmbedControl` (page 583).

The `SetKeyboardFocus` function sets the keyboard focus to a specified control part. The control to receive keyboard focus can be deactivated or invisible. This permits you to set the focus for an item in a dialog box before the dialog box is displayed.

Availability
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code
QTCarbonShell

Declared In
`Controls.h`

SetTabEnabled

Enables and disables a tab control.

```
OSErr SetTabEnabled (
   ControlRef inTabControl,
   SInt16 inTabToHilite,
   Boolean inEnabled
);
```

Parameters

inTabControl

The control reference for the tab.

inTabToHilite

The tab to highlight.

inEnabled

A Boolean whose value is `true` if the tab is to be enabled or `false` to disable the tab.

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`HITabbedView.h`

SetUpControlBackground

Applies the proper background color for the given control to the current port.

```
OSErr SetUpControlBackground (
   ControlRef inControl,
   SInt16 inDepth,
   Boolean inIsColorDevice
);
```

Parameters

inControl

The `ControlRef` that wants to erase. For a description of this data type, see ControlRef (page 709).

inDepth

A short integer indicating the color depth of the device onto which drawing will take place.On Mac OS X, this parameter is ignored; you should always pass `32`.

inIsColorDevice

A Boolean indicating whether the draw device is a color device. On Mac OS X, this parameter is ignored; you should always pass `true`.

Return Value
A result code. See "Control Manager Result Codes" (page 824). An OSStatus code indicating success or failure. The most likely error is a `controlHandleInvalidErr`, resulting from a bad `ControlRef`. Any non-noErr result indicates that the color set up failed, and that the caller should probably give up its attempt to draw.

Discussion
An embedding-savvy control which erases before drawing must ensure that its background color properly matches the body color of any parent controls on top of which it draws. This routine asks the Control Manager to determine and apply the proper background color to the current port.

If a ControlColorProc callback has been provided for the given control, the callback will be called to set up the background color. If no proc exists, or if the proc returns a value other than `noErr`, the Control Manager ascends the parent chain for the given control looking for a control which has a special background (see the `kControlHasSpecialBackground` feature bit). The first such parent is asked to set up the background color (see the `kControlMsgSetUpBackground` message). If no such parent exists, the Control Manager applies any ThemeBrush which has been associated with the owning window (see `SetThemeWindowBackground`).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

SetUpControlTextColor

Applies the proper text color for the given control to the current port.¬¬

```
OSErr SetUpControlTextColor (
   ControlRef inControl,
   SInt16 inDepth,
   Boolean inIsColorDevice
);
```

Parameters

`inControl`
The `ControlRef` that wants to draw text.

`inDepth`
A short integer indicating the color depth of the device onto which drawing will take place. On Mac OS X, this parameter is ignored; you should always pass `32`.

`inIsColorDevice`
A Boolean indicating whether the draw device is a color device. On Mac OS X, this parameter is ignored; you should always pass `true`.

Return Value
A result code. See "Control Manager Result Codes" (page 824). An OSStatus code indicating success or failure. The most likely error is a `controlHandleInvalidErr`, resulting from a bad `ControlRef`. Any non-noErr result indicates that the color set up failed, and that the caller should probably give up its attempt to draw.

Discussion
An embedding-savvy control which draws text must ensure that its text color properly contrasts the background on which it draws. This routine asks the Control Manager to determine and apply the proper text color to the current port.

If a ControlColorProc has been provided for the given control, the proc will be called to set up the text color. If no proc exists, or if the proc returns a value other than noErr, the Control Manager ascends the parent chain for the given control looking for a control which has a special background (see the `kControlHasSpecialBackground` feature bit). The first such parent is asked to set up the text color (see the `kControlMsgApplyTextColor` message). If no such parent exists, the Control Manager chooses a text color which contrasts any ThemeBrush which has been associated with the owning window (see `SetThemeWindowBackground`).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

ShowControl

Makes an invisible control, and any latent embedded controls, visible.

```
void ShowControl (
   ControlRef theControl
);
```

Parameters

theControl
A handle to the control to make visible. For a description of this data type, see ControlRef (page 709).

Discussion
If the specified control is invisible, the `ShowControl` function makes it visible and immediately draws the control within its window without using your window's standard updating mechanism. Note that the `ShowControl` function draws the control in its window, but the control can still be completely or partially obscured by overlapping windows or other objects. If the control is already visible, `ShowControl` has no effect.

When showing groups of controls, the state of an embedded control that is hidden or deactivated is preserved, so that when the embedder control is shown or activated, the embedded control appears in the same state as the embedder. If the specified control has embedded controls, `ShowControl` makes the embedded controls visible as well.

An embedded control is considered latent when it is deactivated or hidden due to its embedder control being deactivated or hidden. If you call `ShowControl` on a latent embedded control whose embedder is disabled, the embedded control will remain invisible until its embedder control is enabled.

You can make a control invisible in several ways:

- Specifying its invisibility in the control resource.
- Passing a value of `false` in the `visible` parameter of `NewControl` (page 634).
- Calling `HideControl` (page 618).
- Calling `SetControlVisibility` (page 663).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

SizeControl

Changes the size of a control's rectangle.

```
void SizeControl (
   ControlRef theControl,
   SInt16 w,
   SInt16 h
);
```

Parameters

theControl

A handle to the control you wish to resize.

w

The new width, in pixels, of the resized control.

h

The new height, in pixels, of the resized control.

Discussion

The `SizeControl` function changes the rectangle specified in the `contrlRect` field of the control structure. The lower-right corner of the rectangle is adjusted so that it has the width and height specified by the `w` and `h` parameters the position of the upper-left corner is not changed. If the control is currently visible, it's first hidden and then redrawn in its new size. The `SizeControl` function will change the window's update region.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

QTCarbonShell

Declared In

`Controls.h`

TestControl

Obtains the control part in which a mouse-down event occurred.

```
ControlPartCode TestControl (
   ControlRef theControl,
   Point testPoint
);
```

Parameters

theControl

A handle to the control in which the mouse-down event occurred.

testPoint

The point, in a window's local coordinates, where the mouse-down event occurred.

Return Value

The part code of the control part, or `0` if the point is outside the control; see "Control Meta Part Code Constants" (page 790), "Control Part Code Constants " (page 748), and" Control State Part Code Constants" (page 751). If the control is invisible or inactive, `TestControl` returns 0. For a description of this data type, see ControlPartCode (page 708).

Discussion
The `TestControl` function is called by the `FindControl` (page 585) and `TrackControl` (page 671) functions; your application does not normally call it.

When the control specified by the parameter `theControl` is visible and active, `TestControl` tests which part of the control contains the point specified by the parameter `testPoint`.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

TrackControl

Responds to cursor movements in a control while the mouse button is down. (**Deprecated.** Use `HandleControlClick` (page 613) instead.)

Not recommended

```
ControlPartCode TrackControl (
   ControlRef theControl,
   Point startPoint,
   ControlActionUPP actionProc
);
```

Parameters

theControl
A handle to the control in which a mouse-down event occurred. For a description of this data type, see ControlRef (page 709).

startPoint
A point, specified in coordinates local to the window, where the mouse-down event occurred.

actionProc
A pointer to an action function defining the action your application takes while the user holds down the mouse button. The value of the `actionProc` parameter can be a valid `procPtr`, `NULL`, or `-1`. A value of `-1` indicates that the control should either perform auto tracking, or if it is incapable of doing so, do nothing (like `NULL`). See `ControlActionProcPtr` (page 675) for information about an action function to specify in this parameter.

Return Value
If the user releases the mouse button while the cursor is inside a control part, `TrackControl` returns a value of type `ControlPartCode`identifying the control part in which the mouse-up event occurs; see "Control Meta Part Code Constants" (page 790), "Control Part Code Constants " (page 748), and" Control State Part Code Constants" (page 751). `TrackControl` returns 0 if the user releases the mouse button while the cursor is outside the control part. If the user releases the mouse button when the cursor is in an indicator such as a scroll box, `TrackControl` calls the control's control definition function to reposition the indicator. For a description of this data type, see ControlPartCode (page 708).

Discussion
When the Appearance Manager is available, you should typically call `HandleControlClick` (page 613) instead of `TrackControl` to follow the user's cursor movements in a control and provide visual feedback until the user releases the mouse button. Unlike the `TrackControl` function, `HandleControlClick` also accepts modifier key information so that the control may take into account the current modifier key state if the control is set up to handle its own tracking.

If the Appearance Manager is not available, you can use the `TrackControl` function to follow the user's cursor movements in a control and provide visual feedback until the user releases the mouse button. The visual feedback given by `TrackControl` depends on the control part in which the mouse-down event occurs. When highlighting is appropriate, for example, `TrackControl` highlights the control part (and removes the highlighting when the user releases the mouse button). When the user holds down the mouse button while the cursor is in an indicator (such as the scroll box of a scroll bar) and moves the mouse, `TrackControl` responds by dragging a dotted outline of the indicator.

While the user holds down the mouse button with the cursor in one of the standard controls, `TrackControl` performs the following actions, depending on the value you pass in the parameter `actionProc`. (For other controls, what you pass in this parameter depends on how you define the control.)

- If you pass `NULL` in the `actionProc` parameter, `TrackControl` uses no action function and therefore performs no additional actions beyond highlighting the control or dragging the indicator. This is appropriate for buttons, checkboxes, radio buttons, and the scroll box of a scroll bar.
- If you pass a pointer to an action function in the `actionProc` parameter, you must provide the function, and it must define some action that your application repeats as long as the user holds down the mouse button. This is appropriate for the scroll arrows and gray areas of a scroll bar.
- If you pass `Pointer(-1)` in the `actionProc` parameter, `TrackControl` looks in the `contrlAction` field of the control structure for a pointer to the control's action function. This is appropriate when you are tracking the cursor in a pop-up menu. (You can use the `GetControlAction` function to determine the value of this field, and you can use the `SetControlAction` function to change this value.) If the `contrlAction` field of the control structure contains a function pointer, `TrackControl` uses the action function it points to; if the field of the control structure also contains the value `Pointer(-1)`, `TrackControl` calls the control's control definition function to perform the necessary action you may wish to do this if you define your own control definition function for a custom control. If the field of the control structure contains the value `NULL`, `TrackControl` performs no action.

Note that when you need to handle events in alert and dialog boxes, Dialog Manager functions automatically call `FindControl` and `TrackControl`.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

UpdateControls

Draws controls in the specified update region of a window.

```
void UpdateControls (
   WindowRef inWindow,
   RgnHandle inUpdateRegion
);
```

Parameters

theWindow

On input, a pointer to the window containing the controls to update.

updateRegion

On input, a handle to the update region of the specified window.

Discussion

The `UpdateControls` function, which should not be called in a compositing window, draws only those controls in the specified window that need updating. This function is faster than the `DrawControls` (page 581) function, which draws all of the controls in a window. By contrast, `UpdateControls` draws only those controls in the update region.

Your application should call `UpdateControls` upon receiving an update event for a window that contains controls. While the Dialog Manager handles update events for controls in alert boxes and dialog boxes, Window Manager functions such as `SelectWindow`, `ShowWindow`, and `BringToFront` do not automatically call `UpdateControls` to display the window's controls.

In response to an update event, you normally call `UpdateControls` after using the Window Manager function `BeginUpdate` and before using the Window Manager function `EndUpdate`. You should set the `updateRegion` parameter to the visible region of the window's port, as specified in the port's `visRgn` field. Note that if your application draws parts of a control outside of its rectangle, `UpdateControls` might not redraw it.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Controls.h`

Callbacks by Task

Defining Your Own Action Function

`ControlActionProcPtr` (page 675)

Defines actions to be performed repeatedly in response to a mouse-down event in a control part.

Defining Your Own Control Definition Function

`ControlDefProcPtr` (page 677)

If you wish to define new, nonstandard controls for your application, you must write a control definition function and either register it with the system using `RegisterControlDefinition` (page 640) or create it directly using `CreateCustomControl` (page 543).

Defining Your Own Key Filter Function

`ControlKeyFilterProcPtr` (page 686)
The key filter function allows for the interception and possible changing of keystrokes destined for a control.

Defining Your Own Text Validation Function

`ControlEditTextValidationProcPtr` (page 685)
Ensures that the content of an editable text control is valid.

Defining Your Own User Pane Functions

`ControlUserPaneActivateProcPtr` (page 687)
Handles activate and deactivate event processing.

`ControlUserPaneBackgroundProcPtr` (page 688)
Sets the background color or pattern for user panes that support embedding.

`ControlUserPaneDrawProcPtr` (page 690)
Draws the content of your user pane control in the rectangle of user pane control.

`ControlUserPaneFocusProcPtr` (page 691)
Handles keyboard focus.

`ControlUserPaneHitTestProcPtr` (page 692)
Returns the part code of the control that the point was in when the mouse-down event occurred.

`ControlUserPaneIdleProcPtr` (page 692)
Performs idle processing.

`ControlUserPaneKeyDownProcPtr` (page 693)
Handles keyboard event processing.

`ControlUserPaneTrackingProcPtr` (page 695)
Tracks a control while the user holds down the mouse button.

Miscellaneous

`ControlCNTLToCollectionProcPtr` (page 676)

`ControlColorProcPtr` (page 677)

`EditUnicodePostUpdateProcPtr` (page 696)

Callbacks

ControlActionProcPtr

Defines actions to be performed repeatedly in response to a mouse-down event in a control part.

```
typedef void (*ControlActionProcPtr) (
    ControlRef theControl,
    ControlPartCode partCode
);
```

If you name your function `MyControlActionProc`, you would declare it like this:

```
void MyControlActionProc (
    ControlRef theControl,
    ControlPartCode partCode
);
```

Parameters

theControl

The control in which the mouse-down event occurred. For a description of this data type, see ControlRef (page 709).

partCode

The control part in which the mouse-down event occurred; see "Control Meta Part Code Constants" (page 790), "Control Part Code Constants " (page 748), and " Control State Part Code Constants" (page 751). When the cursor is still in the control part where the mouse-down event first occurred, this parameter contains that control's part code. When the user drags the cursor outside the original control part, this parameter contains 0.

Discussion

The Control Manager defines the data type `ControlActionUPP` to identify the universal procedure pointer for this application-defined callback function. To provide a pointer to your callback, you can use the function `NewControlActionUPP` (page 636). You can do so with code similar to the following:

```
ControlActionUPP myActionUPP;
myActionUPP = NewControlActionUPP (MyControlActionCallback);
```

When a mouse-down event occurs in a control, `HandleControlClick` (page 613) and `TrackControl` (page 671) respond as is appropriate, typically by highlighting the control or dragging the indicator as long as the user holds down the mouse button. You can define other actions to be performed repeatedly during this interval. To do so, define your own action function and point to it in the `actionProc` parameter of the `TrackControl` function or the `inAction` parameter of `HandleControlClick`. This is the only way to specify actions in response to all mouse-down events in a control or indicator.

When your action function is called for a control part, the action function is passed a handle to the control and the control's part code. The action function should then respond as is appropriate. `MyActionProc` is an example of such an action function. The only exception to this is for indicators that don't support live feedback.

If the mouse-down event occurs in an indicator of a control that does not support live feedback, your action function should take no parameters, because the user may move the cursor outside the indicator while dragging it.

As an alternative to passing a pointer to your action function in a parameter to `TrackControl`, you can use the function `SetControlAction` (page 649) to store a pointer to the action function in the `contrlAction` field in the control structure. When you pass `Pointer(-1)` instead of a function pointer to `TrackControl`, `TrackControl` uses the action function pointed to in the control structure.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Controls.h`

ControlCNTLToCollectionProcPtr

```
typedef OSStatus (*ControlCNTLToCollectionProcPtr) (
    const Rect * bounds,
    SInt16 value,
    Boolean visible,
    SInt16 max,
    SInt16 min,
    SInt16 procID,
    SInt32 refCon,
    ConstStr255Param title,
    Collection collection
);
```

If you name your function `MyControlCNTLToCollectionProc`, you would declare it like this:

```
OSStatus ControlCNTLToCollectionProcPtr (
    const Rect * bounds,
    SInt16 value,
    Boolean visible,
    SInt16 max,
    SInt16 min,
    SInt16 procID,
    SInt32 refCon,
    ConstStr255Param title,
    Collection collection
);
```

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

ControlColorProcPtr

```
typedef OSStatus (*ControlColorProcPtr) (
    ControlRef inControl,
    SInt16 inMessage,
    SInt16 inDrawDepth,
    Boolean inDrawInColor
);
```

If you name your function `MyControlColorProc`, you would declare it like this:

```
OSStatus ControlColorProcPtr (
    ControlRef inControl,
    SInt16 inMessage,
    SInt16 inDrawDepth,
    Boolean inDrawInColor
);
```

Return Value
A result code. See "Control Manager Result Codes" (page 824).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

ControlDefProcPtr

If you wish to define new, nonstandard controls for your application, you must write a control definition function and either register it with the system using `RegisterControlDefinition` (page 640) or create it directly using `CreateCustomControl` (page 543).

```
typedef SInt32 (*ControlDefProcPtr) (
    SInt16 varCode,
    ControlRef theControl,
    ControlDefProcMessage message,
    SInt32 param
);
```

If you name your function `MyControlDefProc`, you would declare it like this:

```
SInt32 MyControlDefProc (
    SInt16 varCode,
    ControlRef theControl,
    ControlDefProcMessage message,
    SInt32 param
);
```

Parameters
varCode
The control's variation code.

theControl
A handle to the control that the operation will affect.

message
A code for the task to be performed. See "Control Definition Message Constants" (page 735) for a description of the constants which you can use here. The subsections that follow explain each of these tasks in detail. For a description of this data type, see ControlDefProcMessage (page 702).

param
Data associated with the task specified by the `message` parameter. If the task requires no data, this parameter is ignored.

Return Value
The function results that your control definition function returns depend on the value that the Control Manager passes in the `message` parameter.

Discussion
Note that Carbon does not allow you to store custom control definitions in a `'CDEF'` resource file as you could in preCarbon systems.

The Control Manager defines the data type `ControlDefUPP` to identify the universal procedure pointer for this application-defined callback function. To provide a pointer to your callback, you can use the function `NewControlDefUPP` (page 637). You can do so with code similar to the following:

```
ControlDefUPP myControlDefUPP;
myControlDefUPP = NewControlDefUPP (MyControlDefCallback);
```

A control definition function determines how a control generally looks and behaves. Various Control Manager functions call a control definition function whenever they need to perform a control-dependent action, such as drawing the control on the screen. In addition to standard control definition functions, defined by the system, you can make your own custom control definition functions.

When various Control Manager functions need to perform a type-dependent action on the control, they call the control definition function and pass it the variation code for its type as a parameter. You can define your own variation codes; this allows you to use one custom definition to handle several variations of the same general control.

To define your own type of control, you write a control definition function, compile it as a resource of type `'CDEF'`, and store it in your resource file. Whenever you create a control, you specify a control definition ID, which the Control Manager uses to determine the control definition function. The control definition ID is an integer that contains the resource ID of the control definition function in its upper 12 bits and a variation code in its lower 4 bits. Thus, for a given resource ID and variation code

control definition ID = (16 x resource ID) + variation code

For example, buttons, checkboxes, and radio buttons all use the standard control definition function with resource ID 0. Because they have variation codes of 0, 1, and 2, respectively, their respective control definition IDs are 0, 1, and 2. See the control definition IDs in the "Control Manager Constants" section for more details.

The Control Manager calls the Resource Manager to access a control definition function with the given resource ID. The Resource Manager reads a control definition function into memory and returns a handle to it. The Control Manager stores this handle in the `contrlDefProc` field of the control structure.

The Control Manager calls your control definition function under various circumstances; the Control Manager uses the `message` parameter to inform your control definition function what action it must perform. The data that the Control Manager passes in the `param` parameter, the action that your control definition function

must undertake, and the function results that your control definition function returns all depend on the value that the Control Manager passes in the `message` parameter. The rest of this section describes how to respond to the various values that the Control Manager passes in the `message` parameter.

Drawing the Control or Its Part

When the Control Manager passes the value `drawCntl` in the `message` parameter, your control definition function should respond by drawing the indicator or the entire control.

The Control Manager passes one of the drawing constants described in `ReverseKeyboardFocus` (page 642) in the low word of the `param` parameter to specify whether the user is drawing an indicator or the whole control. The high-order word of the `param` parameter may contain undefined data; therefore, evaluate only the low-order word of this parameter.

With the exception of part code 128, which is reserved for future use and should not be used, any other value indicates a part code for the control.

If the specified control is visible, your control definition function should draw the control (or the part specified in the `param` parameter) within the control's rectangle. If the control is invisible (that is, if its `contrlVis` field is set to 0), your control definition function does nothing.

When drawing the control or its part, take into account the current values of its `contrlHilite` and `contrlValue` fields in the control structure.

If the part code for your control's indicator is passed in `param`, assume that the indicator hasn't moved the Control Manager, for example, may be calling your control definition function so that you may simply highlight the indicator. However, when your application calls `ClearKeyboardFocus` (page 537), `SetKeyboardFocus` (page 665), and "Control Meta Part Code Constants" (page 790), they in turn may call your control definition function with the `drawCntl` message to redraw the indicator. Since these functions have no way of determining what part code you chose for your indicator, they all pass 129 in `param`, meaning that you should move your indicator. Your control definition function must detect this part code as a special case and remove the indicator from its former location before drawing it. If your control has more than one indicator, you should interpret 129 to mean all indicators.

When sent the message `drawCntl`, your control definition function should return 0 as its function result.

Testing Where the Mouse-Down Event Occurs

When the Control Manager passes the value for the `testCntl` constant in the `message` parameter, your control definition function should respond by determining whether a specified point is in a visible control.

The Control Manager passes a point (in local coordinates) in the `param` parameter. The point's vertical coordinate is contained in the high-order word of the long integer, and horizontal coordinate is contained in the low-order word.

Your control definition function should return the part code of the part that contains the specified point; it should return 0 if the point is outside the control or if the control is inactive.

Calculating the Control and Indicator Regions on 24-Bit Systems

When the Control Manager passes the value for the `calcCRgns` constant in the `message` parameter, your control definition function should calculate the region passed in the `param` parameter for the specified control or its indicator.

The Control Manager passes a QuickDraw region handle in the `param` parameter. If the high-order bit of `param` is set, the region requested is that of the control's indicator otherwise, the region requested is that of the entire control. Your control definition function should clear the high bit of the region handle before calculating the region.

When passed this message, your control definition function should always return 0, and it should express the region in the local coordinate system of the control's window.

Note that the `calcCRgns` message will never be sent to any system running on 32-bit mode and is therefore obsolete in Mac OS 7.6 and later. On Mac OS 7.6 and later, the `calcCntlRgn` and `calcThumbRgn` messages are sent instead.

Calculating the Control and Indicator Regions on 32-Bit Systems

When the Control Manager passes the values for the `calcCntlRgn` or `calcThumbRgn` constants in the `message` parameter, your control definition function should calculate the region for the specified control or its indicator using the QuickDraw region handle passed in the `param` parameter.

If the Control Manager passes the value for the `calcThumbRgn` constant in the `message` parameter, calculate the region occupied by the indicator. If the Control Manager passes the value for the `calcCntlRgn` constant in the `message` parameter, calculate the region for the entire control.

When passed this message, your control definition function should always return 0, and it should express the region in the local coordinate system of the control's window.

Performing Additional Control Initialization

After initializing fields of a control structure as appropriate when creating a new control, the Control Manager passes `initCntl` in the `message` parameter to give your control definition function the opportunity to perform any type-specific initialization you may require. For example, the standard control definition function for scroll bars allocates space for a region to hold the scroll box and stores the region handle in the `contrlData` field of the new control structure.

When passed the value for the `initCntl` constant in the `message` parameter, your control definition function should ignore the `param` parameter and return 0 as a function result.

Performing Additional Control Disposal Actions

The function `DisposeControl` (page 573) passes `dispCntl` in the `message` parameter to give your control definition function the opportunity to carry out any additional actions when disposing of a control. For example, the standard definition function for scroll bars releases the memory occupied by the scroll box region, whose handle is kept in the `contrlData` field of the control structure.

When passed the value for the `dispCntl` constant in the `message` parameter, your control definition function should ignore the `param` parameter and return 0 as a function result.

Dragging the Control or Its Indicator

When a mouse-up event occurs in the indicator of a control, the " Control State Part Code Constants" (page 751) or `ControlKeyDownRec` (page 706) functions call your control definition function and pass `posCntl` in the `message` parameter. In this case, the Control Manager passes a point (in coordinates local to the control's window) in the `param` parameter that specifies the vertical and horizontal offset, in pixels, by which your control definition function should move the indicator from its current position. Typically, this is the offset

between the points where the cursor was when the user pressed and released the mouse button while dragging the indicator. The point's vertical offset is contained in the high-order word of the `param` parameter, and its horizontal offset is contained in the low-order word.

Your definition function should calculate the control's new setting based on the given offset and then, to reflect the new setting, redraw the control and update the `contrlValue` field in the control structure. Your control definition function should ignore the `param` parameter and return 0 as a function result.

Calculating Parameters for Dragging the Indicator

When the Control Manager passes the value for `thumbCntl` in the `message` parameter, your control definition function should respond by calculating values analogous to the `limitRect`, `slopRect`, and `axis` parameters of `DragControl` that constrain how the indicator is dragged. On entry, the fields *`limitRect.top`* and *`limitRect.left`* contain the point where the mouse-down event first occurred.

The Control Manager passes a pointer to a structure of type `IndicatorDragConstraint` in the param parameter. Your definition function should store the appropriate values into the fields of the structure pointed to by the param parameter; they're analogous to the similarly named parameters of the Window Manager function `DragGrayRgn`.

Performing Custom Dragging

When the Control Manager passes the value for the `dragCntl` constant in the message parameter, the `param` parameter typically contains a custom dragging constant with one of the values described in "Drag Control Constants" to specify whether the user is dragging an indicator or the whole control.

When the Appearance Manager is present, the message `kControlMsgHandleTracking` should be sent instead of `dragCntl` to handle any custom tracking; see "Performing Custom Tracking" below.

If you want to use the Control Manager's default method of dragging, which is to call `DragControl` to drag the control or the Window Manager function DragGrayRgn to drag its indicator, return 0 as the function result for your control definition function.

If your control definition function returns a non-zero value, your control definition function (not the Control Manager) must drag the specified control (or its indicator) to follow the cursor until the user releases the mouse button. If the user drags the entire control, your definition function should use the function `MoveControl` to reposition the control to its new location after the user releases the mouse button. If the user drags the indicator, your definition function must calculate the control's new setting (based on the pixel offset between the points where the cursor was when the user pressed and released the mouse button while dragging the indicator) and then, to reflect the new setting, redraw the control and update the `contrlValue` field in the control structure. Note that, in this case, the functions `HandleControlClick` and `TrackControl` return 0 whether or not the user changes the indicator's position. Thus, you must determine whether the user has changed the control's setting by another method, for instance, by comparing the control's value before and after the call to `HandleControlClick`.

Executing an Action Function

The only way to specify actions in response to all mouse-down events in a control or its indicator is to define your own control definition function that specifies an action function. When you create the control, your control definition function must first respond to the `initCntl` message by storing (ControlDefUPP)-1L in the `contrlAction` field of the control structure. (The Control Manager sends the `initCntl` message to your control definition function after initializing the fields of a new control structure.) Then, when your application passes (ControlActionUPP)-1L in the actionProc n passes (ControlActionUPP)-1L in the actionProc parameter of `HandleControlClick` or `TrackControl`, `HandleControlClick` calls your control definition

function with the `autoTrack` message. The Control Manager passes the part code of the part where the mouse-down event occurs in the param parameter. Your control definition function should then use this information to respond as an action function would.

Note that for the `autoTrack` message, the high-order word of the param parameter may contain undefined data; therefore, evaluate only the low-order word of this parameter.

If the mouse-down event occurs in an indicator of a control that supports live feedback, your action function should take two parameters (a handle to the control and the part code of the control where the mouse-down event first occurred). This action function is the same one you would use to define actions to be performed in control part codes in respotrolActionProcPtr"> ControlActionProcPtr.

If the mouse-down event occurs in an indicator of a control that does not support live feedback, your action function should take no parameters, because the user may move the cursor outside the indicator while dragging it.

Specifying Whether Appearance-Compliant Messages Are Supported

If your control definition function supports Appearance-compliant messages, it should return `kControlSupportsNewMessages` as a function result when the Control Manager passes `kControlMsgTestNewMsgSupport` in the message parameter.

Specifying Which Appearance-Compliant Messages Are Supported

If your control definition function supports Appearance-compliant messages, it should return a bit field of the features it supports in response to the `kControlMsgGetFeatures` message. Your control definition function should ignore the param parameter.

Drawing a Ghost Image of the Indicator

If your control definition function supports indicator ghosting, it should return `kControlSupportsGhosting` as one of the feature bits in response to a `kControlMsgGetFeatures` message. If this bit is set and the control indicator is being tracked, the Control Manager calls your control definition function and passes `kControlMsgDrawGhost` in the message parameter. A handle to the region where the ghost should be drawn will be passed in the `param` parameter.

Your control definition function should respond by redrawing the control with the ghosted indicator at the specified location and should return 0 as its function result.

Note that the ghost indicator should always be drawn before the actual indicator so that it appears underneath the actual indicator.

Calculating the Optimal Control Rectangle

If your control definition function supports calculating the optimal dimensions of the control rectangle, it should return `kControlSupportsCalcBestRect` as one of the feature bits in response to the `kControlMsgGetFeatures` message. If this bit is set and `GetBestControlRect` is called, the Control Manager will call your control definition function and pass `kControlMsgCalcBestRect` in the message parameter. The Control Manager passes a pointer to a control size calculation structure, `ControlCalcSizeRec`, in the `param` parameter.

Your control definition function should respond by calculating the width and height of the optimal control rectangle and adjusting the rectangle by setting the `height` and `width` fields of the control size calculation structure to the appropriate values. If your control definition function displays text, it should pass in the offset from the bottom of control to the base of the text in the `baseLine` field of the structure. Your control

definition function should return the offset value stored in the structure's in the `baseLine` field of the structure. Your control definition function should return the offset value stored in the structure's `baseLine` field.

Performing Custom Tracking

If your control definition function supports custom tracking, it should return `kControlHandlesTracking` as one of the feature bits in response to a `kControlMsgGetFeatures` message. If this bit is set and a mouse-down event occurs in your control, `TrackControl` or `HandleControlClick` calls your control definition function and passes `kControlMsgHandlesTracking` in the message parameter. The Control Manager passes a pointer to a control tracking structure, `ControlTrackingRec`, in the `param` parameter. Your control definition function should respond appropriately and return the part code that was hit, or kControlNoPart if the mouse-down event occurred outside the control; see "Control Meta Part Code Constants" (page 790) "Control Part Code Constants " (page 748) and " Control State Part Code Constants" (page 751).

Handling Keyboard Focus

If your control definition function can change its keyboard focus, it should set `kControlSupportsFocus` and `kControlGetsFocusOnClick` as feature bits in response to a `kControlMsgGetFeatures` message. If these bits are set and the `AdvanceKeyboardFocus`, `ReverseKeyboardFocus`, `ClearKeyboardFocus`, or `SetKeyboardFocus` function is called, the Control Manager calls your control definition function and passes `kControlMsgFocus` in the message parameter.

The Control Manager passes one of the control focus part code constants described in "Control Meta Part Code Constants" (page 790).

If the Control Manager passes in a part code, your control definition function should focus on the specified part code. Your function can interpret this in any way it wishes.

Your control definition function should return the control focus part code or actual control part that was focused on. Return `kControlFocusNoPart` if your control does not accept focus or has just relinquished it. Return a non-zero part code to indicate that your control received keyboard focus. Your control definition function is responsible for maintaining which part is focused.

Handling Keyboard Events

If your control definition function can handle keyboard events, it should return `kControlSupportsFocus` —every control that supports keyboard focus must also be able to handle keyboard events—as one of the feature bits in response to a `kControlMsgGetFeatures` message. If this bit is set, the Control Manager will pass `kControlMsgKeyDown` in the message parameter. The Control Manager passes a pointer to a control key down structure, `ControlKeyDownRec`, in the `param` parameter. Your control definition function should respond by processing the keyboard event as appropriate and return 0 as the function result.

Performing Idle Processing

If your control definition function can perform idle processing, it should return `kControlWantsIdle` as one of the feature bits in response to a `kControlMsgGetFeatures` message. If this bit is set and `IdleControls` is called for the window your control is in, the Control Manager will pass `kControlMsgIdle` in the message parameter. Your control definition function should ignore the `param` parameter and respond appropriately. For example, indeterminate progress indicators and asynchronous arrows use idle time to perform their animation.

Your control definition function should return 0 as the function result.

Getting and Setting Control-Specific Data

If your control definition function supports getting and setting control-specific data, it should return `kControlSupportsDataAccess` as one of its features bits in response to the `kControlMsgGetFeatures` message. If this bit is set, the Control Manager will call your control definition function and pass `kControlMsgSetData` in the message parameter when `ControlDataAccessRec`, in the param parameter. Your definition function should respond by filling out the structure and returning an operating system status message as the function result.

Handling Activate and Deactivate Events

If your control definition function wants to be informed whenever it is being activated or deactivated, it should return `kControlWantsActivate` as one of the feature bits in response to the `kControlMsgGetFeatures` message. If this bit is set and your control definition function is being activated or deactivated, the Control Manager calls it and passes `kControlMsgActivate` in the message parameter. The Control Manager passes a 0 or 1 in the param parameter. A value of 0 indicates that the control is being deactivated; 1 indicates that it is being activated.

Your control definition function should respond by performing any special processing before the user pane becomes activated or deactivated, such as deactivating its `TEHandle` or `ListHandle` if it is about to be deactivated.

Your control definition function should return 0 as the function result.

Setting a Control's Background Color or Pattern

If your control definition function supports embedding and draws its own background, it should return `kControlHasSpecialBackground` as one of the feature bits in response to the `kControlMsgGetFeatures` message. If this bit is set and an embedding hierarchy of controls is being drawn in your control, the Control Manager passes `kControlMsgSetUpBackground` in the message parameter of your control definition function. The Control Manager passes a pointer to a filled-in control background structure, `ControlBackgroundRec`, in the `param` parameter. Your control definition function should respond by setting its background color or pattern to whatever is appropriate given the bit depth and device type passed in. Your control definition function should return 0 as the function result.

Supporting Live Feedback

If your control definition function supports live feedback while tracking the indicator, it should return `kControlSupportsLiveFeedback` as one of the feature bits in response to the `kControlMsgGetFeatures` message. If this bit is set, the Control Manager will call your control definition function when it tracks the indicator and pass `kControlMsgCalcValueFromPos` in the message parameter. The Control Manager passes a handle to the indicator region being dragged in the param parameter.

Your control definition function should respond by calculating its value and drawing the control based on the new indicator region passed in. Your control definition function should not recalculate its indicator position. After the user is done dragging the indicator, your control definition function will be called with a `posCntl` message at which time you can recalculate the position of the indicator. Not recalculating the indicator position each time your control definition function is called creates a smooth dragging experience for the user.

Your control definition function should return 0 as the function result.

Being Informed When Subcontrols Are Added or Removed

If your control definition function wishes to be informed when subcontrols are added or removed, it should return `kControlSupportsEmbedding` as one of the feature bits in response to the `kControlMsgGetFeatures` message. If this bit is set, the Control Manager passes `ControlMsgSubControlAdded` in the message parameter immediately after a subcontrol is added, or it passes `kControlMsgSubControlRemoved` just before a subcontrol is removed.

Being Informed When Subcontrols Are Added or Removed

If your control definition function wishes to be informed when subcontrols are added or removed, it should return `kControlSupportsEmbedding` as one of the feature bits in response to the `kControlMsgGetFeatures` message. If this bit is set, the Control Manager passes `ControlMsgSubControlAdded` in the message parameter immediately after a subcontrol is added, or it passes `kControlMsgSubControlRemoved` just before a subcontrol is removed from your embedder control. A handle to the control being added or removed from the embedding hierarchy is passed in the param parameter. Your control definition function should respond appropriately and return 0 as the function result.

Typically, a control definition function only supports this message if it wants to do extra processing in response to changes in its embedded controls. Radio groups use these messages to perform necessary processing for handling embedded controls. For example, if a currently selected radio button is deleted, the group can adjust itself accordingly.

Carbon Porting Notes

Moving forward, you should consider using Carbon Event-based custom controls rather than those based on CDEF messages. See *Handling Carbon Windows and Controls* for more information about creating Carbon event-based controls.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Controls.h`

ControlEditTextValidationProcPtr

Ensures that the content of an editable text control is valid.

```
typedef void (*ControlEditTextValidationProcPtr) (
    ControlRef control
);
```

If you name your function `MyControlEditTextValidationProc`, you would declare it like this:

```
void MyControlEditTextValidationProc (
    ControlRef control
);
```

Parameters

control

A handle to the control containing the editable text to be validated. For a description of this data type, see ControlRef (page 709).

Discussion
Your application typically uses a `MyControlEditTextValidationCallback` function in conjunction with a key filter function to ensure that editable text is valid in cases such as a cut, paste, or clear, where a key filter cannot be called. Use the `kControlEditTextValidationProcTag` control data tag constant, described in "Editable Text Control Data Tag Constants" (page 756), with the functions `SetControlData` and `GetControlData` to set or retrieve a `MyControlEditTextValidationCallback` function.

Note that if you are using the inline input editable text control variant, the Control Manager will not call your `MyControlEditTextValidationCallback` function during inline input. Instead, you may install your own Text Services Manager `TSMTEPostUpdateUPP` callback function to validate text during inline input, or your application can validate the input itself, immediately prior to using the text.

The Control Manager defines the data type `ControlEditTextValidationUPP` to identify the universal procedure pointer for this application-defined callback function. To provide a pointer to your callback, you can use the function `NewControlEditTextValidationUPP` (page 637). You can do so with code similar to the following:

```
ControlEditTextValidationUPP myControlEditTextValidationUPP;
myControlEditTextValidationUPP = NewControlEditTextValidationUPP
(MyControlEditTextValidationCallback);
```

You can then pass `myControlEditTextValidationUPP` in the `inData` parameter of `SetControlData`. When you no longer need the universal procedure pointer, you should remove it using the `DisposeRoutineDescriptor` function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HITextViews.h`

ControlKeyFilterProcPtr

The key filter function allows for the interception and possible changing of keystrokes destined for a control.

```
typedef ControlKeyFilterResult (*ControlKeyFilterProcPtr) (
    ControlRef theControl,
    SInt16 * keyCode,
    SInt16 * charCode,
    EventModifiers * modifiers
);
```

If you name your function `MyControlKeyFilterProc`, you would declare it like this:

```
ControlKeyFilterResult MyControlKeyFilterProc (
    ControlRef theControl,
    SInt16 * keyCode,
    SInt16 * charCode,
    EventModifiers * modifiers
);
```

Parameters

`theControl`
A handle to the control in which the key-down event occurred.

keyCode

The virtual key code derived from the event structure. This value represents the key pressed or released by the user. It is always the same for a specific physical key on a particular keyboard regardless of which modifier keys were also pressed.

charCode

A particular character derived from the event structure. This value depends on the virtual key code, the state of the modifier keys, and the current `'KCHR'` resource. Because this filter provides WorldScript-encoded text in its parameters, it provides no meaningful information for key events generated when a Unicode keyboard layout or input method is active; these layouts and input methods generate Unicode text that often cannot be translated into any WorldScript encoding.

modifiers

The constant in the `modifiers` field of the event structure specifying the state of the modifier keys and the mouse button at the time the event was posted.

Return Value
Returns a value indicating whether or not it allowed or blocked keystrokes; see "Key Filter Result Codes" (page 782). For a description of this data type, see ControlKeyFilterResult (page 707).

Discussion
Controls that support text input (such as editable text and list box controls) can attach a key filter function to filter key strokes and modify them on return.

Important: On Mac OS X, you should avoid using this filter, or at most, use the filter as an indication that the text is changing but do not depend on the charCode parameter to the filter. Use a `kEventTextInputUnicodeForKeyEvent` Carbon event handler as a replacement for the ControlKeyFilter callback; on Mac OS X v10.4 and later, you can also use a `kEventTextShouldChangeInRange` or `kEventTextDidChange` event handler.

The Control Manager defines the data type `ControlKeyFilterUPP` to identify the universal procedure pointer for this application-defined callback function. To provide a pointer to your callback, you can use the function `NewControlKeyFilterUPP` (page 637). You can do so with code similar to the following:

```
ControlKeyFilterUPP myControlKeyFilterUPP;
myControlKeyFilterUPP = NewControlKeyFilterUPP (MyControlKeyFilterCallback);
```

Your key filter function can intercept and change keystrokes destined for a control. Your key filter function can change the keystroke, leave it alone, or block your control definition function from receiving it. For example, an editable text control can use a key filter function to allow only numeric values to be input in its field.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Controls.h`

ControlUserPaneActivateProcPtr

Handles activate and deactivate event processing.

```
typedef void (*ControlUserPaneActivateProcPtr) (
    ControlRef control,
    Boolean activating
);
```

If you name your function `MyControlUserPaneActivateProc`, you would declare it like this:

```
void MyControlUserPaneActivateProc (
    ControlRef control,
    Boolean activating
);
```

Parameters

control

A handle to the control in which the activate event occurred.

activating

If `true`, the control is being activated. If `false`, the control is being deactivated.

Discussion

The Control Manager defines the data type `UserPaneActivateUPP` to identify the universal procedure pointer for this application-defined callback function. To provide a pointer to your callback, you can use the function `NewControlUserPaneActivateUPP` (page 638). You can do so with code similar to the following:

```
ControlUserPaneActivateUPP myControlUserPaneActivateUPP;
myControlUserPaneActivateUPP = NewControlUserPaneActivateUPP
(MyControlUserPaneActivateCallback);
```

Your `MyControlUserPaneActivateCallback` function should perform any special processing before the user pane becomes activated or deactivated. For example, it should deactivate its `TEHandle` or `ListHandle` if the user pane is about to be deactivated.

This function is called only if you've set the `kControlWantsActivate` feature bit on creation of the user pane control.

Once you have provided a user pane application-defined function, you can call the function `SetControlData` (page 652) in order to associate your function with a control. User pane application-defined functions are identified to `SetControlData` by tag constants for a description of the tag constants, see the "Control Manager Constants" section. For example, once you have created the function `MyControlUserPaneActivateCallback`, pass `kControlUserPaneActivateProcTag` in the `tagName` parameter of `SetControlData` (page 652).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`HIContainerViews.h`

ControlUserPaneBackgroundProcPtr

Sets the background color or pattern for user panes that support embedding.

```
typedef void (*ControlUserPaneBackgroundProcPtr) (
    ControlRef control,
    ControlBackgroundPtr info
);
```

If you name your function `MyControlUserPaneBackgroundProc`, you would declare it like this:

```
void MyControlUserPaneBackgroundProc (
    ControlRef control,
    ControlBackgroundPtr info
);
```

Parameters

control

A handle to the control for which the background color or pattern is to be set.

info

A pointer to information such as the depth and type of the drawing device. For a description of the `ControlBackgroundPtr` data type, see ControlBackgroundRec (page 698).

Discussion

The Control Manager defines the data type `ControlUserPaneBackgroundUPP` to identify the universal procedure pointer for this application-defined callback function. To provide a pointer to your callback, you can use the function `NewControlUserPaneBackgroundUPP` (page 638). You can do so with code similar to the following:

```
ControlUserPaneBackgroundUPP myControlUserPaneBackgroundUPP;
myControlUserPaneBackgroundUPP = NewControlUserPaneBackgroundUPP
(MyControlUserPaneBackgroundCallback);
```

Your `MyControlUserPaneBackgroundCallback` function should set the user pane background color or pattern to whatever is appropriate given the bit depth and device type passed in. Your `MyControlUserPaneBackgroundCallback` function is called to set up the background color. This ensures that when an embedded control calls `EraseRgn` or `EraseRect`, the background is erased to the correct color or pattern.

This function is called only if there is a control embedded in the user pane and if you've set the `kControlHasSpecialBackground` and `kControlSupportsEmbedding` feature bits on creation of the user pane control.

Once you have provided a user pane application-defined function, you can call the function `SetControlData` (page 652) in order to associate your function with a control. User pane application-defined functions are identified to `SetControlData` by tag constants for a description of the tag constants, see the "Control Manager Constants" section. For example, once you have created the function `MyControlUserPaneBackgroundCallback`, pass `kControlUserPaneBackgroundProcTag` in the `tagName` parameter of `SetControlData` (page 652).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`HIContainerViews.h`

ControlUserPaneDrawProcPtr

Draws the content of your user pane control in the rectangle of user pane control.

```
typedef void (*ControlUserPaneDrawProcPtr) (
    ControlRef control,
    SInt16 part
);
```

If you name your function `MyControlUserPaneDrawProc`, you would declare it like this:

```
void MyControlUserPaneDrawProc (
    ControlRef control,
    SInt16 part
);
```

Parameters

control

A handle to the user pane control in which you wish drawing to occur. For a description of this data type, see ControlRef (page 709).

part

The part code of the control you should draw. If 0, draw the entire control.

Discussion

The Control Manager defines the data type `ControlUserPaneDrawUPP` to identify the universal procedure pointer for this application-defined callback function. To provide a pointer to your callback, you can use the function `NewControlUserPaneDrawUPP` (page 638). You can do so with code similar to the following:

```
ControlUserPaneDrawUPP myControlUserPaneDrawUPP;
myControlUserPaneDrawUPP = NewControlUserPaneDrawUPP
(MyControlUserPaneDrawCallback);
```

Application-defined user pane functions provide you with the ability to create a custom theme-compliant control without writing your own control definition function. A user pane is a general purpose stub control; it can be used as the root control for a window, as well as providing a way to hook in application-defined functions such as those described below. When the Appearance Manager is available, user panes should be used in dialog boxes instead of user items.

Once you have provided a user pane application-defined function, you can call the function `SetControlData` (page 652) in order to associate your function with a control. User pane application-defined functions are identified to `SetControlData` by tag constants for a description of the tag constants, see the “Control Manager Constants” section.

For example, to set a user pane draw function, pass the constant `kControlUserPaneDrawProcTag` in the `tagName` parameter of `SetControlData` (page 652).The Control Manager then draws the control using a universal procedure pointer to your user pane draw function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`HIContainerViews.h`

ControlUserPaneFocusProcPtr

Handles keyboard focus.

```
typedef ControlPartCode (*ControlUserPaneFocusProcPtr) (
    ControlRef control,
    ControlFocusPart action
);
```

If you name your function `MyControlUserPaneFocusProc`, you would declare it like this:

```
ControlPartCode MyControlUserPaneFocusProc (
    ControlRef control,
    ControlFocusPart action
);
```

Parameters

control

A handle to the control that is to adjust its focus.

action

The part code of the user pane to receive keyboard focus; see `ControlDefProcPtr` (page 677).

Return Value

The part of the user pane actually focused. The constant `kControlFocusNoPart` is returned if the user pane has lost the focus or cannot be focused. For a description of this data type, see ControlPartCode (page 708).

Discussion

The Control Manager defines the data type `ControlUserPaneFocusUPP` to identify the universal procedure pointer for this application-defined callback function. To provide a pointer to your callback, you can use the function `NewControlUserPaneFocusUPP` (page 638). You can do so with code similar to the following:

```
ControlUserPaneFocusUPP myControlUserPaneFocusUPP;
myControlUserPaneFocusUPP = NewControlUserPaneFocusUPP
(MyControlUserPaneFocusCallback);
```

Your `MyControlUserPaneFocusCallback` function is called in response to a change in keyboard focus. It should respond by changing keyboard focus based on the part code passed in the `action` parameter. This function is called only if you've set the `kControlSupportsFocus` feature bit on creation of the user pane control.

Once you have provided a user pane application-defined function, you can call the function `SetControlData` (page 652) in order to associate your function with a control. User pane application-defined functions are identified to `SetControlData` by tag constants for a description of the tag constants, see the "Control Manager Constants" section. For example, once you have created the function `MyControlUserPaneFocusCallback`, pass `kControlUserPaneFocusProcTag` in the `tagName` parameter of `SetControlData` (page 652).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`HIContainerViews.h`

ControlUserPaneHitTestProcPtr

Returns the part code of the control that the point was in when the mouse-down event occurred.

```
typedef ControlPartCode (*ControlUserPaneHitTestProcPtr) (
    ControlRef control,
    Point where
);
```

If you name your function `MyControlUserPaneHitTestProc`, you would declare it like this:

```
ControlPartCode MyControlUserPaneHitTestProc (
    ControlRef control,
    Point where
);
```

Parameters

control

A handle to the control in which the mouse-down event occurred. For a description of this data type, see ControlRef (page 709).

where

The point, in a window's local coordinates, where the mouse-down event occurred.

Return Value

The part code of the control where the mouse-down event occurred. If the point was not over a control, your function should return `kControlNoPart`. For a description of this data type, see ControlPartCode (page 708).

Discussion

The Control Manager defines the data type `ControlUserPaneHitTestUPP` to identify the universal procedure pointer for this application-defined callback function. To provide a pointer to your callback, you can use the function `NewControlUserPaneHitTestUPP` (page 639). You can do so with code similar to the following:

```
ControlUserPaneHitTestUPP myControlUserPaneHitTestUPP;
myControlUserPaneHitTestUPP = NewControlUserPaneHitTestUPP
(MyControlUserPaneHitTestCallback);
```

Once you have provided a user pane application-defined function, you can call the function `SetControlData` (page 652) in order to associate your function with a control. User pane application-defined functions are identified to `SetControlData` by tag constants for a description of the tag constants, see the "Control Manager Constants" section. For example, once you have created the function `MyControlUserPaneHitTestCallback`, pass `kControlUserPaneHitTestProcTag` in the `tagName` parameter of `SetControlData`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`HIContainerViews.h`

ControlUserPaneIdleProcPtr

Performs idle processing.

```
typedef void (*ControlUserPaneIdleProcPtr) (
    ControlRef control
);
```

If you name your function `MyControlUserPaneIdleProc`, you would declare it like this:

```
void MyControlUserPaneIdleProc (
    ControlRef control
);
```

Parameters

control

A handle to the control for which you wish to perform idle processing. For a description of this data type, see ControlRef (page 709).

Discussion

The Control Manager defines the data type `ControlUserPaneIdleUPP` to identify the universal procedure pointer for this application-defined callback function. To provide a pointer to your callback, you can use the function `NewControlUserPaneIdleUPP` (page 639). You can do so with code similar to the following:

```
ControlUserPaneIdleUPP myControlUserPaneIdleUPP;
myControlUserPaneIdleUPP = NewControlUserPaneIdleUPP
(MyControlUserPaneIdleCallback);
```

This function is called only if you've set the `kControlWantsIdle` feature bit on creation of the user pane control.

Once you have provided a user pane application-defined function, you can call the function `SetControlData` (page 652) in order to associate your function with a control. User pane application-defined functions are identified to `SetControlData` by tag constants for a description of the tag constants, see the "Control Manager Constants" section. For example, once you have created the function `MyControlUserPaneIdleCallback`, pass `kControlUserPaneIdleProcTag` in the `tagName` parameter of `SetControlData` (page 652).

Availability

This function is available with Appearance Manager 1.0 and later.

Declared In

`HIContainerViews.h`

ControlUserPaneKeyDownProcPtr

Handles keyboard event processing.

```
typedef ControlPartCode (*ControlUserPaneKeyDownProcPtr) (
    ControlRef control,
    SInt16 keyCode,
    SInt16 charCode,
    SInt16 modifiers
);
```

If you name your function `MyControlUserPaneKeyDownProc`, you would declare it like this:

```
ControlPartCode MyControlUserPaneKeyDownProc (
    ControlRef control,
```

```
    SInt16 keyCode,
    SInt16 charCode,
    SInt16 modifiers
);
```

Parameters

control

A handle to the control in which the keyboard event occurred. For a description of this data type, see ControlRef (page 709).

keyCode

The virtual key code derived from event structure. This value represents the key pressed or released by the user. It is always the same for a specific physical key on a particular keyboard regardless of which modifier keys were also pressed.

charCode

A particular character derived from the event structure. This value depends on the virtual key code, the state of the modifier keys, and the current `'KCHR'` resource.

modifiers

The constant in the `modifiers` field of the event structure specifying the state of the modifier keys and the mouse button at the time the event was posted.

Return Value

The part code of the control where the keyboard event occurred. If the keyboard event did not occur in a control, your function should return `kControlNoPart`. For a description of this data type, see ControlPartCode (page 708).

Discussion

The Control Manager defines the data type `UserPaneKeyDownUPP` to identify the universal procedure pointer for this application-defined callback function. To provide a pointer to your callback, you can use the function `NewControlUserPaneKeyDownUPP` (page 639). You can do so with code similar to the following:

```
ControlUserPaneKeyDownUPP myControlUserPaneKeyDownUPP;
myControlUserPaneKeyDownUPP = NewControlUserPaneKeyDownUPP
(MyControlUserPaneKeyDownCallback);
```

Your `MyControlUserPaneKeyDownCallback` function should handle the key pressed or released by the user and return the part code of the control where the keyboard event occurred. This function is called only if you've set the `kControlSupportsFocus` feature bit on creation of the user pane control.

Once you have provided a user pane application-defined function, you can call the function `SetControlData` (page 652) in order to associate your function with a control. User pane application-defined functions are identified to `SetControlData` by tag constants for a description of the tag constants, see the "Control Manager Constants" section. For example, once you have created the function `MyControlUserPaneKeyDownCallback`, pass `kControlUserPaneKeyDownProcTag` in the `tagName` parameter of `SetControlData` (page 652).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`HIContainerViews.h`

ControlUserPaneTrackingProcPtr

Tracks a control while the user holds down the mouse button.

```
typedef ControlPartCode (*ControlUserPaneTrackingProcPtr) (
    ControlRef control,
    Point startPt,
    ControlActionUPP actionProc
);
```

If you name your function `MyControlUserPaneTrackingProc`, you would declare it like this:

```
ControlPartCode MyControlUserPaneTrackingProc (
    ControlRef control,
    Point startPt,
    ControlActionUPP actionProc
);
```

Parameters

control

A handle to the control in which the mouse-down event occurred. For a description of this data type, see ControlRef (page 709).

startPt

The location of the cursor at the time the mouse button was first pressed, in local coordinates. Your application retrieves this point from the `where` field of the event structure.

actionProc

A pointer to an action function defining what action your application takes while the user holds down the mouse button. The value of the `actionProc` parameter can be a valid `procPtr`, `NULL`, or -1. A value of -1 indicates that the control should either perform auto tracking, or if it is incapable of doing so, do nothing (like `NULL`). For a description of this data type, see ControlActionProcPtr (page 675).

Return Value

The part code of the control part that was tracked. If tracking was unsuccessful, `kControlNoPartCode` is returned. For a description of this data type, see ControlPartCode (page 708).

Discussion

The Control Manager defines the data type `ControlUserPaneTrackingUPP` to identify the universal procedure pointer for this application-defined callback function. To provide a pointer to your callback, you can use the function `NewControlUserPaneTrackingUPP` (page 640). You can do so with code similar to the following:

```
ControlUserPaneTrackingUPP myControlUserPaneTrackingUPP;
myControlUserPaneTrackingUPP = NewControlUserPaneTrackingUPP
(MyControlUserPaneTrackingCallback);
```

Your `MyControlUserPaneTrackingCallback` function should track the control by repeatedly calling the action function specified in the `actionProc` parameter until the mouse button is released. When the mouse button is released, your function should return the part code of the control part that was tracked. This function is called only if you've set the `kControlHandlesTracking` feature bit on creation of the user pane control.

Once you have provided a user pane application-defined function, you can call the function `SetControlData` (page 652) in order to associate your function with a control. User pane application-defined functions are identified to `SetControlData` by tag constants for a description of the tag constants, see the

"Control Manager Constants" section. For example, once you have created the function `MyControlUserPaneTrackingCallback`, pass `kControlUserPaneTrackingProcTag` in the `tagName` parameter of `SetControlData` (page 652).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HIContainerViews.h`

EditUnicodePostUpdateProcPtr

```
typedef Boolean (*EditUnicodePostUpdateProcPtr) (
    UniCharArrayHandle uniText,
    UniCharCount uniTextLength,
    UniCharArrayOffset iStartOffset,
    UniCharArrayOffset iEndOffset,
    void * refcon
);
```

If you name your function `MyEditUnicodePostUpdateProc`, you would declare it like this:

```
Boolean EditUnicodePostUpdateProcPtr (
    UniCharArrayHandle uniText,
    UniCharCount uniTextLength,
    UniCharArrayOffset iStartOffset,
    UniCharArrayOffset iEndOffset,
    void * refcon
);
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HITextViews.h`

Data Types

AuxCtlHandle

```
typedef AuxCtlPtr* AuxCtlHandle;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Controls.h`

AuxCtlPtr

```
typedef AuxCtlRec* AuxCtlPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Controls.h`

AuxCtlRec

```
struct AuxCtlRec {
    Handle acNext;
    ControlRef acOwner;
    CCTabHandle acCTable;
    SInt16 acFlags;
    SInt32 acReserved;
    SInt32 acRefCon;
};
typedef AuxCtlRec AuxCtlRec;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Controls.h`

ClickActivationResult

```
typedef UInt32 ClickActivationResult;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Controls.h`

ControlApplyTextColorRec

```
struct ControlApplyTextColorRec {
    SInt16 depth;
    Boolean colorDevice;
    Boolean active;
};
typedef struct ControlApplyTextColorRec ControlApplyTextColorRec;
typedef ControlApplyTextColorRec * ControlApplyTextColorPtr;
```

Fields
`depth`
The Control Manager sets this field to specify the bit depth (in pixels) of the current graphics port.

`colorDevice`
The Control Manager passes a value of `true` if you are drawing on a color device; otherwise, `false`.

`active`
The Control Manager passes a value of `true` to specify a color suitable for active text; otherwise, `false`.

Discussion
If you implement a custom control definition function, when the Control Manager passes the message `kControlMsgApplyTextColor` in your control definition function's `message` parameter, it also passes a pointer to a structure of type `ControlApplyTextColorRec` in the `param` parameter. The Control Manager sets the `ControlApplyTextColorRec` structure to contain data describing the current drawing environment, and your control definition function is responsible for using that data to apply the proper text color to the current graphics port.

See "Control Definition Message Constants" (page 735) for more details on the `kControlMsgApplyTextColor` message.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

ControlBackgroundRec

```
struct ControlBackgroundRec {
  SInt16 depth;
   Boolean colorDevice;
};
typedef struct ControlBackgroundRec ControlBackgroundRec;
typedef ControlBackgroundRec * ControlBackgroundPtr;
```

Fields
`depth`
A signed 16-bit integer indicating the bit depth (in pixels) of the current graphics port.

`colorDevice`
A Boolean value. If `true`, you are drawing on a color device. If `false`, you are drawing on a monochrome device.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

ControlBevelButtonBehavior

```
typedef UInt16 ControlBevelButtonBehavior;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
HIButtonViews.h

ControlBevelButtonMenuBehavior

```
typedef UInt16 ControlBevelButtonMenuBehavior;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
HIButtonViews.h

ControlButtonContentInfo

```
struct ControlButtonContentInfo {
    ControlContentType contentType
    union {
        SInt16 resID;
        CIconHandle cIconHandle;
        Handle iconSuite;
        IconRef iconRef;
        PicHandle picture;
        Handle ICONHandle;
        CGImageRef imageRef;
    } u;
};
typedef struct ControlButtonContentInfo ControlButtonContentInfo;
typedef ControlButtonContentInfo ControlImageContentInfo;
```

Fields

`contentType`

Specifies the bevel button or image well content type and whether the content is text-only, resource-based, or handle-based; see "Control Content Type Constants" (page 770) for possible values. The value specified in the `contentType` field determines which of the other fields in the structure are used. For a description of this data type, see ControlContentType (page 701).

`u`

If the content type specified in the `contentType` field is `kControlContentIconSuiteRes`, `kControlContentCIconRes`, or `kControlContentPictRes`, this field contains the resource ID of a picture, color icon, or icon suite resource. If the `contentType` field is `kControlContentCGImageRef`, this field contains a `CGImageRef`.

Discussion
You can use the `ControlButtonContentInfo` structure to specify the content for a bevel button or image well. Values of type `ControlButtonContentInfo` are set via `SetControlData` (page 652) and obtained from `GetControlData` (page 594), in conjunction with the `kControlBevelButtonContentTag` and `kControlImageWellContentTag` constants; see "Bevel Button Control Data Tag Constants" (page 723) and "Image Well Control Data Tag Constants" (page 765).

Version Notes
The `ControlButtonContentInfo` type is available with Appearance Manager 1.0 and later.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Controls.h`

ControlCalcSizeRec

```
struct ControlCalcSizeRec {
    SInt16 height;
    SInt16 width;
    SInt16 baseLine;
};
typedef struct ControlCalcSizeRec ControlCalcSizeRec;
typedef ControlCalcSizeRec * ControlCalcSizePtr;
```

Fields

`height`
The optimal height (in pixels) of the control's bounding rectangle.

`width`
The optimal width (in pixels) of the control's bounding rectangle.

`baseLine`
The offset from the bottom of the control to the base of the text. This value is generally negative.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

ControlCapabilities

```
typedef UInt32 ControlCapabilities;
```

ControlClickActivationRec

```
struct ControlClickActivationRec {
    Point localPoint;
    EventModifiers modifiers;
    ClickActivationResult result;
};
typedef struct ControlClickActivationRec ControlClickActivationRec;
typedef ControlClickActivationRec * ControlClickActivationPtr;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

ControlContentType

```
typedef SInt16 ControlContentType;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Controls.h`

ControlContextualMenuClickRec

```
struct ControlContextualMenuClickRec {
    Point localPoint;
    Boolean menuDisplayed;
};
typedef struct ControlContextualMenuClickRec ControlContextualMenuClickRec;
typedef ControlContextualMenuClickRec * ControlContextualMenuClickPtr;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

ControlDataAccessRec

```
struct ControlDataAccessRec {
    ResType tag;
    ResType part;
    Size size;
    Ptr dataPtr;
};
typedef struct ControlDataAccessRec ControlDataAccessRec;
typedef ControlDataAccessRec * ControlDataAccessPtr;
```

Fields

`tag`

A constant representing a piece of data that is passed in (in response to a `kControlMsgSetData` message) or returned (in response to a `kControlMsgGetData` message); see "Scrolling Text Box Control Data Tag Constants" (page 809) for a description of these constants. The control definition function should return `errDataNotSupported` if the value in the `tag` parameter is unknown or invalid.

`part`

The part of the control that this data should be applied to. If the information is not tied to a specific part of the control or the control has no parts, pass 0.

`size`

On entry, the size of the buffer pointed to by the `dataPtr`field. In response to a `kControlMsgGetData` message, this field should be adjusted to reflect the actual size of the data that the control is maintaining. If the size of the buffer being passed in is smaller than the actual size of the data, the control definition function should return `errDataSizeMismatch`.

`dataPtr`

A pointer to a buffer to read or write the information requested. In response to a `kControlMsgGetData` message, this field could be `NULL`, indicating that you wish to return the size of the data in the `size` field.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

ControlDefProcMessage

```
typedef SInt16 ControlDefProcMessage;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

ControlDefSpec

```
struct ControlDefSpec {
    ControlDefType defType
    union {
        ControlDefUPP defProc;
        void * classRef;
    } u;
};
typedef struct ControlDefSpec ControlDefSpec;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

ControlDefType

```
typedef UInt32 ControlDefType;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

ControlEditTextSelectionRec

```
struct ControlEditTextSelectionRec {
    SInt16 selStart;
    SInt16 selEnd;
};
typedef struct ControlEditTextSelectionRec ControlEditTextSelectionRec;
typedef ControlEditTextSelectionRec * ControlEditTextSelectionPtr;
```

Fields

`selStart`

The start of the editable text selection.

`selEnd`

The end of the editable text selection.

Discussion

You can use the `ControlEditTextSelectionRec` type to specify a selection range in an editable text control. You pass a pointer to the editable text selection structure to `GetControlData` (page 594) and `SetControlData` (page 652) to access and set the current selection range in an editable text control.

Version Notes

The `ControlEditTextSelectionRec` type is available with Appearance Manager 1.0 and later.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`HITextViews.h`

ControlFocusPart

```
typedef SInt16 ControlFocusPart;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Controls.h`

ControlFontStyleRec

```
struct ControlFontStyleRec {
    SInt16 flags;
    SInt16 font;
    SInt16 size;
    SInt16 style;
    SInt16 mode;
    SInt16 just;
    RGBColor foreColor;
    RGBColor backColor;
};
typedef struct ControlFontStyleRec ControlFontStyleRec;
typedef ControlFontStyleRec * ControlFontStylePtr;
```

Fields

`flags`

A value specifying which fields of the structure should be applied to the control; see "Mac OS 8.5 Control Font Style Flag Constant" (page 787) and "Control Font Style Flag Constants" (page 745). If none of the flags in the `flags` field of the structure are set, the control uses the system font unless the control variant `kControlUsesOwningWindowsFontVariant` has been specified, in which case the control uses the window font.

`font`

If the `kControlUseFontMask` bit is set, then this field contains a value specifying the ID of the font family to use. If this bit is not set, then the system default font is used. A meta font constant can be specified instead; see "Meta Font Constants" (page 791).

`size`

If the `kControlUseSizeMask` bit is set, then this field contains a value specifying the point size of the text. If the `kControlAddSizeMask` bit is set, this value will represent the size to add to the current point size of the text. A meta font constant can be specified instead; see "Meta Font Constants" (page 791).

`style`

If the `kControlUseFaceMask` bit is set, then this field contains a value specifying which styles to apply to the text. If all bits are clear, the plain font style is used. The bit numbers and the styles they represent are bold (0), italic (1), underline (2), outline (3), shadow (4), condensed (5), and extended (6).

`mode`

If the `kControlUseModeMask` bit is set, then this field contains a value specifying how characters are drawn in the bit image. See *Inside Macintosh: Imaging With QuickDraw* for a discussion of transfer modes.

`just`

If the `kControlUseJustMask` bit is set, then this field contains a value specifying text justification. Possible values are `teFlushDefault` (0), `teCenter` (1), `teFlushRight` (-1), and `teFlushLeft` (-2).

`foreColor`

If the `kControlUseForeColorMask` bit is set, then this field contains an RGB color value to use when drawing the text.

`backColor`

If the `kControlUseBackColorMask` bit is set, then this field contains an RGB color value to use when drawing the background behind the text. In certain text modes, background color is ignored.

Discussion
You can use the `ControlFontStyleRec` type to specify a control's font. You pass a pointer to the control font style structure in the `inStyle` parameter of `SetControlFontStyle` (page 654) to specify a control's font. If none of the flags in the `flags` field of the structure are set, the control uses the system font unless the control variant `kControlUsesOwningWindowsFontVariant` has been specified, in which case the control uses the window font. The `ControlFontStyleRec` type is available with Appearance Manager 1.0 and later.

Note that if you wish to specify the font for controls in a dialog box, you should use a dialog font table resource, which is automatically read in by the Dialog Manager.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Controls.h`

ControlGetRegionRec

```
struct ControlGetRegionRec {
    RgnHandle region;
    ControlPartCode part;
};
typedef struct ControlGetRegionRec ControlGetRegionRec;
typedef ControlGetRegionRec * ControlGetRegionPtr;
```

Fields
`region`
A value allocated by the Control Manager. Your control definition function should set this field to the region that contains the control part specified in the `part` field.

`part`
The Control Manager passes a constant identifying the control part for which a region is to be obtained. For descriptions of possible values, see "Control Part Code Constants " (page 748), "Control Part Code Constants " (page 748), and " Control State Part Code Constants" (page 751). For a description of this data type, see ControlPartCode (page 708).

Discussion
If you implement a custom control definition function, when the Control Manager passes the message `kControlMsgGetRegion` in your control definition function's `message` parameter, it also passes a pointer to a structure of type `ControlGetRegionRec` in the `param` parameter. Your control definition function is responsible for setting the `region` field of the `ControlGetRegionRec` structure to the region that contains the control part which the Control Manager specifies in the `part` field.

See "Control Definition Message Constants" (page 735) for more details on the `kControlMsgGetRegion` message.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

ControlHandle

```
typedef ControlRef ControlHandle;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HIObject.h`

ControlID

```
struct ControlID {
    OSType signature;
    SInt32 id;
};
typedef struct ControlID ControlID;
typedef ControlID HIViewID;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Controls.h`

ControlImageContentInfo

```
typedef ControlButtonContentInfo ControlImageContentInfo;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Controls.h`

ControlKeyDownRec

```
struct ControlKeyDownRec {
    EventModifiers modifiers;
    SInt16 keyCode;
    SInt16 charCode;
};
typedef struct ControlKeyDownRec ControlKeyDownRec;
typedef ControlKeyDownRec * ControlKeyDownPtr;
```

Fields
`modifiers`
The constant in the `modifiers` field of the event structure specifying the state of the modifier keys and the mouse button at the time the event was posted.

`keyCode`
: The virtual key code derived from the event structure. This value represents the key pressed or released by the user. It is always the same for a specific physical key on a particular keyboard regardless of which modifier keys were also pressed.

`charCode`
: A particular character derived from the event structure. This value depends on the virtual key code, the state of the modifier keys, and the current `'KCHR'` resource.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Controls.h`

ControlKeyFilterResult

```
typedef SInt16 ControlKeyFilterResult;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Controls.h`

ControlKeyScriptBehavior

```
typedef UInt32 ControlKeyScriptBehavior;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Controls.h`

ControlKind

```
struct ControlKind {
    OSType signature;
    OSType kind;
};
typedef struct ControlKind ControlKind;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Controls.h`

ControlNotification

```
typedef UInt32 ControlNotification;
```

ControlNotificationUPP

```
typedef ControlNotificationProcPtr ControlNotificationUPP;
```

ControlPartCode

```
typedef SInt16 ControlPartCode;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Controls.h`

ControlPopupArrowOrientation

```
typedef UInt16 ControlPopupArrowOrientation;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HIPopupButton.h`

ControlPopupArrowSize

```
typedef UInt16 ControlPopupArrowSize;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HIPopupButton.h`

ControlPtr

```
typedef ControlRecord* ControlPtr;
```

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In
`Controls.h`

ControlRecord

```
struct ControlRecord {
    ControlRef nextControl;
    WindowRef contrlOwner;
    Rect contrlRect;
    UInt8 contrlVis;
    UInt8 contrlHilite;
    SInt16 contrlValue;
    SInt16 contrlMin;
    SInt16 contrlMax;
    Handle contrlDefProc;
    Handle contrlData;
    ControlActionUPP contrlAction;
    SInt32 contrlRfCon;
    Str255 contrlTitle;
};
typedef ControlRecord ControlRecord;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Controls.h`

ControlRef

Defines an opaque type that represents a control.

```
typedef struct OpaqueControlRef * ControlRef;
```

Discussion
A control is a user interface object that gives feedback or otherwise facilitates user interaction. The `ControlRef` type is an opaque type used to describe a control's properties. You can obtain and change the values in a control by calling specific control accessor functions.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HIObject.h`

ControlSetCursorRec

```
struct ControlSetCursorRec {
    Point localPoint;
    EventModifiers modifiers;
    Boolean cursorWasSet;
};
typedef struct ControlSetCursorRec ControlSetCursorRec;
typedef ControlSetCursorRec * ControlSetCursorPtr;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
Controls.h

ControlSize

```
typedef UInt16 ControlSize;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
Controls.h

ControlTabEntry

```
struct ControlTabEntry {
    ControlButtonContentInfo * icon;
    CFStringRef name;
    Boolean enabled;
};
typedef struct ControlTabEntry ControlTabEntry;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
HITabbedView.h

ControlTabInfoRec

```
struct ControlTabInfoRec {
    SInt16 version;
    SInt16 iconSuiteID;
    Str255 name;
};
typedef struct ControlTabInfoRec ControlTabInfoRec;
```

Fields

`version`
The version of the tab information structure. The only currently available version value is 0.

`iconSuiteID`
The ID of an icon suite to use for the tab label. If the specified ID is not found, no icon is displayed for the tab label. Pass 0 for no icon.

`name`
The title to be used for the tab label.

Discussion
You can use the `ControlTabInfoRec` type to specify the icon and title for a tab control. If you are not creating a tab control with a `'tab#'` resource, you can call `SetControlMaximum` to set the number of tabs in a tab control. Then use the functions `SetControlData` (page 652) and `GetControlData` (page 594) with the `ControlTabInfoRec` structure to access information for an individual tab in a tab control.

Version Notes
The `ControlTabInfoRec` type is available with Appearance Manager 1.0.1 and later.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HITabbedView.h`

ControlTabInfoRecV1

```
struct ControlTabInfoRecV1 {
    SInt16 version;
    SInt16 iconSuiteID;
    CFStringRef name;
};
typedef struct ControlTabInfoRecV1 ControlTabInfoRecV1;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HITabbedView.h`

ControlTemplate

```
struct ControlTemplate {
    Rect controlRect;
    SInt16 controlValue;
    Boolean controlVisible;
    UInt8 fill;
    SInt16 controlMaximum;
    SInt16 controlMinimum;
    SInt16 controlDefProcID;
    SInt32 controlReference;
    Str255 controlTitle;
};
typedef struct ControlTemplate ControlTemplate;
typedef ControlTemplate * ControlTemplatePtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Controls.h`

ControlTrackingRec

```
struct ControlTrackingRec {
    Point startPt;
    EventModifiers modifiers;
    ControlActionUPP action;
};
typedef struct ControlTrackingRec ControlTrackingRec;
typedef ControlTrackingRec * ControlTrackingPtr;
```

Fields

`startPt`

The location of the cursor at the time the mouse button was first pressed, in local coordinates. Your application retrieves this point from the `where` field of the event structure.

`modifiers`

The constant in the `modifiers` field of the event structure specifying the state of the modifier keys and the mouse button at the time the event was posted.

`action`

A pointer to an action function defining what action your application takes while the user holds down the mouse button. The value of the `actionProc` parameter can be a valid `procPtr`, `NULL`, or -1. A value of -1 indicates that the control should either perform auto tracking, or if it is incapable of doing so, do nothing (like `NULL`). See `ControlActionProcPtr` (page 675) for more information about the action function.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Controls.h`

ControlVariant

```
typedef SInt16 ControlVariant;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Controls.h`

DataBrowserCallbacks

```
struct DataBrowserCallbacks {
    UInt32 version
    union {
        struct {
            DataBrowserItemDataUPP itemDataCallback;
            DataBrowserItemCompareUPP itemCompareCallback;
            DataBrowserItemNotificationUPP itemNotificationCallback;
            DataBrowserAddDragItemUPP addDragItemCallback;
            DataBrowserAcceptDragUPP acceptDragCallback;
            DataBrowserReceiveDragUPP receiveDragCallback;
            DataBrowserPostProcessDragUPP postProcessDragCallback;
            DataBrowserItemHelpContentUPP itemHelpContentCallback;
            DataBrowserGetContextualMenuUPP getContextualMenuCallback;
            DataBrowserSelectContextualMenuUPP selectContextualMenuCallback;
        } v1;
    } u;
};
typedef struct DataBrowserCallbacks DataBrowserCallbacks;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
HIDataBrowser.h

DataBrowserCustomCallbacks

```
struct DataBrowserCustomCallbacks {
    UInt32 version
    union {
        struct {
            DataBrowserDrawItemUPP drawItemCallback;
            DataBrowserEditItemUPP editTextCallback;
            DataBrowserHitTestUPP hitTestCallback;
            DataBrowserTrackingUPP trackingCallback;
            DataBrowserItemDragRgnUPP dragRegionCallback;
            DataBrowserItemAcceptDragUPP acceptDragCallback;
            DataBrowserItemReceiveDragUPP receiveDragCallback;
        } v1;
    } u;
};
typedef struct DataBrowserCustomCallbacks DataBrowserCustomCallbacks;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
HIDataBrowser.h

DataBrowserDragFlags

```
typedef  DataBrowserDragFlags;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
HIDataBrowser.h

DataBrowserListViewColumnDesc

```
struct DataBrowserListViewColumnDesc {
    DataBrowserTableViewColumnDesc propertyDesc;
    DataBrowserListViewHeaderDesc headerBtnDesc;
};
typedef struct DataBrowserListViewColumnDesc DataBrowserListViewColumnDesc;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
HIDataBrowser.h

DataBrowserListViewHeaderDesc

```
struct DataBrowserListViewHeaderDesc {
    UInt32 version;
    UInt16 minimumWidth;
    UInt16 maximumWidth;
    SInt16 titleOffset;
    CFStringRef titleString;
    DataBrowserSortOrder initialOrder;
    ControlFontStyleRec btnFontStyle;
    ControlButtonContentInfo btnContentInfo;
};
typedef struct DataBrowserListViewHeaderDesc DataBrowserListViewHeaderDesc;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
HIDataBrowser.h

DataBrowserPropertyDesc

```
struct DataBrowserPropertyDesc {
    DataBrowserPropertyID propertyID;
    DataBrowserPropertyType propertyType;
    DataBrowserPropertyFlags propertyFlags;
};
typedef struct DataBrowserPropertyDesc DataBrowserPropertyDesc;
typedef DataBrowserPropertyDesc DataBrowserTableViewColumnDesc;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HIDataBrowser.h`

DataBrowserPropertyFlags

```
typedef  DataBrowserPropertyFlags;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HIDataBrowser.h`

DataBrowserPropertyPart

```
typedef OSType DataBrowserPropertyPart;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HIDataBrowser.h`

DataBrowserPropertyType

```
typedef OSType DataBrowserPropertyType;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HIDataBrowser.h`

DataBrowserTableViewColumnDesc

```
typedef DataBrowserPropertyDesc DataBrowserTableViewColumnDesc;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
HIDataBrowser.h

DataBrowserTableViewColumnIndex

```
typedef UInt32 DataBrowserTableViewColumnIndex;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
HIDataBrowser.h

DataBrowserTableViewRowIndex

```
typedef UInt32 DataBrowserTableViewRowIndex;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
HIDataBrowser.h

DataBrowserTableViewColumnID

```
typedef DataBrowserPropertyID DataBrowserTableViewColumnID;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
HIDataBrowser.h

DataBrowserViewStyle

```
typedef OSType DataBrowserViewStyle;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
HIDataBrowser.h

DBItemProcDataType

```
typedef void* DBItemProcDataType;
```

Availability
Available in Mac OS X v10.0 through Mac OS X v10.0.

Declared In
`ControlDefinitions.h`

DBRevealItemDataType

```
typedef DataBrowserRevealOptions DBRevealItemDataType;
```

Availability
Available in Mac OS X v10.0 through Mac OS X v10.0.

Declared In
`ControlDefinitions.h`

DBSetSelectionDataType

```
typedef const DataBrowserItemID* DBSetSelectionDataType;
```

Availability
Available in Mac OS X v10.0 through Mac OS X v10.0.

Declared In
`ControlDefinitions.h`

IndicatorDragConstraint

```
struct IndicatorDragConstraint {
    Rect limitRect;
    Rect slopRect;
    DragConstraint axis;
};
typedef struct IndicatorDragConstraint IndicatorDragConstraint;
typedef IndicatorDragConstraint * IndicatorDragConstraintPtr;
```

Fields

`limitRect`
A pointer to a rectangle—whose coordinates should normally coincide with or be contained in the window's content region—delimiting the area in which the user can drag the control's outline.

`slopRect`
A pointer to a rectangle that allows some extra space for the user to move the mouse while still constraining the control within the rectangle specified in the `limitRect` parameter.

`axis`
The axis along which the user may drag the control's outline see "Part Identifier Constants" (page 792).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Controls.h`

IndicatorDragConstraintHandle

```
typedef IndicatorDragConstraintPtr* IndicatorDragConstraintHandle;
```

PopupPrivateData

```
struct PopupPrivateData {
    MenuRef mHandle;
    SInt16 mID;
};
typedef PopupPrivateData PopupPrivateData;
```

Discussion
Use of this structure is not recommended. When the Appearance Manager is available, you should pass the value `kControlPopupButtonMenuHandleTag` in the `tagName` parameter of the `GetControlData` (page 594) function to get the menu handle of a button, and the menu handle and the menu ID of the menu associated with a pop-up menu.

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In
`ControlDefinitions.h`

PopupPrivateDataHandle

```
typedef PopupPrivateDataPtr* PopupPrivateDataHandle;
```

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In
`ControlDefinitions.h`

PopupPrivateDataPtr

```
typedef PopupPrivateData* PopupPrivateDataPtr;
```

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In
`ControlDefinitions.h`

kHIUserPaneClassID

Defines the HIObject class ID for the HIUserPane class.

```
#define kHIUserPaneClassID CFSTR("com.apple.HIUserPane");
```

Availability
Available in Mac OS X v10.4 and later.

Constants

Appearance–compliant Push Button, Radio Button, and Checkbox Control Definition IDs

```
enum {
    kControlPushButtonProc = 368,
    kControlCheckBoxProc = 369,
    kControlRadioButtonProc = 370,
    kControlPushButLeftIconProc = 374,
    kControlPushButRightIconProc = 375
};
```

Constants

`kControlPushButtonProc`
Resource ID: 23

Appearance-compliant push button.This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HIButtonViews.h`.

`kControlCheckBoxProc`
Resource ID: 23

Appearance-compliant checkbox. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HIButtonViews.h`.

`kControlRadioButtonProc`
Resource ID: 23

Appearance-compliant radio button. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HIButtonViews.h`.

`kControlPushButLeftIconProc`
Resource ID: 23

Appearance-compliant push button with a color icon to the left of the control title. (This direction is reversed when the system justification is right to left). The `contrlMax` field of the control structure for this control contains the resource ID of the `'cicn'` resource drawn in the pushbutton. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HIButtonViews.h`.

`kControlPushButRightIconProc`
Resource ID: 23

Appearance-compliant push button with a color icon to right of control title. (This direction is reversed when the system justification is right to left). The `contrlMax` field of the control structure for this control contains the resource ID of the `'cicn'` resource drawn in the pushbutton. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HIButtonViews.h`.

Discussion
When creating a control, your application supplies a control definition ID to one of the Control Manager control-creation functions or to the control resource; see `'CNTL'`. The control definition ID indicates the type of control to create. A control definition ID is an integer that contains the resource ID of a control definition function in its upper 12 bits and a variation code in its lower 4 bits. A control definition ID is derived as follows:

control definition ID = 16 * (`'CDEF'` resource ID) + variation code

A control definition function determines how a control generally looks and behaves. Control definition functions are stored as resources of type `'CDEF'`. Various Control Manager functions call a control definition function whenever they need to perform some control-dependent action, such as drawing the control on the screen. For more information on how to create a control definition function, see "Defining Your Own Control Definition Function".

A control definition function, in turn, can use a variation code to describe variations of the same basic control. For example, all pop-up arrows share the same basic control definition function, which is stored in a resource of type `'CDEF'` and has a resource ID of 12. The standard pop-up arrow is large and points to the right; it has a control definition ID of 192. A variation of this is a large, left-pointing arrow, which has a control definition ID of 193. Still another variation, in which the arrow points up, has a control definition ID of 194.

Your application can use the constants listed here in place of control definition IDs.

If your application contains code that uses the older, pre-Appearance control definition IDs or their constants, your application can use the Appearance Manager to map the old IDs to those for the new, updated controls introduced by the Appearance Manager. In particular, the control definition IDs for pre-Appearance checkboxes, buttons, scroll bars, radio buttons, and pop-up menus will be automatically mapped to Appearance-compliant equivalents.

Asynchronous Arrows Control Definition ID

```
enum {
    kControlChasingArrowsProc = 112
};
```

Constants

`kControlChasingArrowsProc`

Resource ID: 7

Asynchronous arrows. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HIProgressViews.h`.

Discussion

When creating a control, your application supplies a control definition ID to one of the Control Manager control-creation functions or to the control resource; see `'CNTL'`. The control definition ID indicates the type of control to create. A control definition ID is an integer that contains the resource ID of a control definition function in its upper 12 bits and a variation code in its lower 4 bits. A control definition ID is derived as follows:

control definition ID = 16 * (`'CDEF'` resource ID) + variation code

A control definition function determines how a control generally looks and behaves. Control definition functions are stored as resources of type `'CDEF'`. Various Control Manager functions call a control definition function whenever they need to perform some control-dependent action, such as drawing the control on the screen. For more information on how to create a control definition function, see `ControlDefProcPtr` (page 677).

A control definition function, in turn, can use a variation code to describe variations of the same basic control. For example, all pop-up arrows share the same basic control definition function, which is stored in a resource of type `'CDEF'` and has a resource ID of 12. The standard pop-up arrow is large and points to the right; it has a control definition ID of 192. A variation of this is a large, left-pointing arrow, which has a control definition ID of 193. Still another variation, in which the arrow points up, has a control definition ID of 194.

Your application can use the constant listed here in place of a control definition ID.

Bevel Button Behavior Constants

```
enum {
    kControlBehaviorPushbutton = 0,
    kControlBehaviorToggles = 0x0100,
    kControlBehaviorSticky = 0x0200,
    kControlBehaviorSingleValueMenu = 0,
    kControlBehaviorMultiValueMenu = 0x4000,
    kControlBehaviorOffsetContents = 0x8000
};
```

Constants

`kControlBehaviorPushbutton`

Push button (momentary) behavior. The bevel button pops up after being clicked.

Available in Mac OS X v10.0 and later.

Declared in `HIButtonViews.h`.

`kControlBehaviorToggles`
Toggle behavior. The bevel button toggles state automatically when clicked.

Available in Mac OS X v10.0 and later.

Declared in `HIButtonViews.h`.

`kControlBehaviorSticky`
Sticky behavior. Once clicked, the bevel button stays down until your application sets the control's value to 0. This behavior is useful in tool palettes and radio groups.

Available in Mac OS X v10.0 and later.

Declared in `HIButtonViews.h`.

`kControlBehaviorMultiValueMenu`
If this bit is set, the menus are multi-valued. The bevel button does not maintain the menu value as it normally would (requiring that only one item is selected at a time). This allows the user to toggle entries in a menu and have multiple items checked. In this mode, the menu value accessed with the `kControlMenuLastValueTag` will return the value of the last menu item selected.

Available in Mac OS X v10.0 and later.

Declared in `HIButtonViews.h`.

`kControlBehaviorOffsetContents`
Bevel button contents are offset (one pixel down and to the right) when button is pressed.

Available in Mac OS X v10.0 and later.

Declared in `HIButtonViews.h`.

Discussion
You can pass the bevel button behavior constants in the high byte of the `minimumValue` parameter of `NewControl` (page 634) to create a bevel button with a specific behavior.

You can pass the bevel button menu constant, `kControlBehaviorMultiValueMenu`, in the high byte of the `minimumValue` parameter of `NewControl` (page 634) to create a bevel button with a menu of a certain behavior. Bevel buttons with menus have two values: the value of the button and the value of the menu. You can specify the direction of the pop-up menu arrow (down or right) by using the `kControlBevelButtonMenuOnRight` bevel button variant.

The bevel button behavior constants and the bevel button menu constant are available with Appearance Manager 1.0 and later.

Bevel Button Control Data Tag Constants

```
enum {
    kControlBevelButtonContentTag = 'cont',
    kControlBevelButtonTransformTag = 'tran',
    kControlBevelButtonTextAlignTag = 'tali',
    kControlBevelButtonTextOffsetTag = 'toff',
    kControlBevelButtonGraphicAlignTag = 'gali',
    kControlBevelButtonGraphicOffsetTag = 'goff',
    kControlBevelButtonTextPlaceTag = 'tplc',
    kControlBevelButtonMenuValueTag = 'mval',
    kControlBevelButtonMenuHandleTag = 'mhnd',
    kControlBevelButtonMenuRefTag = 'mhnd',
    kControlBevelButtonCenterPopupGlyphTag = 'pglc',
    kControlBevelButtonIsMultiValueMenuTag = 'mult'
};
```

Constants

`kControlBevelButtonContentTag`

Gets or sets a bevel button's content type for drawing see "Bevel Button Menu Constant" (page 727).

Data type returned or set: `ControlButtonContentInfo`structure

Available in Mac OS X v10.0 and later.

Declared in `HIButtonViews.h`.

`kControlBevelButtonTransformTag`

Gets or sets a transform that is added to the standard transform of a bevel button

Data type returned or set: `IconTransformType`

Available in Mac OS X v10.0 and later.

Declared in `HIButtonViews.h`.

`kControlBevelButtonTextAlignTag`

Gets or sets the alignment of text in a bevel button; see "Bevel Button Menu Constant" (page 727).

Data type returned or set: `ControlButtonTextAlignment`

Available in Mac OS X v10.0 and later.

Declared in `HIButtonViews.h`.

`kControlBevelButtonTextOffsetTag`

Gets or sets the number of pixels that text is offset in a bevel button from the button's left or right edge this is used with left, right, or system justification, but it is ignored when the text is center aligned.

Data type returned or set: `SInt16`

Available in Mac OS X v10.0 and later.

Declared in `HIButtonViews.h`.

`kControlBevelButtonGraphicAlignTag`

Gets or sets the alignment of graphics in a bevel button in relation to any text the button may contain; see "Bevel Button Menu Constant" (page 727).

Data type returned or set: `ControlButtonGraphicAlignment`

Available in Mac OS X v10.0 and later.

Declared in `HIButtonViews.h`.

`kControlBevelButtonGraphicOffsetTag`
Gets or sets the horizontal and vertical amounts that a graphic element contained in a bevel button is offset from the button's edges this value is ignored when the graphic is specified to be center aligned on the button. Note that offset values should not be used for bevel buttons with content of type `kControlContentIconRef`, because `IconRef` based icons may change with a theme switch; see "Bevel Button Menu Constant" (page 727).

Data type returned or set: `point`

Available in Mac OS X v10.0 and later.

Declared in `HIButtonViews.h`.

`kControlBevelButtonTextPlaceTag`
Gets or sets the placement of a bevel button's text see "Bevel Button Menu Constant" (page 727).

Data type returned or set: `ControlButtonTextPlacement`

Available in Mac OS X v10.0 and later.

Declared in `HIButtonViews.h`.

`kControlBevelButtonMenuValueTag`
Gets the menu value for a bevel button with an attached menu; see "Bevel Button Menu Constant" (page 727).

Data type returned: `SInt16`

Available in Mac OS X v10.0 and later.

Declared in `HIButtonViews.h`.

`kControlBevelButtonMenuHandleTag`
Gets or sets the menu handle for a bevel button with an attached menu. To set a non-resource-based menu for a bevel button, you must pass in a non-zero value in the `initialValue` parameter of the `NewControl` function, then call the `SetControlData` function with the `kControlBevelButtonMenuHandleTag` constant and the return value from a call to the `NewMenu` function.

Data type returned: `MenuHandle`

Available in Mac OS X v10.0 and later.

Declared in `HIButtonViews.h`.

`kControlBevelButtonCenterPopupGlyphTag`
Gets or sets the position of the pop-up arrow in a bevel button when a pop-up menu is attached.

Data type returned or set: `Boolean`; if `true`, glyph is vertically centered on the right; if `false`, glyph is on the bottom right.

Available in Mac OS X v10.0 and later.

Declared in `HIButtonViews.h`.

`kControlBevelButtonMultiValueMenuTag`
Gets or sets whether the associated menu is a multi-value menu. Available in Mac OS X v10.3 and later.

Data type returned or set: `Boolean`; if `true`, the menu can have multiple selections; otherwise, `false`.

Discussion
You can use the control data tag constants to set or obtain data that is associated with a control. The control data tag constants are passed in the `inTagName` parameters of `SetControlData` (page 652) and `GetControlData` (page 594) to specify the piece of data in a control that you wish to set or get. You can also pass these constants in the `inTagName` parameter of `GetControlDataSize` (page 596) if you wish to

determine the size of variable-length control data. These constants can also be used by custom control definition functions that return the feature bit `kControlSupportsDataAccess` in response to a `kControlMsgGetFeatures` message.

The data that your application sets or obtains can be of various types, dependent upon the control. Therefore, the descriptions of the control data tag constants list the data types for the information that you can set in the `inData` parameter to the `SetControlData` function and that you can get in the `inBuffer` parameter to the `GetControlData` function.

Version Notes
The control data tag constants are available with Appearance Manager 1.0 and later.

Bevel Button Control Definition IDs

```
enum {
    kControlBevelButtonSmallBevelProc = 32,
    kControlBevelButtonNormalBevelProc = 33,
    kControlBevelButtonLargeBevelProc = 34
};
```

Constants

`kControlBevelButtonSmallBevelProc`
Resource ID: 2

Bevel button with a small bevel.

`kControlBevelButtonSmallBevelProc` + `kControlBevelButtonMenuOnRight`

Resource ID: 2 Control Definition ID: 4

Small bevel button with a pop-up menu. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HIButtonViews.h`.

`kControlBevelButtonNormalBevelProc`
Resource ID: 2

Bevel button with a normal bevel. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HIButtonViews.h`.

`kControlBevelButtonLargeBevelProc`
Resource ID: 2

Bevel button with a large bevel. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HIButtonViews.h`.

Discussion
When creating a control, your application supplies a control definition ID to one of the Control Manager control-creation functions or to the control resource; see `'CNTL'`. The control definition ID indicates the type of control to create. A control definition ID is an integer that contains the resource ID of a control definition function in its upper 12 bits and a variation code in its lower 4 bits. A control definition ID is derived as follows:

control definition ID = 16 * ('CDEF' resource ID) + variation code

A control definition function determines how a control generally looks and behaves. Control definition functions are stored as resources of type 'CDEF'. Various Control Manager functions call a control definition function whenever they need to perform some control-dependent action, such as drawing the control on the screen. For more information on how to create a control definition function, see ControlDefProcPtr (page 677).

A control definition function, in turn, can use a variation code to describe variations of the same basic control. For example, all pop-up arrows share the same basic control definition function, which is stored in a resource of type 'CDEF' and has a resource ID of 12. The standard pop-up arrow is large and points to the right; it has a control definition ID of 192. A variation of this is a large, left-pointing arrow, which has a control definition ID of 193. Still another variation, in which the arrow points up, has a control definition ID of 194.

Your application can use the constants listed here in place of control definition IDs.

Bevel Button Graphic Alignment Constants

```
typedef SInt16 ControlButtonGraphicAlignment;
enum {
    kControlBevelButtonAlignSysDirection = -1,
    kControlBevelButtonAlignCenter = 0,
    kControlBevelButtonAlignLeft = 1,
    kControlBevelButtonAlignRight = 2,
    kControlBevelButtonAlignTop = 3,
    kControlBevelButtonAlignBottom = 4,
    kControlBevelButtonAlignTopLeft = 5,
    kControlBevelButtonAlignBottomLeft = 6,
    kControlBevelButtonAlignTopRight = 7,
    kControlBevelButtonAlignBottomRight = 8
};
```

Constants

`kControlBevelButtonAlignSysDirection`
Bevel button graphic is aligned according to the system default script direction (only left or right).
Available in Mac OS X v10.0 and later.
Declared in HIButtonViews.h.

`kControlBevelButtonAlignCenter`
Bevel button graphic is aligned center.
Available in Mac OS X v10.0 and later.
Declared in HIButtonViews.h.

`kControlBevelButtonAlignLeft`
Bevel button graphic is aligned left.
Available in Mac OS X v10.0 and later.
Declared in HIButtonViews.h.

`kControlBevelButtonAlignRight`
Bevel button graphic is aligned right.
Available in Mac OS X v10.0 and later.
Declared in HIButtonViews.h.

`kControlBevelButtonAlignTop`
Bevel button graphic is aligned top.

Available in Mac OS X v10.0 and later.

Declared in `HIButtonViews.h`.

`kControlBevelButtonAlignBottom`
Bevel button graphic is aligned bottom.

Available in Mac OS X v10.0 and later.

Declared in `HIButtonViews.h`.

`kControlBevelButtonAlignTopLeft`
Bevel button graphic is aligned top left.

Available in Mac OS X v10.0 and later.

Declared in `HIButtonViews.h`.

`kControlBevelButtonAlignBottomLeft`
Bevel button graphic is aligned bottom left.

Available in Mac OS X v10.0 and later.

Declared in `HIButtonViews.h`.

`kControlBevelButtonAlignTopRight`
Bevel button graphic is aligned top right.

Available in Mac OS X v10.0 and later.

Declared in `HIButtonViews.h`.

`kControlBevelButtonAlignBottomRight`
Bevel button graphic is aligned bottom right.

Available in Mac OS X v10.0 and later.

Declared in `HIButtonViews.h`.

Discussion
You can use the `ControlButtonGraphicAlignment` constants to specify the alignment of icons and pictures in bevel buttons. These constants are passed in the `inData` parameter of `SetControlData` (page 652) and returned by `GetControlData` (page 594).

Version Notes
The `ControlButtonGraphicAlignment` constants are available with Appearance Manager 1.0 and later.

Bevel Button Menu Constant

```
enum {
    kControlBehaviorCommandMenu = 0x2000
};
```

Constants
`kControlBehaviorCommandMenu`
If this bit is set, the menu contains commands, not choices, and should not be marked with a checkmark. If this bit is set, it overrides the `kControlBehaviorMultiValueMenu` bit. This constant is only available with Appearance 1.0.1 and later.

Available in Mac OS X v10.0 and later.

Declared in `HIButtonViews.h`.

Discussion
You can pass one or more bevel button menu constants in the high byte of the `minimumValue` parameter of `NewControl` (page 634) to create a bevel button with a menu of a certain behavior. Bevel buttons with menus have two values: the value of the button and the value of the menu. You can specify the direction of the pop-up menu arrow (down or right) by using the `kControlBevelButtonMenuOnRight` bevel button variant.

Bevel Button Menu Control Data Tag Constants

```
enum {
    kControlBevelButtonLastMenuTag = 'lmnu',
    kControlBevelButtonMenuDelayTag = 'mdly'
};
```

Constants

`kControlBevelButtonLastMenuTag`
Gets the menu ID of the last menu selected in the submenu or main menu. Available with Appearance Manager 1.0.1 and later.

Data type returned: `SInt16`

Available in Mac OS X v10.0 and later.

Declared in `HIButtonViews.h`.

`kControlBevelButtonMenuDelayTag`
Gets or sets the delay (in number of ticks) before the menu is displayed. Available with Appearance Manager 1.0.1 and later.

Data type returned or set: `SInt32`

Available in Mac OS X v10.0 and later.

Declared in `HIButtonViews.h`.

Discussion
You can use the control data tag constants to set or obtain data that is associated with a control. The control data tag constants are passed in the `inTagName` parameters of `SetControlData` (page 652) and `GetControlData` (page 594) to specify the piece of data in a control that you wish to set or get. You can also pass these constants in the `inTagName` parameter of `GetControlDataSize` (page 596) if you wish to determine the size of variable-length control data. These constants can also be used by custom control definition functions that return the feature bit `kControlSupportsDataAccess` in response to a `kControlMsgGetFeatures` message.

The data that your application sets or obtains can be of various types, dependent upon the control. Therefore, the descriptions of the control data tag constants list the data types for the information that you can set in the `inData` parameter to the `SetControlData` function and that you can get in the `inBuffer` parameter to the `GetControlData` function.

Bevel Button Text Alignment Constants

```
typedef SInt16 ControlButtonTextAlignment;
enum {
    kControlBevelButtonAlignTextSysDirection = teFlushDefault,
    kControlBevelButtonAlignTextCenter = teCenter,
    kControlBevelButtonAlignTextFlushRight = teFlushRight,
    kControlBevelButtonAlignTextFlushLeft = teFlushLeft
};
```

Constants

`kControlBevelButtonAlignTextSysDirection`
Bevel button text is aligned according to the current script direction (left or right).

Available in Mac OS X v10.0 and later.

Declared in `HIButtonViews.h`.

`kControlBevelButtonAlignTextCenter`
Bevel button text is aligned center.

Available in Mac OS X v10.0 and later.

Declared in `HIButtonViews.h`.

`kControlBevelButtonAlignTextFlushRight`
Bevel button text is aligned flush right.

Available in Mac OS X v10.0 and later.

Declared in `HIButtonViews.h`.

`kControlBevelButtonAlignTextFlushLeft`
Bevel button text is aligned flush left.

Available in Mac OS X v10.0 and later.

Declared in `HIButtonViews.h`.

Discussion

You can use the `ControlButtonTextAlignment` constants to specify the alignment of text in a bevel button. These constants are passed in the `inData` parameter of `SetControlData` (page 652) and returned by `GetControlData` (page 594).

Version Notes

The `ControlButtonTextAlignment` constants are available with Appearance Manager 1.0 and later.

Bevel Button Text Placement Constants

```
typedef SInt16 ControlButtonTextPlacement;
enum {
    kControlBevelButtonPlaceSysDirection = -1,
    kControlBevelButtonPlaceNormally = 0,
    kControlBevelButtonPlaceToRightOfGraphic = 1,
    kControlBevelButtonPlaceToLeftOfGraphic = 2,
    kControlBevelButtonPlaceBelowGraphic = 3,
    kControlBevelButtonPlaceAboveGraphic = 4
};
```

Constants

`kControlBevelButtonPlaceSysDirection`
Bevel button text is placed according to the system default script direction.
Available in Mac OS X v10.0 and later.
Declared in `HIButtonViews.h`.

`kControlBevelButtonPlaceNormally`
Bevel button text is centered.
Available in Mac OS X v10.0 and later.
Declared in `HIButtonViews.h`.

`kControlBevelButtonPlaceToRightOfGraphic`
Bevel button text is placed to the right of the graphic.
Available in Mac OS X v10.0 and later.
Declared in `HIButtonViews.h`.

`kControlBevelButtonPlaceToLeftOfGraphic`
Bevel button text is placed to the left of the graphic.
Available in Mac OS X v10.0 and later.
Declared in `HIButtonViews.h`.

`kControlBevelButtonPlaceBelowGraphic`
Bevel button text is placed below the graphic.
Available in Mac OS X v10.0 and later.
Declared in `HIButtonViews.h`.

`kControlBevelButtonPlaceAboveGraphic`
Bevel button text is placed above the graphic.
Available in Mac OS X v10.0 and later.
Declared in `HIButtonViews.h`.

Discussion

You can use the `ControlButtonTextPlacement` constants to specify the placement of text in a bevel button, in relation to an icon or picture. These constants are passed in the `inData` parameter of `SetControlData` (page 652) and returned by `GetControlData` (page 594). They can be used in conjunction with bevel button text and graphic alignment constants to create, for example, a button where the graphic and text are left justified with the text below the graphic.

Version Notes

The `ControlButtonTextPlacement` constants are available with Appearance Manager 1.0 and later.

Checkbox and Radio Button AutoToggle Control Definition IDs

```
enum {
    kControlCheckBoxAutoToggleProc = 371,
    kControlRadioButtonAutoToggleProc = 372
};
```

Constants

`kControlCheckBoxAutoToggleProc`
Identifies a checkbox control (`'CDEF'` resource ID 23) that automatically changes among its various states (on, off, mixed) in response to user actions. Your application must only call the function `GetControl32BitValue` (page 590) to get the checkbox's new state—there is no need to manually change the control's value after tracking successfully.

Available in Mac OS X v10.0 and later.

Declared in `HIButtonViews.h`.

`kControlRadioButtonAutoToggleProc`
Identifies a radio button control (`'CDEF'` resource ID 23) that automatically changes among its various states (on, off, mixed) in response to user actions. Your application must only call the function `GetControl32BitValue` (page 590) to get the radio button's new state—there is no need to manually change the control's value after tracking successfully.

Available in Mac OS X v10.0 and later.

Declared in `HIButtonViews.h`.

Discussion
The Mac OS 8.5 Control Manager defines these new control definition IDs.

When creating a control, your application supplies a control definition ID to one of the Control Manager control-creation functions or to the control resource; see `'CNTL'`. The control definition ID indicates the type of control to create. A control definition ID is an integer that contains the resource ID of a control definition function in its upper 12 bits and a variation code in its lower 4 bits. A control definition ID is derived as follows:

control definition ID = 16 * (`'CDEF'` resource ID) + variation code

A control definition function determines how a control generally looks and behaves. Control definition functions are stored as resources of type `'CDEF'`. Various Control Manager functions call a control definition function whenever they need to perform some control-dependent action, such as drawing the control on the screen. For more information on how to create a control definition function, see `ControlDefProcPtr` (page 677).

A control definition function, in turn, can use a variation code to describe variations of the same basic control. For example, all pop-up arrows share the same basic control definition function, which is stored in a resource of type `'CDEF'` and has a resource ID of 12. The standard pop-up arrow is large and points to the right; it has a control definition ID of 192. A variation of this is a large, left-pointing arrow, which has a control definition ID of 193. Still another variation, in which the arrow points up, has a control definition ID of 194.

Your application can use the constants listed here in place of control definition IDs. These constants, and their associated IDs, are not supported unless the Appearance Manager is available.

Checkbox Value Constants

```
enum {
    kControlCheckBoxUncheckedValue = 0,
    kControlCheckBoxCheckedValue = 1,
    kControlCheckBoxMixedValue = 2
};
```

Constants

`kControlCheckBoxUncheckedValue`
The checkbox is unchecked.

Available in Mac OS X v10.0 and later.

Declared in `HIButtonViews.h`.

`kControlCheckBoxCheckedValue`
The checkbox is checked.

Available in Mac OS X v10.0 and later.

Declared in `HIButtonViews.h`.

`kControlCheckBoxMixedValue`
Mixed value. Indicates that a setting is on for some elements in a selection and off for others. This state only applies to standard Appearance-compliant checkboxes.

Available in Mac OS X v10.0 and later.

Declared in `HIButtonViews.h`.

Discussion
The checkbox value constants specify the value of a standard checkbox control and are passed in the `newValue` parameter of `SetControlValue` (page 661) and are returned by `GetControlValue` (page 606) .

Version Notes
The checkbox value constants are changed with Appearance Manager 1.0 to support mixed-value checkboxes.

Clock Control Data Tag Constants

```
enum {
    kControlClockLongDateTag = 'date',
    kControlClockFontStyleTag = kControlFontStyleTag,
    kControlClockAnimatingTag = 'anim'
};
```

Constants

`kControlClockLongDateTag`
Gets or sets the clock control's time or date.

Data type returned or set: `LongDateRec` structure. Note that depending on the variant of clock control specified, some of the fields in the `longDateRec` structure may not be valid. For example, if the control variant displays only a non-live user-adjustable date, the hour and minute fields are not valid and will contain garbage.

Available in Mac OS X v10.0 and later.

Declared in `HIClockView.h`.

Discussion
You can use the control data tag constants to set or obtain data that is associated with a control. The control data tag constants are passed in the `inTagName` parameters of `SetControlData` (page 652) and `GetControlData` (page 594) to specify the piece of data in a control that you wish to set or get. You can also pass these constants in the `inTagName` parameter of `GetControlDataSize` (page 596) if you wish to determine the size of variable-length control data. These constants can also be used by custom control definition functions that return the feature bit `kControlSupportsDataAccess` in response to a `kControlMsgGetFeatures` message.

The data that your application sets or obtains can be of various types, dependent upon the control. Therefore, the descriptions of the control data tag constants list the data types for the information that you can set in the `inData` parameter to the `SetControlData` function and that you can get in the `inBuffer` parameter to the `GetControlData` function.

Version Notes
The control data tag constants are available with Appearance Manager 1.0 and later.

Clock Control Definition IDs

```
enum {
    kControlClockTimeProc = 240,
    kControlClockTimeSecondsProc = 241,
    kControlClockDateProc = 242,
    kControlClockMonthYearProc = 243
};
```

Constants

`kControlClockTimeProc`
Resource ID: 15

Clock control displaying hour/minutes. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HIClockView.h`.

`kControlClockTimeSecondsProc`
Resource ID: 15

Clock control displaying hours/minutes/seconds. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HIClockView.h`.

`kControlClockDateProc`
Resource ID: 15

Clock control displaying date/month/year. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HIClockView.h`.

`kControlClockMonthYearProc`

Resource ID: 15

Clock control displaying month/year. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HIClockView.h`.

Discussion

When creating a control, your application supplies a control definition ID to one of the Control Manager control-creation functions or to the control resource; see `'CNTL'`. The control definition ID indicates the type of control to create. A control definition ID is an integer that contains the resource ID of a control definition function in its upper 12 bits and a variation code in its lower 4 bits. A control definition ID is derived as follows:

control definition ID = 16 * (`'CDEF'` resource ID) + variation code

A control definition function determines how a control generally looks and behaves. Control definition functions are stored as resources of type `'CDEF'`. Various Control Manager functions call a control definition function whenever they need to perform some control-dependent action, such as drawing the control on the screen. For more information on how to create a control definition function, see `ControlDefProcPtr` (page 677).

A control definition function, in turn, can use a variation code to describe variations of the same basic control. For example, all pop-up arrows share the same basic control definition function, which is stored in a resource of type `'CDEF'` and has a resource ID of 12. The standard pop-up arrow is large and points to the right; it has a control definition ID of 192. A variation of this is a large, left-pointing arrow, which has a control definition ID of 193. Still another variation, in which the arrow points up, has a control definition ID of 194.

Your application can use the constants listed here in place of control definition IDs.

Clock Value Flag Constants

```
typedef UInt32 ControlClockFlags;
enum {
    kControlClockFlagStandard = 0,
    kControlClockNoFlags = 0,
    kControlClockFlagDisplayOnly = 1,
    kControlClockIsDisplayOnly = 1,
    kControlClockFlagLive = 2,
    kControlClockIsLive = 2
};
```

Constants

`kControlClockNoFlags`

Indicates that clock is editable but does not display the current “live” time.

Available in Mac OS X v10.0 and later.

Declared in `HIClockView.h`.

`kControlClockIsDisplayOnly`

When only this bit is set, the clock is not editable. When this bit and the `kControlClockIsLive` bit is set, the clock automatically updates on idle (clock will have the current time).

Available in Mac OS X v10.0 and later.

Declared in `HIClockView.h`.

`kControlClockIsLive`

When only this bit is set, the clock automatically updates on idle and any changes to the clock affect the system clock. When this bit and the `kControlClockIsDisplayOnly` bit is set, the clock automatically updates on idle (clock will have the current time), but is not editable.

Available in Mac OS X v10.0 and later.

Declared in `HIClockView.h`.

Discussion
You can use the clock value flag constants to specify behaviors for a clock control. You can pass one or more of these mask constants into the control (`'CNTL'`) resource or in the `initialValue` parameter of `NewControl` (page 634). Note that the standard clock control is editable and supports keyboard focus. Also, the little arrows that allow manipulation of the date and time are part of the control, not a separate embedded little arrows control. The clock value flag constants are available with Appearance Manager 1.0 and later.

Control Definition Message Constants

The Control Manager passes constants of type `ControlDefProcMessage` to Indicate the action your control definition function must perform.

```
enum {
    drawCntl = 0,
    testCntl = 1,
    calcCRgns = 2,
    initCntl = 3,
    dispCntl = 4,
    posCntl = 5,
    thumbCntl = 6,
    dragCntl = 7,
    autoTrack = 8,
    calcCntlRgn = 10,
    calcThumbRgn = 11,
    drawThumbOutline = 12,
    kControlMsgDrawGhost = 13,
    kControlMsgCalcBestRect = 14,
    kControlMsgHandleTracking = 15,
    kControlMsgFocus = 16,
    kControlMsgKeyDown = 17,
    kControlMsgIdle = 18,
    kControlMsgGetFeatures = 19,
    kControlMsgSetData = 20,
    kControlMsgGetData = 21,
    kControlMsgActivate = 22,
    kControlMsgSetUpBackground = 23,
    kControlMsgCalcValueFromPos = 26,
    kControlMsgTestNewMsgSupport = 27,
    kControlMsgSubValueChanged = 25,
    kControlMsgSubControlAdded = 28,
    kControlMsgSubControlRemoved = 29,
    kControlMsgApplyTextColor = 30,
    kControlMsgGetRegion = 31,
    kControlMsgFlatten = 32,
    kControlMsgSetCursor = 33,
    kControlMsgDragEnter = 38,
    kControlMsgDragLeave = 39,
    kControlMsgDragWithin = 40,
    kControlMsgDragReceive = 41,
    kControlMsgDisplayDebugInfo = 46,
    kControlMsgContextualMenuClick = 47,
    kControlMsgGetClickActivation = 48
};
```

Constants

`drawCntl`

Draw the entire control or part of a control.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Controls.h`.

`testCntl`

Test where the mouse has been pressed.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Controls.h`.

`calcCRgns`
: Calculate the region for the control or the indicator in 24-bit systems. This message is obsolete in Mac OS 7.6 and later.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Controls.h`.

`initCntl`
: Perform additional control initialization.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Controls.h`.

`dispCntl`
: Perform additional control disposal actions.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Controls.h`.

`posCntl`
: Move and update the indicator setting.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Controls.h`.

`thumbCntl`
: Calculate the parameters for dragging the indicator.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Controls.h`.

`dragCntl`
: Perform customized dragging (of the control or its indicator).

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Controls.h`.

`autoTrack`
: Execute the specified action function.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Controls.h`.

`calcCntlRgn`
: Calculate the control region in 32-bit systems.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Controls.h`.

`calcThumbRgn`
Calculate the indicator region in 32-bit systems.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Controls.h`.

`kControlMsgDrawGhost`
Draw a ghost image of the indicator.
Available with Appearance Manager 1.0 and later.
Not available to 64-bit applications.
Declared in `Controls.h`.

`kControlMsgCalcBestRect`
Calculate the optimal control rectangle.
Available with Appearance Manager 1.0 and later.
Not available to 64-bit applications.
Declared in `Controls.h`.

`kControlMsgHandleTracking`
Perform custom tracking.
Available with Appearance Manager 1.0 and later.
Not available to 64-bit applications.
Declared in `Controls.h`.

`kControlMsgFocus`
Handle keyboard focus.
Available with Appearance Manager 1.0 and later.
Not available to 64-bit applications.
Declared in `Controls.h`.

`kControlMsgKeyDown`
Handle keyboard events.
Available with Appearance Manager 1.0 and later.
Not available to 64-bit applications.
Declared in `Controls.h`.

`kControlMsgIdle`
Perform idle processing.
Available with Appearance Manager 1.0 and later.
Not available to 64-bit applications.
Declared in `Controls.h`.

`kControlMsgGetFeatures`
Specify which Appearance-compliant messages are supported.
Available with Appearance Manager 1.0 and later.
Not available to 64-bit applications.
Declared in `Controls.h`.

`kControlMsgSetData`
Set control-specific data.

Available with Appearance Manager 1.0 and later.

Not available to 64-bit applications.

Declared in `Controls.h`.

`kControlMsgGetData`
Get control-specific data.

Available with Appearance Manager 1.0 and later.

Not available to 64-bit applications.

Declared in `Controls.h`.

`kControlMsgActivate`
Handle activate and deactivate events.

Available with Appearance Manager 1.0 and later.

Not available to 64-bit applications.

Declared in `Controls.h`.

`kControlMsgSetUpBackground`
Set the control's background color or pattern (only available if the control supports embedding).

Available with Appearance Manager 1.0 and later.

Not available to 64-bit applications.

Declared in `Controls.h`.

`kControlMsgCalcValueFromPos`
Support live feedback while dragging the indicator and calculate the control value based on the new indicator region.

Available with Appearance Manager 1.0 and later.

Not available to 64-bit applications.

Declared in `Controls.h`.

`kControlMsgTestNewMsgSupport`
Specify whether Appearance-compliant messages are supported.

Available with Appearance Manager 1.0 and later.

Not available to 64-bit applications.

Declared in `Controls.h`.

`kControlMsgSubValueChanged`
Be informed that the value of a subcontrol embedded in the control has changed; this message is useful for radio groups.

Available with Appearance 1.0.1 and later.

Not available to 64-bit applications.

Declared in `Controls.h`.

`kControlMsgSubControlAdded`
Be informed that a subcontrol has been embedded in the control.

Available with Appearance 1.0.1 and later.

Not available to 64-bit applications.

Declared in `Controls.h`.

`kControlMsgSubControlRemoved`

Be informed that a subcontrol is about to be removed from the control.

Available with Appearance 1.0.1 and later.

Not available to 64-bit applications.

Declared in `Controls.h`.

`kControlMsgApplyTextColor`

Set the foreground color to be consistent with the current drawing environment and suitable for display against the background color or pattern. To indicate that your control definition function supports this message, set the `kControlHasSpecialBackground` feature bit. When this message is sent, the Control Manager passes a pointer to a structure of type `ControlGetRegionRec` (page 705) in your control definition function's `param` parameter. The Control Manager sets the `ControlApplyTextColorRec` structure to contain data describing the current drawing environment. Your control definition function is responsible for using this data to apply a text color which is consistent with the current theme and optimally readable on the control's background. Your control definition function should return 0 as the function result.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Controls.h`.

`kControlMsgGetRegion`

Obtain the region occupied by the specified control part. To indicate that your control definition function supports this message, set the `kControlSupportsGetRegion` feature bit. When this message is sent, the Control Manager passes a pointer to a structure of type `ControlGetRegionRec` (page 705) in your control definition function's `param` parameter. Your control definition function is responsible for setting the `region` field of the `ControlGetRegionRec` structure to the region that contains the control part which the Control Manager specifies in the `part` field. Your control definition function return a result code of type `OSStatus` as the function result.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Controls.h`.

Discussion

The Control Manager may pass these constants in the `message` parameter of your control definition function to specify the actions that your function must perform. For more information, see `ControlDefProcPtr` (page 677).

Control Features Constants

```
enum {
    kControlSupportsGhosting = 1 << 0,
    kControlSupportsEmbedding = 1 << 1,
    kControlSupportsFocus = 1 << 2,
    kControlWantsIdle = 1 << 3,
    kControlWantsActivate = 1 << 4,
    kControlHandlesTracking = 1 << 5,
    kControlSupportsDataAccess = 1 << 6,
    kControlHasSpecialBackground = 1 << 7,
    kControlGetsFocusOnClick = 1 << 8,
    kControlSupportsCalcBestRect = 1 << 9,
    kControlSupportsLiveFeedback = 1 << 10,
    kControlHasRadioBehavior = 1 << 11,
    kControlSupportsDragAndDrop = 1 << 12,
    kControlAutoToggles = 1 << 14,
    kControlSupportsGetRegion = 1 << 17,
    kControlSupportsFlattening = 1 << 19,
    kControlSupportsSetCursor = 1 << 20,
    kControlSupportsContextualMenus = 1 << 21,
    kControlSupportsClickActivation = 1 << 22,
    kControlIdlesWithTimer = 1 << 23
};
```

Constants

`kControlSupportsGhosting`
If this bit (bit 0) is set, the control definition function supports the `kControlMsgDrawGhost` message.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kControlSupportsEmbedding`
If this bit (bit 1) is set, the control definition function supports the `kControlMsgSubControlAdded` and `kControlMsgSubControlRemoved`messages.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kControlSupportsFocus`
If this bit (bit 2) is set, the control definition function supports the `kControlMsgKeyDown` message. If this bit and the `kControlGetsFocusOnClick` bit are set, the control definition function supports the `kControlMsgFocus` message.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kControlWantsIdle`
If this bit (bit 3) is set, the control definition function supports the `kControlMsgIdle` message.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kControlWantsActivate`
If this bit (bit 4) is set, the control definition function supports the `kControlMsgActivate` message.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kControlHandlesTracking`
If this bit (bit 5) is set, the control definition function supports the `kControlMsgHandleTracking` message.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kControlSupportsDataAccess`
If this bit (bit 6) is set, the control definition function supports the `kControlMsgGetData` and `kControlMsgSetData` messages.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kControlHasSpecialBackground`
If this bit (bit 7) is set, the control definition function supports the `kControlMsgSetUpBackground` message.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kControlGetsFocusOnClick`
If this bit (bit 8) and the `kControlSupportsFocus` bit are set, the control definition function supports the `kControlMsgFocus` message.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kControlSupportsCalcBestRect`
If this bit (bit 9) is set, the control definition function supports the `kControlMsgCalcBestRect` message.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kControlSupportsLiveFeedback`
If this bit (bit 10) is set, the control definition function supports the `kControlMsgCalcValueFromPos` message.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kControlHasRadioBehavior`
If this bit (bit 11) is set, the control definition function supports radio button behavior and can be embedded in a radio group control. This constant is available with Appearance 1.0.1 and later.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kControlAutoToggles`
If the bit specified by this mask is set, the control definition function supports automatically changing among various states (on, off, mixed) in response to user actions.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kControlSupportsGetRegion`
If the bit specified by this mask is set, the control definition function supports the `kControlMsgGetRegion` message, described in "Control Definition Message Constants" (page 735).

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

Discussion
If your control definition function supports Appearance-compliant messages, it should return a bit field of the features it supports, composed of one or more of these bits.

Control Focus Part Code Constants

```
enum {
    kControlFocusNoPart = 0,
    kControlFocusNextPart = -1,
    kControlFocusPrevPart = -2
};
```

Constants
`kControlFocusNoPart`
Your control definition function should relinquish its focus and return `kControlFocusNoPart`. It might respond by deactivating its text edit handle and erasing its focus ring. If the control is at the end of its subparts, it should return `kControlFocusNoPart`. This tells the focusing mechanism to jump to the next control that supports focus.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kControlFocusNextPart`
Your control definition function should change keyboard focus to its next part, the entire control, or remove keyboard focus from the control, depending upon the circumstances.

For multiple part controls that already had keyboard focus, the next part of the control would receive keyboard focus when `kControlFocusNextPart` was passed in the `param` parameter. For example, a clock control with keyboard focus would change its focus to the left-most element of the control (the month field).

For single-part controls that did not have keyboard focus and are now receiving it, the entire control would receive keyboard focus when `kControlFocusNextPart` was passed in the `param` parameter.

For single-part controls that already had keyboard focus and are now losing it, the entire control would lose keyboard focus.

If you are passed `kControlFocusNextPart` and have run out of parts, return `kControlFocusNoPart` to indicate that the user tabbed past the control.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kControlFocusPrevPart`

Your control definition function should change keyboard focus to its previous part, the entire control, or remove keyboard focus from the control, depending upon the circumstances.

For multiple part controls that already had keyboard focus, the previous part of the control would receive keyboard focus when `kControlFocusPrevPart` was passed in the `param` parameter. For example, a clock control with keyboard focus would change its focus to the right-most element of the control (the year field).

For single-part controls that did not have keyboard focus and are now receiving it, the entire control would receive keyboard focus when `kControlFocusNextPart` was passed in the `param` parameter.

For single-part controls that already had keyboard focus and are now losing it, the entire control would lose keyboard focus.

If you are passed `kControlFocusPrevPart` and have run out of parts, return `kControlFocusNoPart` to indicate that the user tabbed past the control.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

Control Font Style and Key Filter Data Tag Constants

```
enum {
    kControlFontStyleTag = 'font',
    kControlKeyFilterTag = 'fltr',
    kControlKindTag = 'kind',
    kControlSizeTag = 'size'
};
```

Constants

`kControlFontStyleTag`

Sent with a pointer to a ControlKind record to be filled in. Only valid for GetControlData.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kControlKeyFilterTag`

Gets or sets the key filter function for controls that handle filtered input (includes editable text and list box).

Data type returned or set: `ControlKeyFilterUPP`

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

Discussion

You can use the control data tag constants to set or obtain data that is associated with a control. The control data tag constants are passed in the `inTagName` parameters of `SetControlData` (page 652) and `GetControlData` (page 594) to specify the piece of data in a control that you wish to set or get. You can also pass these constants in the `inTagName` parameter of `GetControlDataSize` (page 596) if you wish to determine the size of variable-length control data (e.g., text in an editable text control). These constants can also be used by custom control definition functions that return the feature bit `kControlSupportsDataAccess` in response to a `kControlMsgGetFeatures` message.

The data that your application sets or obtains can be of various types, dependent upon the control. Therefore, the descriptions of the control data tag constants list the data types for the information that you can set in the `inData` parameter to the `SetControlData` function and that you can get in the `inBuffer` parameter to the `GetControlData` function.

Version Notes
The control data tag constants are available with Appearance Manager 1.0 and later.

Control Font Style Flag Constants

```
enum {
    kControlUseFontMask = 0x0001,
    kControlUseFaceMask = 0x0002,
    kControlUseSizeMask = 0x0004,
    kControlUseForeColorMask = 0x0008,
    kControlUseBackColorMask = 0x0010,
    kControlUseModeMask = 0x0020,
    kControlUseJustMask = 0x0040,
    kControlUseAllMask = 0x00FF,
    kControlAddFontSizeMask = 0x0100
};
```

Constants

`kControlUseFontMask`
If the `kControlUseFontMask` flag is set (bit 0), the `font` field of the control font style structure is applied to the control.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kControlUseFaceMask`
If the `kControlUseFaceMask` flag is set (bit 1), the `style` field of the control font style structure is applied to the control. This flag is ignored if you specify a meta font value; see "Meta Font Constants" (page 791).

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kControlUseSizeMask`
If the `kControlUseSizeMask` flag is set (bit 2), the `size` field of the control font style structure is applied to the control. This flag is ignored if you specify a meta font value; see "Meta Font Constants" (page 791).

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kControlUseForeColorMask`
If the `kControlUseForeColorMask` flag is set (bit 3), the `foreColor` field of the control font style structure is applied to the control. This flag only applies to static text controls.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kControlUseBackColorMask`
If the `kControlUseBackColorMask` flag is set (bit 4), the `backColor` field of the control font style structure is applied to the control. This flag only applies to static text controls.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kControlUseModeMask`
If the `kControlUseModeMask` flag is set (bit 5), the text mode specified in the `mode` field of the control font style structure is applied to the control.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kControlUseJustMask`
If the `kControlUseJustMask` flag is set (bit 6), the `just` field of the control font style structure is applied to the control.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kControlUseAllMask`
If `kControlUseAllMask` is used, all flags in this mask will be set except `kControlUseAddFontSizeMask`.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kControlAddFontSizeMask`
If the `kControlUseAddFontSizeMask` flag is set (bit 8), the Dialog Manager will add a specified font size to the `size` field of the control font style structure. This flag is ignored if you specify a meta font value; see "Meta Font Constants" (page 791).

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

Discussion
You can pass one or more control font style flag constants in the `flags` field of the control font style structure to specify the field(s) of the structure that should be applied to the control; see ControlFontStyleRec (page 704). If none of the flags are set, the control uses the system font unless a control variant specifies use of a window font.

Version Notes
These control font style flag constants are available with Appearance Manager 1.0 and later.

Control Key Script Behavior Constants

```
enum {
    kControlKeyScriptBehaviorAllowAnyScript = 'any ',
    kControlKeyScriptBehaviorPrefersRoman = 'prmn',
    kControlKeyScriptBehaviorRequiresRoman = 'rrmn'
};
```

Constants

`kControlKeyScriptBehaviorAllowAnyScript`

Does not change the current keyboard and allows the user to change the keyboard at will. This is the default for non-password fields.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kControlKeyScriptBehaviorPrefersRoman`

Changes the current keyboard to Roman whenever the editable text field receives focus but allows the user to change the keyboard at will. This is the default for password fields.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kControlKeyScriptBehaviorRequiresRoman`

Changes the current keyboard to Roman whenever the editable text field receives focus and does not allow the user to change the keyboard.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

Discussion

With the Mac OS 8.5 Control Manager, you can use these constants of type `ControlKeyScriptBehavior` to specify the kind of behavior to be used in an editable text control with respect to changing and locking the keyboard menu as the field is focused. The `ControlKeyScriptBehavior` constants are set and retrieved with the `kControlEditTextKeyScriptBehaviorTag` control data tag constant; for details on `kControlEditTextKeyScriptBehaviorTag`, see "Editable Text Control Data Tag Constants" (page 756).

Control Part Code Constants

```
enum {
    kControlLabelPart = 1,
    kControlMenuPart = 2,
    kControlTrianglePart = 4,
    kControlEditTextPart = 5,
    kControlPicturePart = 6,
    kControlIconPart = 7,
    kControlClockPart = 8,
    kControlListBoxPart = 24,
    kControlListBoxDoubleClickPart = 25,
    kControlImageWellPart = 26,
    kControlRadioGroupPart = 27,
    kControlButtonPart = 10,
    kControlCheckBoxPart = 11,
    kControlRadioButtonPart = 11,
    kControlUpButtonPart = 20,
    kControlDownButtonPart = 21,
    kControlPageUpPart = 22,
    kControlPageDownPart = 23,
    kControlClockHourDayPart = 9,
    kControlClockMinuteMonthPart = 10,
    kControlClockSecondYearPart = 11,
    kControlClockAMPMPart = 12,
    kControlDataBrowserPart = 24,
    kControlDataBrowserDraggedPart = 25
};
```

Constants

`kControlLabelPart`
Identifies the label of a pop-up menu control.
Available in Mac OS X v10.0 and later.
Declared in `ControlDefinitions.h`.

`kControlMenuPart`
Identifies the menu of a pop-up menu control. For bevel buttons with a menu attached, this part code specifies a menu item of the bevel button.
Available in Mac OS X v10.0 and later.
Declared in `ControlDefinitions.h`.

`kControlTrianglePart`
Identifies a disclosure triangle control.
Available in Mac OS X v10.0 and later.
Declared in `ControlDefinitions.h`.

`kControlEditTextPart`
Identifies an editable text control.
Available with Appearance Manager 1.0 and later.
Declared in `ControlDefinitions.h`.

`kControlPicturePart`
Identifies a picture control.
Available with Appearance Manager 1.0 and later.
Declared in `ControlDefinitions.h`.

`kControlIconPart`
Identifies an icon control.
Available with Appearance Manager 1.0 and later.
Declared in `ControlDefinitions.h`.

`kControlClockPart`
Identifies a clock control.
Available with Appearance Manager 1.0 and later.
Declared in `ControlDefinitions.h`.

`kControlListBoxPart`
Identifies a list box control.
Available with Appearance Manager 1.0 and later.
Declared in `ControlDefinitions.h`.

`kControlListBoxDoubleClickPart`
Identifies a double-click in a list box control.
Available with Appearance Manager 1.0 and later.
Declared in `ControlDefinitions.h`.

`kControlImageWellPart`
Identifies an image well control.
Available with Appearance Manager 1.0 and later.
Declared in `ControlDefinitions.h`.

`kControlRadioGroupPart`
Identifies a radio group control.
Available with Appearance Manager 1.0.2 and later.
Declared in `ControlDefinitions.h`.

`kControlButtonPart`
Identifies either a push button or bevel button control. For bevel buttons with a menu attached, this part code specifies the button but not the attached menu.
Available in Mac OS X v10.0 and later.
Declared in `ControlDefinitions.h`.

`kControlCheckBoxPart`
Identifies a checkbox control.
Available in Mac OS X v10.0 and later.
Declared in `ControlDefinitions.h`.

`kControlRadioButtonPart`
Identifies a radio button control.
Available in Mac OS X v10.0 and later.
Declared in `ControlDefinitions.h`.

`kControlUpButtonPart`
Identifies the up button of a scroll bar control (the arrow at the top or the left).
Available in Mac OS X v10.0 and later.
Declared in `ControlDefinitions.h`.

`kControlDownButtonPart`
Identifies the down button of a scroll bar control (the arrow at the right or the bottom).

Available in Mac OS X v10.0 and later.

Declared in `ControlDefinitions.h`.

`kControlPageUpPart`
Identifies the page-up part of a scroll bar control.

Available in Mac OS X v10.0 and later.

Declared in `ControlDefinitions.h`.

`kControlPageDownPart`
Identifies the page-down part of a scroll bar control.

Available in Mac OS X v10.0 and later.

Declared in `ControlDefinitions.h`.

`kControlClockHourDayPart`
Identifies the part of a clock control that contains the hour or the day. The Mac OS 8.5 Control Manager defines this new control part code constant.

Available in Mac OS X v10.0 and later.

Declared in `ControlDefinitions.h`.

`kControlClockMinuteMonthPart`
Identifies the part of a clock control that contains the minute or the month. The Mac OS 8.5 Control Manager defines this new control part code constant.

Available in Mac OS X v10.0 and later.

Declared in `ControlDefinitions.h`.

`kControlClockSecondYearPart`
Identifies the part of a clock control that contains the second or the year. The Mac OS 8.5 Control Manager defines this new control part code constant.

Available in Mac OS X v10.0 and later.

Declared in `ControlDefinitions.h`.

`kControlClockAMPMPart`
Identifies the part of a clock control that contains the AM/PM information. The Mac OS 8.5 Control Manager defines this new control part code constant.

Available in Mac OS X v10.0 and later.

Declared in `ControlDefinitions.h`.

Discussion
Constants of type `ControlPartCode` identify specific parts of controls.

Part codes are meaningful only within the scope of a single control definition function. For example, the standard tab control uses part codes 1...N, where N is the number of tabs, even though those numbers do collide with part codes defined for use with other control definition functions. Therefore, when you wish to specify part codes for the tab control for use with the function `SetControlData`, for example, you should use a part code corresponding to a 1-based index of the tab whose data you wish to set. In other words, the first tab is part code 1, the second tab is part code 2, and so on.

Control State Part Code Constants

```
enum {
    kControlNoPart = 0,
    kControlIndicatorPart = 129,
    kControlDisabledPart = 254,
    kControlInactivePart = 255
};
```

Constants

`kControlNoPart`

Identifies no specific control part. This value unhighlights any highlighted part of the control when passed to the `HiliteControl` function. For events in bevel buttons with an attached menu, this part code indicates that either the mouse was released outside the bevel button and menu or that the button was disabled.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kControlIndicatorPart`

Identifies the scroll box of a scroll bar control.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kControlDisabledPart`

Used with `HiliteControl` (page 619)to disable the control.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kControlInactivePart`

Used with `HiliteControl` (page 619) to make the control inactive.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

Discussion

Constants of type `ControlPartCode` identify specific parts of controls.

Part codes are meaningful only within the scope of a single control definition function. For example, the standard tab control uses part codes 1...N, where N is the number of tabs, even though those numbers do collide with part codes defined for use with other control definition functions. Therefore, when you wish to specify part codes for the tab control for use with the function `SetControlData`, for example, you should use a part code corresponding to a 1-based index of the tab whose data you wish to set. In other words, the first tab is part code 1, the second tab is part code 2, and so on.

Note that if you wish to create part codes for a custom control definition function, you may assign values anywhere within the ranges 1–128 and 130–253. Note also that the function `FindControl` does not typically return the `kControlDisabledPart` or `kControlInactivePart` part codes and never returns them with standard controls.

Control Variant Constants

```
enum {
    kControlNoVariant = 0,
    kControlUsesOwningWindowsFontVariant = 1 << 3
};
```

Constants

`kControlNoVariant`

Specifies no change to the standard control resource.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kControlUsesOwningWindowsFontVariant`

Specifies that the control use the window font for any control text.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

Discussion

You can use the control variant constants with any of the standard control resource IDs to specify additional features of a control.

Version Notes

The control variant constants are changed with Appearance Manager 1.0 to support the additional control types available with the Appearance Manager.

Control Bevel Button Menu Placement Constants

```
typedef UInt16 ControlBevelButtonMenuPlacement;
enum {
    kControlBevelButtonMenuOnBottom = 0,
    kControlBevelButtonMenuOnRight = (1 << 2)
};
```

Control Bevel Thickness Constants

```
typedef UInt16 ControlBevelThickness;
enum {
    kControlBevelButtonSmallBevel = 0,
    kControlBevelButtonNormalBevel = 1,
    kControlBevelButtonLargeBevel = 2
};
```

Control Clock Type Constants

```
typedef UInt16 ControlClockType;
enum {
    kControlClockTypeHourMinute = 0,
    kControlClockTypeHourMinuteSecond = 1,
    kControlClockTypeMonthDayYear = 2,
    kControlClockTypeMonthYear = 3
};
```

Control Disclosure Triangle Orientation Constants

```
typedef UInt16 ControlDisclosureTriangleOrientation;
enum {
    kControlDisclosureTrianglePointDefault = 0,
    kControlDisclosureTrianglePointRight = 1,
    kControlDisclosureTrianglePointLeft = 2
};
```

Control Notify Constants

```
enum {
    controlNotifyNothing = 'nada',
    controlNotifyClick = 'clik',
    controlNotifyFocus = 'focu',
    controlNotifyKey = 'key '
};
```

Control Push Button Icon Alignment Constants

```
typedef UInt16 ControlPushButtonIconAlignment;
enum {
    kControlPushButtonIconOnLeft = 6,
```

```
    kControlPushButtonIconOnRight = 7
};
```

Control Round Button Size Constants

```
typedef SInt16 ControlRoundButtonSize;
enum {
    kControlRoundButtonNormalSize = kControlSizeNormal,
    kControlRoundButtonLargeSize = kControlSizeLarge
};
```

Control Slider Orientation Constants

```
typedef UInt16 ControlSliderOrientation;
enum {
    kControlSliderPointsDownOrRight = 0,
    kControlSliderPointsUpOrLeft = 1,
    kControlSliderDoesNotPoint = 2
};
```

Control Tab Direction Constants

```
typedef UInt16 ControlTabDirection;
enum {
    kControlTabDirectionNorth = 0,
    kControlTabDirectionSouth = 1,
    kControlTabDirectionEast = 2,
    kControlTabDirectionWest = 3
};
```

Control Tab Size Constants

```
typedef UInt16 ControlTabSize;
enum {
    kControlTabSizeLarge = kControlSizeNormal,
    kControlTabSizeSmall = kControlSizeSmall
};
```

Drag Control Constants

Specify whether the user is dragging an indicator or the whole control.

```
enum {
    kDragControlEntireControl = 0,
    kDragControlIndicator = 1
};
```

Constants

`kDragControlEntireControl`
Dragging the entire control.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Controls.h`.

`kDragControlIndicator`
Dragging the indicator.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `Controls.h`.

Drawing Constants

```
enum {
    kDrawControlEntireControl = 0,
    kDrawControlIndicatorOnly = 129
};
```

Constants

`kDrawControlEntireControl`
Draw the entire control.
Available in Mac OS X v10.0 and later.
Declared in `Controls.h`.

`kDrawControlIndicatorOnly`
Draw the indicator only.
Available in Mac OS X v10.0 and later.
Declared in `Controls.h`.

Discussion
The Control Manager passes one of these drawing constants in the low word of the `param` parameter to specify whether the user is drawing an indicator or the whole control. The high-order word of the `param` parameter may contain undefined data; therefore, evaluate only the low-order word of this parameter.

Editable Text Control Data Tag Constants

```
enum {
    kControlEditTextStyleTag = kControlFontStyleTag,
    kControlEditTextTextTag = 'text',
    kControlEditTextTEHandleTag = 'than',
    kControlEditTextKeyFilterTag = kControlKeyFilterTag,
    kControlEditTextSelectionTag = 'sele',
    kControlEditTextPasswordTag = 'pass'
};
enum {
    kControlEditTextKeyScriptBehaviorTag = 'kscr',
    kControlEditTextLockedTag = 'lock',
    kControlEditTextFixedTextTag = 'ftxt',
    kControlEditTextValidationProcTag = 'vali',
    kControlEditTextInlinePreUpdateProcTag = 'prup',
    kControlEditTextInlinePostUpdateProcTag = 'poup'
};
enum {
    kControlEditTextCFStringTag = 'cfst',
    kControlEditTextPasswordCFStringTag = 'pwcf'
};
```

Constants

`kControlEditTextTextTag`

Gets or sets text in an editable text control.

Data type returned or set: character buffer

Available in Mac OS X v10.0 and later.

Declared in `HITextViews.h`.

`kControlEditTextTEHandleTag`

Gets a handle to a text edit structure.

Data type returned: `TEHandle`

Available in Mac OS X v10.0 and later.

Declared in `ControlDefinitions.h`.

`kControlEditTextSelectionTag`

Gets or sets the selection in an editable text control.

Data type returned or set: `ControlEditTextSelectionRec` structure

Available in Mac OS X v10.0 and later.

Declared in `HITextViews.h`.

`kControlEditTextKeyScriptBehaviorTag`

Gets or sets the kind of behavior to be used in an editable text control with respect to changing and locking the keyboard menu as the field is focused.

Data type retrieved or set: `ControlKeyScriptBehavior`. The default for password fields is `kControlKeyScriptBehaviorPrefersRoman`. The default for non-password fields is `kControlKeyScriptBehaviorAllowAnyScript`. See `ControlEditTextValidationProcPtr` (page 685) for descriptions of possible values.

Available in Appearance 1.1 or Mac OS 8.5 and later.

Available in Mac OS X v10.0 and later.

Declared in `HITextViews.h`.

`kControlEditTextLockedTag`

Gets or sets whether the text in an editable text control is currently editable.

Data type retrieved or set: `Boolean`; if `true`, the text is locked and cannot be edited; if `false`, the text is editable.

Available in Appearance 1.1 or Mac OS 8.5 and later.

Available in Mac OS X v10.0 and later.

Declared in `HITextViews.h`.

`kControlEditTextFixedTextTag`

Gets or sets inline input text in an editable text control, after confirming any text in the active input area with the Text Services Manager function `FixTSMDocument`.

Data type retrieved or set: character buffer

Available in Appearance 1.1 or Mac OS 8.5 and later.

Available in Mac OS X v10.0 and later.

Declared in `HITextViews.h`.

`kControlEditTextValidationProcTag`

Gets or sets a universal procedure pointer to a callback function such as that described in `ControlEditTextValidationProcPtr` (page 685), which can be used to validate editable text after an operation that changes the text, such as inline text entry, a cut, or paste.

Data type retrieved or set: `ControlEditTextValidationUPP`

Available in Appearance 1.1 or Mac OS 8.5 and later.

Available in Mac OS X v10.0 and later.

Declared in `HITextViews.h`.

`kControlEditTextInlinePreUpdateProcTag`

Gets or sets a universal procedure pointer to a Text Services Manager pre-update callback function.

Data type retrieved or set: `TSMTEPreUpdateUPP`

Available in Appearance 1.1 or Mac OS 8.5 and later.

Available in Mac OS X v10.0 and later.

Declared in `ControlDefinitions.h`.

`kControlEditTextInlinePostUpdateProcTag`

Gets or sets a universal procedure pointer to a Text Services Manager post-update callback function.

Data type retrieved or set: `TSMTEPostUpdateUPP`

Available in Appearance 1.1 or Mac OS 8.5 and later.

Available in Mac OS X v10.0 and later.

Declared in `ControlDefinitions.h`.

`kControlEditTextCFStringTag`

Retrieves the contents of the edit text field as a CFString.

Data type retrieved: `CFStringRef`. You must release the string when you no longer need it.

Available in CarbonLib 1.5 and Mac OS X v10.0 and later.

Available in Mac OS X v10.0 and later.

Declared in `HITextViews.h`.

`kControlEditTextPasswordCFStringTag`

Extract the content of the edit text field if it is a password field.

Data type retrieved: `CFStringRef`. You must release the string when you no longer need it.

Available in CarbonLib 1.5 and Mac OS X v10.0 and later.

Available in Mac OS X v10.1 and later.

Declared in `HITextViews.h`.

Discussion

You can use the control data tag constants to set or obtain data that is associated with a control. The control data tag constants are passed in the `inTagName` parameters of `SetControlData` (page 652) and `GetControlData` (page 594) to specify the piece of data in a control that you wish to set or get. You can also pass these constants in the `inTagName` parameter of `GetControlDataSize` (page 596) if you wish to determine the size of variable-length control data (e.g., text in an editable text control). These constants can also be used by custom control definition functions that return the feature bit `kControlSupportsDataAccess` in response to a `kControlMsgGetFeatures` message.

The data that your application sets or obtains can be of various types, dependent upon the control. Therefore, the descriptions of the control data tag constants list the data types for the information that you can set in the `inData` parameter to the `SetControlData` function and that you can get in the `inBuffer` parameter to the `GetControlData` function.

Version Notes

The control data tag constants are available with Appearance Manager 1.0 and later.

Editable Text Control Definition ID Constants

```
enum {
    kControlEditTextProc = 272,
    kControlEditTextPasswordProc = 274
};
```

Constants

`kControlEditTextProc`

Resource ID: 17

Editable text field for windows. This control maintains its own text handle (`TEHandle`).

This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `ControlDefinitions.h`.

`kControlEditTextPasswordProc`

Resource ID: 17

Editable text field for passwords. This control is supported by the Script Manager. Password text can be accessed via the `kEditTextPasswordTag` constant; see "Editable Text Control Data Tag Constants" (page 756).

This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `ControlDefinitions.h`.

Discussion
When creating a control, your application supplies a control definition ID to one of the Control Manager control-creation functions or to the control resource; see `'CNTL'`. The control definition ID indicates the type of control to create. A control definition ID is an integer that contains the resource ID of a control definition function in its upper 12 bits and a variation code in its lower 4 bits. A control definition ID is derived as follows:

control definition ID = 16 * (`'CDEF'` resource ID) + variation code

A control definition function determines how a control generally looks and behaves. Control definition functions are stored as resources of type `'CDEF'`. Various Control Manager functions call a control definition function whenever they need to perform some control-dependent action, such as drawing the control on the screen. For more information on how to create a control definition function, see `ControlDefProcPtr` (page 677).

A control definition function, in turn, can use a variation code to describe variations of the same basic control. For example, all pop-up arrows share the same basic control definition function, which is stored in a resource of type `'CDEF'` and has a resource ID of 12. The standard pop-up arrow is large and points to the right; it has a control definition ID of 192. A variation of this is a large, left-pointing arrow, which has a control definition ID of 193. Still another variation, in which the arrow points up, has a control definition ID of 194.

Your application can use the constants listed here in place of control definition IDs.

Data Browser Error Constants

```
enum {
    errDataBrowserNotConfigured = -4970,
    errDataBrowserItemNotFound = -4971,
    errDataBrowserItemNotAdded = -4975,
    errDataBrowserPropertyNotFound = -4972,
    errDataBrowserInvalidPropertyPart = -4973,
    errDataBrowserInvalidPropertyData = -4974,
    errDataBrowserPropertyNotSupported = -4979
};
```

Group Box Control Data Tag Constants

```
enum {
    kControlGroupBoxMenuHandleTag = 'mhan',
    kControlGroupBoxMenuRefTag = 'mhan',
    kControlGroupBoxFontStyleTag = kControlFontStyleTag,
    kControlGroupBoxFrameRectTag = 'frec'
};
```

Constants
`kControlGroupBoxMenuHandleTag`
Gets the menu handle of a group box.
Data type returned: `MenuHandle`
Available in Mac OS X v10.0 and later.
Declared in `HIContainerViews.h`.

`kControlGroupBoxFrameRectTag`

Gets the full rectangle that content is drawn in. This is slightly different from the content region because the full rectangle includes the frame drawn around the content. Available in Mac OS X v10.3 and later.

Data type returned: `Rect`

Available in Mac OS X v10.3 and later.

Declared in `HIContainerViews.h`.

Discussion

You can use the control data tag constants to set or obtain data that is associated with a control. The control data tag constants are passed in the `inTagName` parameters of `SetControlData` (page 652) and `GetControlData` (page 594) to specify the piece of data in a control that you wish to set or get. You can also pass these constants in the `inTagName` parameter of `GetControlDataSize` (page 596) if you wish to determine the size of variable-length control data. These constants can also be used by custom control definition functions that return the feature bit `kControlSupportsDataAccess` in response to a `kControlMsgGetFeatures` message.

The data that your application sets or obtains can be of various types, dependent upon the control. Therefore, the descriptions of the control data tag constants list the data types for the information that you can set in the `inData` parameter to the `SetControlData` function and that you can get in the `inBuffer` parameter to the `GetControlData` function.

Version Notes

The control data tag constants are available with Appearance Manager 1.0 and later.

Group Box Control Definition ID Constants

```
enum {
    kControlGroupBoxTextTitleProc = 160,
    kControlGroupBoxCheckBoxProc = 161,
    kControlGroupBoxPopupButtonProc = 162,
    kControlGroupBoxSecondaryTextTitleProc = 164,
    kControlGroupBoxSecondaryCheckBoxProc = 165,
    kControlGroupBoxSecondaryPopupButtonProc = 166
};
```

Constants

`kControlGroupBoxTextTitleProc`

Resource ID: 10

Primary group box with text title. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HIContainerViews.h`.

`kControlGroupBoxCheckBoxProc`

Resource ID: 10

Primary group box with checkbox title. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HIContainerViews.h`.

`kControlGroupBoxPopupButtonProc`
Resource ID: 10

Primary group box with pop-up button title. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HIContainerViews.h`.

`kControlGroupBoxSecondaryTextTitleProc`
Resource ID: 10

Secondary group box with text title. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HIContainerViews.h`.

`kControlGroupBoxSecondaryCheckBoxProc`
Resource ID: 10

Secondary group box with checkbox title. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HIContainerViews.h`.

`kControlGroupBoxSecondaryPopupButtonProc`
Resource ID: 10

Secondary group box with pop-up button title. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HIContainerViews.h`.

Discussion
When creating a control, your application supplies a control definition ID to one of the Control Manager control-creation functions or to the control resource; see `'CNTL'`. The control definition ID indicates the type of control to create. A control definition ID is an integer that contains the resource ID of a control definition function in its upper 12 bits and a variation code in its lower 4 bits. A control definition ID is derived as follows:

control definition ID = 16 * (`'CDEF'` resource ID) + variation code

A control definition function determines how a control generally looks and behaves. Control definition functions are stored as resources of type `'CDEF'`. Various Control Manager functions call a control definition function whenever they need to perform some control-dependent action, such as drawing the control on the screen. For more information on how to create a control definition function, see `ControlDefProcPtr` (page 677).

A control definition function, in turn, can use a variation code to describe variations of the same basic control. For example, all pop-up arrows share the same basic control definition function, which is stored in a resource of type `'CDEF'` and has a resource ID of 12. The standard pop-up arrow is large and points to the right; it has a control definition ID of 192. A variation of this is a large, left-pointing arrow, which has a control definition ID of 193. Still another variation, in which the arrow points up, has a control definition ID of 194.

Your application can use the constants listed here in place of control definition IDs.

Icon Control Data Tag Constants

```
enum {
    kControlIconTransformTag = 'trfm',
    kControlIconAlignmentTag = 'algn'
};
```

Constants

`kControlIconTransformTag`

Gets or sets a transform that is added to the standard transform of an icon see "Icon Utilities."

Data type returned or set: `IconTransformType`

Available in Mac OS X v10.0 and later.

Declared in `HIImageViews.h`.

`kControlIconAlignmentTag`

Gets or sets an icon's position (centered, left, right).

Data type returned or set: `IconAlignmentType`

Available in Mac OS X v10.0 and later.

Declared in `HIImageViews.h`.

Discussion

You can use the control data tag constants to set or obtain data that is associated with a control. The control data tag constants are passed in the `inTagName` parameters of `SetControlData` (page 652) and `GetControlData` (page 594) to specify the piece of data in a control that you wish to set or get. You can also pass these constants in the `inTagName` parameter of `GetControlDataSize` (page 596) if you wish to determine the size of variable-length control data. These constants can also be used by custom control definition functions that return the feature bit `kControlSupportsDataAccess` in response to a `kControlMsgGetFeatures` message.

The data that your application sets or obtains can be of various types, dependent upon the control. Therefore, the descriptions of the control data tag constants list the data types for the information that you can set in the `inData` parameter to the `SetControlData` function and that you can get in the `inBuffer` parameter to the `GetControlData` function.

Version Notes

The control data tag constants are available with Appearance Manager 1.0 and later.

Icon Control Definition ID Constants

```
enum {
    kControlIconProc = 320,
    kControlIconNoTrackProc = 321,
    kControlIconSuiteProc = 322,
    kControlIconSuiteNoTrackProc = 323
};
enum {
    kControlIconRefProc = 324,
    kControlIconRefNoTrackProc = 325
};
```

Constants

`kControlIconProc`
Resource ID: 20

Icon control. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HIImageViews.h`.

`kControlIconNoTrackProc`
Resource ID: 20

Non-tracking icon. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HIImageViews.h`.

`kControlIconSuiteProc`
Resource ID: 20

Icon suite. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HIImageViews.h`.

`kControlIconSuiteNoTrackProc`
Resource ID: 20

Non-tracking icon suite. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HIImageViews.h`.

`kControlIconRefProc`

Identifies the variant of the icon control (`'CDEF'` resource ID 20) that supports all standard types of icon-based content. Note that you do not supply content for this control upon its creation with a call to the `NewControl` function. Rather, after the control's creation you can set or change its content at any time by passing the `SetControlData` function the `kControlIconContentTag` control data tag constant and a `ControlButtonContentInfo` structure containing any of the allowable data types. Supported data types for this icon control variant are specified with the following `ControlContentType` values: `kControlContentIconSuiteRes`, `kControlContentCIconRes` (uses a black-and-white `'ICON'` resource if the color resource isn't available), `kControlContentIconSuiteHandle`, `kControlContentCIconHandle`, and `kControlContentIconRef`. Note, too, that if you supply the `kControlContentIconRef` value, you must first use Icon Services functions to register your resources and generate `IconRef` values. See the `ControlButtonContentInfo` (page 699) data type and "Control Content Type Constants" (page 770) for more information.

Declared in `HIImageViews.h`.

Available in Mac OS 8.5 and later.

`kControlIconRefNoTrackProc`

Identifies the non-tracking variant of the icon control (`'CDEF'` resource ID 20) that supports all standard types of icon-based content. This control immediately returns `kControlIconPart` as the part code hit without tracking. Note that you do not supply content for this control upon its creation with a call to the `NewControl` function. Rather, after the control's creation you can set or change its content at any time by passing the `SetControlData` function the `kControlIconContentTag` control data tag constant and a `ControlButtonContentInfo` structure containing any of the allowable data types. Supported data types for this icon control variant are specified with the following `ControlContentType` values: `kControlContentIconSuiteRes`, `kControlContentCIconRes` (uses a black-and-white `'ICON'` resource if the color resource isn't available), `kControlContentIconSuiteHandle`, `kControlContentCIconHandle`, and `kControlContentIconRef`. Note, too, that if you supply the `kControlContentIconRef` value, you must first use Icon Services functions to register your resources and generate `IconRef` values. See the `ControlButtonContentInfo` (page 699) data type and "Control Content Type Constants" (page 770) for more information.

Declared in `HIImageViews.h`.

Available in Mac OS 8.5 and later.

Discussion

When creating a control, your application supplies a control definition ID to one of the Control Manager control-creation functions or to the control resource; see `'CNTL'`. The control definition ID indicates the type of control to create. A control definition ID is an integer that contains the resource ID of a control definition function in its upper 12 bits and a variation code in its lower 4 bits. A control definition ID is derived as follows:

control definition ID = 16 * (`'CDEF'` resource ID) + variation code

A control definition function determines how a control generally looks and behaves. Control definition functions are stored as resources of type `'CDEF'`. Various Control Manager functions call a control definition function whenever they need to perform some control-dependent action, such as drawing the control on the screen. For more information on how to create a control definition function, see `ControlDefProcPtr` (page 677).

A control definition function, in turn, can use a variation code to describe variations of the same basic control. For example, all pop-up arrows share the same basic control definition function, which is stored in a resource of type 'CDEF' and has a resource ID of 12. The standard pop-up arrow is large and points to the right; it has a control definition ID of 192. A variation of this is a large, left-pointing arrow, which has a control definition ID of 193. Still another variation, in which the arrow points up, has a control definition ID of 194.

Your application can use the constants listed here in place of control definition IDs.

Image Well Control Data Tag Constants

```
enum {
    kControlImageWellContentTag = 'cont',
    kControlImageWellTransformTag = 'tran',
    kControlImageWellIsDragDestinationTag = 'drag'
};
```

Constants

`kControlImageWellContentTag`

Gets or sets the content for an image well; see `ControlButtonContentInfo` (page 699).

Data type returned or set: `ControlButtonContentInfo` structure

Available in Mac OS X v10.0 and later.

Declared in `HIImageViews.h`.

`kControlImageWellTransformTag`

Gets or sets a transform that is added to the standard transform of an image well.

Data type returned or set: `IconTransformType`

Available in Mac OS X v10.0 and later.

Declared in `HIImageViews.h`.

Discussion

You can use the control data tag constants to set or obtain data that is associated with a control. The control data tag constants are passed in the `inTagName` parameters of `SetControlData` (page 652) and `GetControlData` (page 594) to specify the piece of data in a control that you wish to set or get. You can also pass these constants in the `inTagName` parameter of `GetControlDataSize` (page 596) if you wish to determine the size of variable-length control data. These constants can also be used by custom control definition functions that return the feature bit `kControlSupportsDataAccess` in response to a `kControlMsgGetFeatures` message.

The data that your application sets or obtains can be of various types, dependent upon the control. Therefore, the descriptions of the control data tag constants list the data types for the information that you can set in the `inData` parameter to the `SetControlData` function and that you can get in the `inBuffer` parameter to the `GetControlData` function.

Version Notes

The control data tag constants are available with Appearance Manager 1.0 and later.

Image Well Control Definition ID

```
enum {
    kControlImageWellProc = 176
};
```

Constants

`kControlImageWellProc`

Resource ID: 11

Image well. This control behaves as a palette-type object: it can be selected by clicking, and clicking on another object should change the keyboard focus. If the keyboard focus is removed, your application should then set the value to 0 to remove the checked border.

This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HIImageViews.h`.

Discussion

When creating a control, your application supplies a control definition ID to one of the Control Manager control-creation functions or to the control resource; see `'CNTL'`. The control definition ID indicates the type of control to create. A control definition ID is an integer that contains the resource ID of a control definition function in its upper 12 bits and a variation code in its lower 4 bits. A control definition ID is derived as follows:

control definition ID = 16 * (`'CDEF'` resource ID) + variation code

A control definition function determines how a control generally looks and behaves. Control definition functions are stored as resources of type `'CDEF'`. Various Control Manager functions call a control definition function whenever they need to perform some control-dependent action, such as drawing the control on the screen. For more information on how to create a control definition function, see `ControlDefProcPtr` (page 677).

A control definition function, in turn, can use a variation code to describe variations of the same basic control. For example, all pop-up arrows share the same basic control definition function, which is stored in a resource of type `'CDEF'` and has a resource ID of 12. The standard pop-up arrow is large and points to the right; it has a control definition ID of 192. A variation of this is a large, left-pointing arrow, which has a control definition ID of 193. Still another variation, in which the arrow points up, has a control definition ID of 194.

Your application can use the constant listed here in place of a control definition ID.

inLabel

```
enum {
    inLabel = kControlLabelPart,
    inMenu = kControlMenuPart,
    inTriangle = kControlTrianglePart,
    inButton = kControlButtonPart,
    inCheckBox = kControlCheckBoxPart,
    inUpButton = kControlUpButtonPart,
    inDownButton = kControlDownButtonPart,
    inPageUp = kControlPageUpPart,
    inPageDown = kControlPageDownPart
};
```

inThumb

```
enum {
    inThumb = kControlIndicatorPart,
    kNoHiliteControlPart = kControlNoPart,
    kInIndicatorControlPart = kControlIndicatorPart,
    kReservedControlPart = kControlDisabledPart,
    kControlInactiveControlPart = kControlInactivePart
};
```

kControlBevelButtonOwnedMenuRefTag

```
enum {
    kControlBevelButtonOwnedMenuRefTag = 'omrf',
    kControlBevelButtonKindTag = 'bebk'
};
```

Bevel Button Size Constants

```
enum {
    kControlBevelButtonSmallBevelVariant = 0,
    kControlBevelButtonNormalBevelVariant = (1 << 0),
    kControlBevelButtonLargeBevelVariant = (1 << 1),
    kControlBevelButtonMenuOnRightVariant = (1 << 2)
};
```

Control Can Auto Invalidate Constant

```
enum {
    kControlCanAutoInvalidate = 1
};
```

Control Chasing Arrows Animating Tag Constant

```
enum {
    kControlChasingArrowsAnimatingTag = 'anim'
```

```
};
```

Control Collection Tag Constants

Specify initial control values passed in a collection.

```
enum {
    kControlCollectionTagBounds = 'boun',
    kControlCollectionTagValue = 'valu',
    kControlCollectionTagMinimum = 'min ',
    kControlCollectionTagMaximum = 'max ',
    kControlCollectionTagViewSize = 'view',
    kControlCollectionTagVisibility = 'visi',
    kControlCollectionTagRefCon = 'refc',
    kControlCollectionTagTitle = 'titl',
    kControlCollectionTagUnicodeTitle = 'uttl',
    kControlCollectionTagIDSignature = 'idsi',
    kControlCollectionTagIDID = 'idid',
    kControlCollectionTagCommand = 'cmd ',
    kControlCollectionTagVarCode = 'varc'
};
```

Constants

`kControlCollectionTagBounds`
A value of type `Rect` that contains the bounding rectangle of the control.
Available in Mac OS X v10.0 and later.
Declared in `Controls.h`.

`kControlCollectionTagValue`
A value of type `SInt32` that contains the value of the control.
Available in Mac OS X v10.0 and later.
Declared in `Controls.h`.

`kControlCollectionTagMinimum`
A value of type `SInt32` that contains the minimum value of the control.
Available in Mac OS X v10.0 and later.
Declared in `Controls.h`.

`kControlCollectionTagMaximum`
A value of type `SInt32` that contains the maximum value of the control.
Available in Mac OS X v10.0 and later.
Declared in `Controls.h`.

`kControlCollectionTagViewSize`
A value of type `SInt32` that contains the view size of the control.
Available in Mac OS X v10.0 and later.
Declared in `Controls.h`.

`kControlCollectionTagVisibility`
A Boolean that contains the visible state of the control. This tag is only interpreted on CarbonLib versions through 1.5.x and Mac OS X v10.0.x. This tag is not interpreted on CarbonLib 1.6 and later nor on Mac OS X v10.1 and later; use of this tag is not recommended.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kControlCollectionTagRefCon`
A value of type `SInt32` that contains the refcon for the control.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kControlCollectionTagTitle`
A character array of arbitrary size that contains the title of the control.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kControlCollectionTagUnicodeTitle`
A character array of arbitrary size that contains the title of the control as received via `CFStringCreateExternalRepresentation`.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kControlCollectionTagIDSignature`
An OSType that contains the ControlID signature for the control.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kControlCollectionTagIDID`
A value of type `SInt32` that contains the ControlID ID for the control.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kControlCollectionTagCommand`
A value of type `UInt32` that contains the command.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kControlCollectionTagVarCode`
A value of type `SInt16` that contains the variant for the control.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

Discussion
These standard tags are used in the initial data collection that is passed in the `param` parameter to the `initCntl` message and in the `kEventParamInitCollection` parameter to the `kEventControlInitialize` event (Carbon only).

All tags at ID 0 in a control's collection are reserved for Control Manager use. Custom control definitions should use other IDs.

Most of these tags are interpreted when you call `CreateCustomControl` (page 543). The Control Manager puts the value in the right place before it calls the control definition with the initialization message.

Control Collection Tag Subcontrols Constant

```
enum {
    kControlCollectionTagSubControls = 'subc'
};
```

Control Content Type Constants

```
enum {
    kControlContentTextOnly = 0,
    kControlNoContent = 0,
    kControlContentIconSuiteRes = 1,
    kControlContentCIconRes = 2,
    kControlContentPictRes = 3,
    kControlContentICONRes = 4,
    kControlContentIconSuiteHandle = 129,
    kControlContentCIconHandle = 130,
    kControlContentPictHandle = 131,
    kControlContentIconRef = 132,
    kControlContentICON = 133
};
```

Constants

`kControlContentTextOnly`
Content type is text. This constant is passed in the `contentType` field of the `ControlButtonContentInfo` structure if the content is text only. The variation code `kControlUsesOwningWindowsFontVariant` applies when text content is used.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kControlContentIconSuiteRes`
Content type uses an icon suite resource ID. The resource ID of the icon suite resource you wish to display should be in the `resID` field of the `ControlButtonContentInfo` structure.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kControlContentCIconRes`
Content type is a color icon resource ID. The resource ID of the color icon resource you wish to display should be in the `resID` field of the `ControlButtonContentInfo` structure.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kControlContentPictRes`
Content type is a picture resource ID. The resource ID of the picture resource you wish to display should be in the `resID` field of the `ControlButtonContentInfo` structure.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kControlContentIconSuiteHandle`
Content type is an icon suite handle. The handle of the icon suite you wish to display should be in the `iconSuite` field of the `ControlButtonContentInfo` structure.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kControlContentCIconHandle`
Content type uses a color icon handle. The handle of the color icon you wish to display should be in the `cIconHandle` field of the `ControlButtonContentInfo` structure.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kControlContentPictHandle`
Content type uses a picture handle. The handle of the picture you wish to display should be in the `picture` field of the `ControlButtonContentInfo` structure.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kControlContentIconRef`
Content type is `IconRef`. An `IconRef` value for the icon you wish to display should be provided in the `iconRef` field of the `ControlButtonContentInfo` structure. Note that the `kControlBevelButtonGraphicOffsetTag` control data tag constant should not be used with `IconRef` based bevel button content, because `IconRef` based icons may change with a theme switch; see "Bevel Button Control Data Tag Constants" (page 723). Supported with Mac OS 8.5 and later.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

Control Data Browser Tag Constants

```
enum {
    kControlDataBrowserIncludesFrameAndFocusTag = 'brdr',
    kControlDataBrowserKeyFilterTag = kControlEditTextKeyFilterTag,
    kControlDataBrowserEditTextKeyFilterTag = kControlDataBrowserKeyFilterTag,
   kControlDataBrowserEditTextValidationProcTag =  kControlEditTextValidationProcTag
};
```

Control Def Constants

```
enum {
    kControlDefProcPtr = 0,
    kControlDefObjectClass = 1
};
```

Constants

`kControlDefProcPtr`
Indicates raw, proc-ptr based access.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Controls.h`.

`kControlDefObjectClass`
Indicates event-based definition (Mac OS X only).

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Controls.h`.

Control Def Type Constants

```
enum {
    kControlDefProcType = 'CDEF',
    kControlTemplateResourceType = 'CNTL',
    kControlColorTableResourceType = 'cctb',
    kControlDefProcResourceType = 'CDEF'
};
```

Disclosure Triangle Constants

```
enum {
    kControlDisclosureButtonClosed = 0,
    kControlDisclosureButtonDisclosed = 1
};
```

Constants

`kControlDisclosureButtonClosed`
The control will be drawn suggesting a closed state.
Available in Mac OS X v10.0 and later.
Declared in `HIDisclosureViews.h`.

`kControlDisclosureButtonDisclosed`
The control will be drawn suggesting a disclosed state.
Available in Mac OS X v10.0 and later.
Declared in `HIDisclosureViews.h`.

Unicode Control Data Tags

Specify data tags used with Unicode edit text controls.

```
enum {
  kControlEditTextSingleLineTag = 'sglc',
  kControlEditTextInsertTextBufferTag = 'intx',
  kControlEditTextInsertCFStringRefTag = 'incf',
  kControlEditUnicodeTextPostUpdateProcTag = 'upup'
};
```

Constants

`kControlEditTextSingleLineTag`
Indicates whether the control should always be single line.
Data to set or get is type `Boolean`.
Available in Mac OS X v10.2 and later.
Declared in `HITextViews.h`.

`kControlEditTextInsertTextBufferTag`
Gets or sets the control's text as WorldScript encoded text. Available in Mac OS X v10.3 and later.
Data to get or set is an array of `char` types.
Available in Mac OS X v10.3 and later.
Declared in `HITextViews.h`.

`kControlEditTextInsertCFStringRefTag`

Gets or sets the control's text as a CFString. Available in Mac OS X v10.3 and later.

Data to get or set is type `CFStringRef`. If obtaining the string, be sure to release it after you are done with it.

Available in Mac OS X v10.3 and later.

Declared in `HITextViews.h`.

`kControlEditUnicodeTextPostUpdateProcTag`

Gets or sets the post-update callback function.

Data to get or set is type `UnicodePostUpdateUPP`.

Available in Mac OS X v10.0 and later.

Declared in `HITextViews.h`.

Discussion

You use these tags in conjunction with `GetControlData` (page 594) and `SetControlData` (page 652) on Unicode edit text controls.

Control Edit Unicode Text Post Update Proc Tag Constant

```
enum {
    kControlEditUnicodeTextPostUpdateProcTag = 'upup'
};
```

Control Edit Unicode Text Proc Constants

```
enum {
    kControlEditUnicodeTextProc = 912,
    kControlEditUnicodeTextPasswordProc = 914
};
```

Control Entire Control Constant

```
enum {
    kControlEntireControl = 0
};
```

Control Kind Bevel Button Constant

```
enum {
    kControlKindBevelButton = 'bevl'
};
```

Control Kind Chasing Arrows Constant

```
enum {
    kControlKindChasingArrows = 'carr'
};
```

Control Kind Clock Constant

```
enum {
    kControlKindClock = 'clck'
};
```

Control Kind Data Browser Constant

```
enum {
    kControlKindDataBrowser = 'datb'
};
```

Control Kind Disclosure Button Constant

```
enum {
```

```
    kControlKindDisclosureButton = 'disb'
};
```

Control Kind Disclosure Triangle Constant

```
enum {
    kControlKindDisclosureTriangle = 'dist'
};
```

Control Kind Edit Text Constant

```
enum {
    kControlKindEditText = 'etxt'
};
```

Control Kind Edit Unicode Text Constant

```
enum {
    kControlKindEditUnicodeText = 'eutx'
};
```

Control Kind Group Box Constants

```
enum {
    kControlKindGroupBox = 'grpb',
    kControlKindCheckGroupBox = 'cgrp',
    kControlKindPopupGroupBox = 'pgrp'
};
```

Control Kind Icon Constant

```
enum {
    kControlKindIcon = 'icon'
};
```

Control Kind Image Well Constant

```
enum {
    kControlKindImageWell = 'well'
};
```

Control Kind List Box Constant

```
enum {
    kControlKindListBox = 'lbox'
};
```

kControlKindLittleArrows

```
enum {
    kControlKindLittleArrows = 'larr'
};
```

Control Kind Picture Constant

```
enum {
    kControlKindPicture = 'pict'
};
```

Control Kind Placard Constant

```
enum {
    kControlKindPlacard = 'plac'
};
```

Control Kind Pop-up Arrow Constant

```
enum {
    kControlKindPopupArrow = 'parr'
};
```

Control Kind Pop-up Button Constant

```
enum {
    kControlKindPopupButton = 'popb'
};
```

Control Kind Progress Bar Constants

```
enum {
    kControlKindProgressBar = 'prgb',
    kControlKindRelevanceBar = 'relb'
};
```

Control Kind Push and Radio Button Constants

```
enum {
    kControlKindPushButton = 'push',
    kControlKindPushIconButton = 'picn',
    kControlKindRadioButton = 'rdio',
    kControlKindCheckBox = 'cbox'
};
```

Control Kind Radio Group Constant

```
enum {
    kControlKindRadioGroup = 'rgrp'
};
```

Control Kind Round Button Constant

```
enum {
    kControlKindRoundButton = 'rndb'
};
```

Control Kind Scroll Bar Constant

```
enum {
    kControlKindScrollBar = 'sbar'
};
```

Control Kind Scrolling Text Box Constant

```
enum {
    kControlKindScrollingTextBox = 'stbx'
};
```

Control Kind Separator Constant

```
enum {
    kControlKindSeparator = 'sepa'
};
```

Control Kind Signature Apple Constant

```
enum {
    kControlKindSignatureApple = 'appl'
};
```

Constants

`kControlKindSignatureApple`
Signature of all system controls.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

Control Kind Slider Constant

```
enum {
    kControlKindSlider = 'sldr'
};
```

Control Kind Static Text Constant

```
enum {
    kControlKindStaticText = 'stxt'
};
```

Control Kind Tabs Constant

```
enum {
    kControlKindTabs = 'tabs'
};
```

Control Kind User Pane Constant

```
enum {
    kControlKindUserPane = 'upan'
};
```

Control Kind Window Header Constant

```
enum {
    kControlKindWindowHeader = 'whed'
};
```

Control Picture Handle Tag Constant

```
enum {
    kControlPictureHandleTag = 'pich'
};
```

Control Pop-up Arrow Orientation Constants

```
enum {
    kControlPopupArrowOrientationEast = 0,
    kControlPopupArrowOrientationWest = 1,
    kControlPopupArrowOrientationNorth = 2,
    kControlPopupArrowOrientationSouth = 3
};
```

Control Pop-up Arrow Size Constants

```
enum {
    kControlPopupArrowSizeNormal = 0,
    kControlPopupArrowSizeSmall = 1
};
```

Control Pop-up Button Check Current Tag Constant

```
enum {
    kControlPopupButtonCheckCurrentTag = 'chck'
};
```

Control Property Persistent Constant

```
enum {
    kControlPropertyPersistent = 0x00000001
};
```

Control Round Button Content and Size Tag Constants

```
enum {
    kControlRoundButtonContentTag = 'cont',
    kControlRoundButtonSizeTag = kControlSizeTag
};
```

Control Scrollbar Shows Arrows Tag Constant

```
enum {
    kControlScrollBarShowsArrowsTag = 'arro'
};
```

Control Size Constants

```
enum {
    kControlSizeNormal = 0,
    kControlSizeSmall = 1,
    kControlSizeLarge = 2,
    kControlSizeAuto = 0xFFFF
};
```

Control Supports New Messages Constant

```
enum {
    kControlSupportsNewMessages = ' ok '
};
```

Constants

`kControlSupportsNewMessages`
The control definition function supports new messages introduced with Mac OS 8 and the Appearance Manager.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `Controls.h`.

Discussion
If your control definition function supports Appearance-compliant messages, it should return `kControlSupportsNewMessages` as a function result when the Control Manager passes `kControlMsgTestNewMsgSupport` in the `message` parameter.

Control Tab Image Content Tag Constant

```
enum {
    kControlTabImageContentTag = 'cont'
};
```

Control Tab Info Version Constants

```
enum {
    kControlTabInfoVersionZero = 0,
    kControlTabInfoVersionOne = 1
};
```

Control Tab Type Constants

```
enum {
    kControlTabListResType = 'tab#',
    kControlListDescResType = 'ldes'
};
```

Control Use Theme Font ID Mask Constant

```
enum {
    kControlUseThemeFontIDMask = 0x0080
};
```

Click Activation Constants

Specify constants that indicate the way a control prefers to respond to a click.

```
enum {
    kDoNotActivateAndIgnoreClick = 0,
    kDoNotActivateAndHandleClick = 1,
    kActivateAndIgnoreClick = 2,
    kActivateAndHandleClick = 3
};
typedef UInt32 ClickActivationResult;
```

Constants

`kDoNotActivateAndIgnoreClick`
Indicates that the click should be ignored and that the window should not be activated. This constant is defined for completeness and is rarely used.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kDoNotActivateAndHandleClick`
Indicates that the control should handle the click while the window is still in the background.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kActivateAndIgnoreClick`
Indicates that control doesn't want to respond directly to the click, but window should still be brought forward.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kActivateAndHandleClick`
Indicates that the control wants to respond to the click, but only after the window has been activated.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

Discussion

These constants are used by `GetControlClickActivation` (page 593).

Selection Constants

```
enum {
    kDragSelect = 1,
    kSelectOnlyOne = 2,
    kResetSelection = 4,
    kCmdTogglesSelection = 8,
    kNoDisjointSelection = 16,
    kAlwaysExtendSelection = 32
};
```

Drag Tracking Enter Control Constants

```
enum {
    kDragTrackingEnterControl = 2,
    kDragTrackingInControl = 3,
    kDragTrackingLeaveControl = 4
};
```

Key Filter Result Codes

```
enum {
    kControlKeyFilterBlockKey = 0,
    kControlKeyFilterPassKey = 1
};
```

Constants

`kControlKeyFilterBlockKey`

The keystroke is blocked and not received by the control.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kControlKeyFilterPassKey`

The keystroke is filtered and received by the control.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

Discussion

Your key filter function returns these constants to specify whether or not a keystroke is filtered or blocked.

In Control Part Constants

```
enum {
    kInLabelControlPart = kControlLabelPart,
    kInMenuControlPart = kControlMenuPart,
    kInTriangleControlPart = kControlTrianglePart,
    kInButtonControlPart = kControlButtonPart,
    kInCheckBoxControlPart = kControlCheckBoxPart,
    kInUpButtonControlPart = kControlUpButtonPart,
    kInDownButtonControlPart = kControlDownButtonPart,
    kInPageUpControlPart = kControlPageUpPart,
    kInPageDownControlPart = kControlPageDownPart
};
```

Order Constants

```
enum {
    kOrderUndefined = 0,
    kOrderIncreasing = 1,
    kOrderDecreasing = 2
};
```

List Box Control Data Tag Constants

```
enum {
    kControlListBoxListHandleTag = 'lhan',
    kControlListBoxKeyFilterTag = kControlKeyFilterTag,
    kControlListBoxFontStyleTag = kControlFontStyleTag
};
enum {
    kControlListBoxDoubleClickTag = 'dblc',
    kControlListBoxLDEFTag = 'ldef'
};
```

Constants

`kControlListBoxListHandleTag`

Gets a handle to a list box.

Data type returned: `ListHandle`

Available in Mac OS X v10.0 and later.

Declared in `ControlDefinitions.h`.

`kControlListBoxDoubleClickTag`

Checks to see whether the most recent click in a list box was a double click. Available with Appearance 1.0.1 and later.

Data type returned: `Boolean`; if `true`, the last click was a double click; if `false`, not.

Available in Mac OS X v10.0 and later.

Declared in `ControlDefinitions.h`.

`kControlListBoxLDEFTag`
Sets the `'LDEF'` resource to be used to draw a list box's contents this is useful for creating a list box without an `'ldes'` resource. Available with Appearance 1.0.1 and later.

Data type set: `SInt16`

Available in Mac OS X v10.0 and later.

Declared in `ControlDefinitions.h`.

Discussion
You can use the control data tag constants to set or obtain data that is associated with a control. The control data tag constants are passed in the `inTagName` parameters of `SetControlData` (page 652) and `GetControlData` (page 594) to specify the piece of data in a control that you wish to set or get. You can also pass these constants in the `inTagName` parameter of `GetControlDataSize` (page 596) if you wish to determine the size of variable-length control data. These constants can also be used by custom control definition functions that return the feature bit `kControlSupportsDataAccess` in response to a `kControlMsgGetFeatures` message.

The data that your application sets or obtains can be of various types, dependent upon the control. Therefore, the descriptions of the control data tag constants list the data types for the information that you can set in the `inData` parameter to the `SetControlData` function and that you can get in the `inBuffer` parameter to the `GetControlData` function.

Version Notes
The control data tag constants are available with Appearance Manager 1.0 and later.

List Box Control Definition ID Constants

```
enum {
    kControlListBoxProc = 352,
    kControlListBoxAutoSizeProc = 353
};
```

Constants
`kControlListBoxProc`
Resource ID: 21

List box. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `ControlDefinitions.h`.

`kControlListBoxAutoSizeProc`
Resource ID: 21

Autosizing list box. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `ControlDefinitions.h`.

Discussion
When creating a control, your application supplies a control definition ID to one of the Control Manager control-creation functions or to the control resource; see `'CNTL'`. The control definition ID indicates the type of control to create. A control definition ID is an integer that contains the resource ID of a control definition function in its upper 12 bits and a variation code in its lower 4 bits. A control definition ID is derived as follows:

control definition ID = 16 * ('CDEF' resource ID) + variation code

A control definition function determines how a control generally looks and behaves. Control definition functions are stored as resources of type 'CDEF'. Various Control Manager functions call a control definition function whenever they need to perform some control-dependent action, such as drawing the control on the screen. For more information on how to create a control definition function, see ControlDefProcPtr (page 677).

A control definition function, in turn, can use a variation code to describe variations of the same basic control. For example, all pop-up arrows share the same basic control definition function, which is stored in a resource of type 'CDEF' and has a resource ID of 12. The standard pop-up arrow is large and points to the right; it has a control definition ID of 192. A variation of this is a large, left-pointing arrow, which has a control definition ID of 193. Still another variation, in which the arrow points up, has a control definition ID of 194.

Your application can use the constants listed here in place of control definition IDs.

Little Arrows Control Definition ID Constant

```
enum {
    kControlLittleArrowsProc = 96
};
```

Constants

kControlLittleArrowsProc

Resource ID: 6

Little arrows. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in HILittleArrows.h.

Discussion

When creating a control, your application supplies a control definition ID to one of the Control Manager control-creation functions or to the control resource; see 'CNTL'. The control definition ID indicates the type of control to create. A control definition ID is an integer that contains the resource ID of a control definition function in its upper 12 bits and a variation code in its lower 4 bits. A control definition ID is derived as follows:

control definition ID = 16 * ('CDEF' resource ID) + variation code

A control definition function determines how a control generally looks and behaves. Control definition functions are stored as resources of type 'CDEF'. Various Control Manager functions call a control definition function whenever they need to perform some control-dependent action, such as drawing the control on the screen. For more information on how to create a control definition function, see ControlDefProcPtr (page 677).

A control definition function, in turn, can use a variation code to describe variations of the same basic control. For example, all pop-up arrows share the same basic control definition function, which is stored in a resource of type 'CDEF' and has a resource ID of 12. The standard pop-up arrow is large and points to the right; it has a control definition ID of 192. A variation of this is a large, left-pointing arrow, which has a control definition ID of 193. Still another variation, in which the arrow points up, has a control definition ID of 194.

Your application can use the constant listed here in place of a control definition ID.

Little Arrows Control Tag Constant

```
enum {
    kControlLittleArrowsIncrementValueTag = 'incr'
};
```

Constants

`kControlLittleArrowsIncrementValueTag`

Gets or sets the increment value of the control. Currently, the little arrows control does not use the increment value because the control does not change the value itself. You must use an action proc to change the value. Available in Mac OS X v10.3 and later.

Data type retrieved: `SInt32`

Available in Mac OS X v10.3 and later.

Declared in `HILittleArrows.h`.

Mac OS 8.5 Bevel Button Control Data Tag Constant

```
enum {
    kControlBevelButtonScaleIconTag = 'scal'
};
```

Constants

`kControlBevelButtonScaleIconTag`

Gets or sets whether, when the proper icon size is unavailable, an icon should be scaled for use with a given bevel button. This tag is only for use with icon suites or the `IconRef` data type.

Data type retrieved or set: `Boolean`. If `true`, indicates that if an icon of the ideal size isn't available, a larger or smaller icon should be scaled to the ideal size. If `false`, no scaling should occur; instead, a smaller icon should be drawn or a larger icon clipped. Default is `false`.

Available in Mac OS X v10.0 and later.

Declared in `HIButtonViews.h`.

Discussion

The Mac OS 8.5 Control Manager defines this new control data tag constant. This constant is passed in the `inTagName` parameters of the functions `SetControlData` and `GetControlData` to specify the piece of data in a control that you wish to set or get. You can also pass this constant in the `inTagName` parameter of the function `GetControlDataSize` if you wish to determine the size of variable-length control data. This constant can also be used by custom control definition functions that return the feature bit `kControlSupportsDataAccess` in response to a `kControlMsgGetFeatures` message.

The data that your application gets or sets can be of various types. The description here shows the data type for the information that you can set in the `inData` parameter to the `SetControlData` function and that you can get in the `inBuffer` parameter to the `GetControlData` function.

Mac OS 8.5 Control Font Style Flag Constant

```
enum {
    kControlAddToMetaFontMask = 0x0200
};
```

Constants

`kControlAddToMetaFontMask`

If the bit specified by this mask is set, the control may use a meta-font while also adding other attributes to the font. If the bit specified by this mask is not set, but a meta-font is specified for the control, any additional attributes set for the font are ignored.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

Discussion

With the Mac OS 8.5 Control Manager, you can pass this new control font style flag constant in the `flags` field of the `ControlFontStyleRec` structure to specify the fields of the structure that should be applied to the control. For more on control font style flag constants, see "Control Font Style Flag Constants" (page 745) and the ControlFontStyleRec (page 704) structure.

Mac OS 8.5 Editable Text Control Definition ID Constant

```
enum {
    kControlEditTextInlineInputProc = 276
};
```

Constants

`kControlEditTextInlineInputProc`

Identifies the inline input variant of the editable text control (`'CDEF'` resource ID 17), which supports 2-byte script systems. This variant cannot be combined with the password variant of the editable text box.

Available in Mac OS X v10.0 and later.

Declared in `ControlDefinitions.h`.

Discussion

The Mac OS 8.5 Control Manager defines this new control definition ID.

When creating a control, your application supplies a control definition ID to one of the Control Manager control-creation functions or to the control resource; see `'CNTL'`. The control definition ID indicates the type of control to create. A control definition ID is an integer that contains the resource ID of a control definition function in its upper 12 bits and a variation code in its lower 4 bits. A control definition ID is derived as follows:

control definition ID = 16 * (`'CDEF'` resource ID) + variation code

A control definition function determines how a control generally looks and behaves. Control definition functions are stored as resources of type `'CDEF'`. Various Control Manager functions call a control definition function whenever they need to perform some control-dependent action, such as drawing the control on the screen. For more information on how to create a control definition function, see `ControlDefProcPtr` (page 677).

A control definition function, in turn, can use a variation code to describe variations of the same basic control. For example, all pop-up arrows share the same basic control definition function, which is stored in a resource of type `'CDEF'` and has a resource ID of 12. The standard pop-up arrow is large and points to the right; it has a control definition ID of 192. A variation of this is a large, left-pointing arrow, which has a control definition ID of 193. Still another variation, in which the arrow points up, has a control definition ID of 194.

Your application can use the constant listed here in place of a control definition ID. This of these constant, and its associated ID, is not supported unless the Appearance Manager is available.

Mac OS 8.5 Group Box Control Data Tag Constant

```
enum {
    kControlGroupBoxTitleRectTag = 'trec'
};
```

Constants

`kControlGroupBoxTitleRectTag`

Gets the rectangle that contains the title of a group box (and any associated control, such as a checkbox or other button).

Data type retrieved: `Rect`

Available in Mac OS X v10.0 and later.

Declared in `HIContainerViews.h`.

Discussion

The Mac OS 8.5 Control Manager defines this new control data tag constant. This constant is passed in the `inTagName` parameters of the functions `SetControlData` and `GetControlData` to specify the piece of data in a control that you wish to set or get. You can also pass this constant in the `inTagName` parameter of the function `GetControlDataSize` if you wish to determine the size of variable-length control data. This constant can also be used by custom control definition functions that return the feature bit `kControlSupportsDataAccess` in response to a `kControlMsgGetFeatures` message.

The data that your application gets or sets can be of various types. The description here shows the data type for the information that you can set in the `inData` parameter to the `SetControlData` function and that you can get in the `inBuffer` parameter to the `GetControlData` function.

Mac OS 8.5 Icon Control Data Tag Constants

```
enum {
    kControlIconResourceIDTag = 'ires',
    kControlIconContentTag = 'cont'
};
```

Constants

`kControlIconResourceIDTag`

Gets or sets the resource ID of the icon to use.

Data type retrieved or set: `SInt16`

Available in Mac OS X v10.0 and later.

Declared in `HIImageViews.h`.

`kControlIconContentTag`

Gets or sets the type of content to be used in an icon control.

Data type retrieved or set: `ControlButtonContentInfo` (page 699).

Available in Mac OS X v10.0 and later.

Declared in `HIImageViews.h`.

Discussion

The Mac OS 8.5 Control Manager defines these new control data tag constants. These constants are passed in the `inTagName` parameters of the functions `SetControlData` and `GetControlData` to specify the piece of data in a control that you wish to set or get. You can also pass these constants in the `inTagName` parameter of the function `GetControlDataSize` if you wish to determine the size of variable-length control data. These constants can also be used by custom control definition functions that return the feature bit `kControlSupportsDataAccess` in response to a `kControlMsgGetFeatures` message.

The data that your application gets or sets can be of various types. The descriptions here show the data types for the information that you can set in the `inData` parameter to the `SetControlData` function and that you can get in the `inBuffer` parameter to the `GetControlData` function.

Mac OS 8.5 Pop-up Button Control Data Tag Constants

```
enum {
    kControlPopupButtonExtraHeightTag = 'exht',
    kControlPopupButtonOwnedMenuRefTag = 'omrf'
};
```

Constants

`kControlPopupButtonExtraHeightTag`

Gets or sets the amount of extra vertical white space in a pop-up menu button.

Data type set or retrieved: `SInt16`; default is 0.

Available in Mac OS X v10.0 and later.

Declared in `HIPopupButton.h`.

`kControlPopUpButtonOwnedMenuRefTag`

Sets the menu to be displayed by the popup button control. This tag operates identically to `kControlPopupButtonMenuRefTag`, except that the popup button takes ownership of the specified menu. If the popup button is disposed, or a new menu is specified, the popup button control will automatically release the menu.

Data type set or retrieved: `MenuRef`

Discussion

The Mac OS 8.5 Control Manager defines this new control data tag constant. This constant is passed in the `inTagName` parameters of the functions `SetControlData` and `GetControlData` to specify the piece of data in a control that you wish to set or get. You can also pass this constant in the `inTagName` parameter of the function `GetControlDataSize` if you wish to determine the size of variable-length control data. This constant can also be used by custom control definition functions that return the feature bit `kControlSupportsDataAccess` in response to a `kControlMsgGetFeatures` message.

The data that your application gets or sets can be of various types. The description here shows the data type for the information that you can set in the `inData` parameter to the `SetControlData` function and that you can get in the `inBuffer` parameter to the `GetControlData` function.

Control Meta Part Code Constants

```
enum {
    kControlStructureMetaPart = -1,
    kControlContentMetaPart = -2,
    kControlOpaqueMetaPart = -3,
    kControlClickableMetaPart = -4
};
```

Constants

`kControlStructureMetaPart`

The entire area that the control will draw into. This area may extend beyond the bounds of the control (for example, if the control draws a focus ring outside of its bounds). You may return a superset of the drawn area if this is computationally easier to construct. This area is used to determine the area of a window that should be invalidated and redrawn when a control is invalidated. It is not necessary for a control to return a shape that precisely describes the structure area; for example, a control whose structure is an oval may simply return the oval's bounding rectangle. The default handler for the `kEventControlGetPartRegion` event returns the control's bounds when this part is requested.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kControlContentMetaPart`

The area of the control in which embedded controls should be positioned. This part is only defined for controls that can contain other controls (for example, the group box). This area is largely informational and is not used by the Control Manager itself. The default handler for the `kEventControlGetPartRegion` event returns `errInvalidPartCode` when this part is requested.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kControlOpaqueMetaPart`

The area of the control that, when drawn, is filled with opaque pixels. You may also return a subset of the opaque area if this is computationally easier to construct. If a control is contained in a composited window, the Control Manager will use this area to optimize drawing of other controls that intersect this area; controls that are entirely contained within the opaque area, and that are z-ordered underneath this control, will not be drawn at all, since any drawing would be completely overwritten by this control. The default handler for the `kEventControlGetPartRegion` event returns an empty area when this part is requested.

Available in Mac OS X v10.3 and later.

Declared in `Controls.h`.

`kControlClickableMetaPart`

The area of the control that causes a mouse event to be captured by that control. If a mouse event falls inside the control bounds but outside of this area, then the Control Manager will allow the event to pass through the control to the next control behind it in z-order. This area is used to determine which parts of a window should allow async window dragging when clicked (the draggable area is computed by subtracting the clickable areas of controls in the window from the window's total area). You can also customize the clickable area of a control if you want the control to have an effectively transparent area (for example, a control that draws multiple tabs might want clicks in the space between the tabs to fall through to the next control rather than be captured by the tab-drawing control). The default handler for the `kEventControlGetPartRegion` event returns the control's bounds when this part is requested.

Available in Mac OS X v10.3 and later.

Declared in `Controls.h`.

Discussion
An application that needs the structure and content regions of a control can call `GetControlRegion` (page 605) and specify these meta-parts. A custom HIView that needs to specialize its opaque and clickable regions can provide a `kEventControlGetPartRegion` event handler that checks for these meta-parts and return an appropriate region (or HIShape).

Meta Font Constants

```
enum {
    kControlFontBigSystemFont = -1,
    kControlFontSmallSystemFont = -2,
    kControlFontSmallBoldSystemFont = -3,
    kControlFontViewSystemFont = -4
};
```

Constants

`kControlFontBigSystemFont`
Use the system font.
Available in Mac OS X v10.0 and later.
Declared in `Controls.h`.

`kControlFontSmallSystemFont`
Use the small system font.
Available in Mac OS X v10.0 and later.
Declared in `Controls.h`.

`kControlFontSmallBoldSystemFont`
Use the small emphasized system font (emphasis applied correctly for locale).
Available in Mac OS X v10.0 and later.
Declared in `Controls.h`.

Discussion
You can use the meta font constants in the `font` field of the structure `ControlFontStyleRec` (page 704) and the Font ID field of a dialog font table resource to specify the style, size, and font family of the control font. You should use these meta font constants whenever possible because the system font can change, depending upon the current theme. If none of these constants are specified, the control uses the system font unless directed to use a window font by a control variant.

Version Notes
The meta font constants are available with Appearance Manager 1.0 and later.

Constraint Constants

```
enum {
    noConstraint = kNoConstraint,
    hAxisOnly = 1,
    vAxisOnly = 2
};
```

Part Identifier Constants

```
enum {
    cFrameColor = 0,
    cBodyColor = 1,
    cTextColor = 2,
    cThumbColor = 3,
    kNumberCtlCTabEntries = 4
};
```

Constants

`cFrameColor`

Produces foreground color for scroll arrows and gray area.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`cBodyColor`

Produces color of the scroll box.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`cTextColor`

Produces text color for scroll bars. Currently unused.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`cThumbColor`

Reserved.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

Discussion

The part identifier constants are not recommended with the Appearance Manager. When the Appearance Manager is available and you are using standard controls, part identifier constants are ignored and the colors are determined by the current theme. If you are creating your own control definition function, you can still use these constants in the `partIdentifier` field of a control color table structure to draw a control using colors other than the system default and to identify the part of a control that a color affects.

When the Appearance Manager is not present, you can use these constants in the `partIdentifier` field of a control color table resource `'cctb'` and the `partIdentifier` field of a control color table structure to identify the part of the control that the color affects.

Note that the colors you specify in the color table are blended to produce the colors that are actually used.

Picture Control Definition ID Constants

```
enum {
    kControlPictureProc = 304,
    kControlPictureNoTrackProc = 305
};
```

Constants

`kControlPictureProc`

Resource ID: 19

Picture control. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `ControlDefinitions.h`.

`kControlPictureNoTrackProc`

Resource ID: 19

Non-tracking picture. Immediately returns `kControlPicturePart` as the part code hit without tracking.

This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `ControlDefinitions.h`.

Discussion

When creating a control, your application supplies a control definition ID to one of the Control Manager control-creation functions or to the control resource; see `'CNTL'`. The control definition ID indicates the type of control to create. A control definition ID is an integer that contains the resource ID of a control definition function in its upper 12 bits and a variation code in its lower 4 bits. A control definition ID is derived as follows:

control definition ID = 16 * (`'CDEF'` resource ID) + variation code

A control definition function determines how a control generally looks and behaves. Control definition functions are stored as resources of type `'CDEF'`. Various Control Manager functions call a control definition function whenever they need to perform some control-dependent action, such as drawing the control on the screen. For more information on how to create a control definition function, see `ControlDefProcPtr` (page 677).

A control definition function, in turn, can use a variation code to describe variations of the same basic control. For example, all pop-up arrows share the same basic control definition function, which is stored in a resource of type `'CDEF'` and has a resource ID of 12. The standard pop-up arrow is large and points to the right; it has a control definition ID of 192. A variation of this is a large, left-pointing arrow, which has a control definition ID of 193. Still another variation, in which the arrow points up, has a control definition ID of 194.

Your application can use the constants listed here in place of control definition IDs.

Placard Control Definition ID Constant

```
enum {
    kControlPlacardProc = 224
};
```

Constants

`kControlPlacardProc`

Resource ID: 14

Placard. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HIContainerViews.h`.

Discussion

When creating a control, your application supplies a control definition ID to one of the Control Manager control-creation functions or to the control resource; see `'CNTL'`. The control definition ID indicates the type of control to create. A control definition ID is an integer that contains the resource ID of a control definition function in its upper 12 bits and a variation code in its lower 4 bits. A control definition ID is derived as follows:

control definition ID = 16 * (`'CDEF'` resource ID) + variation code

A control definition function determines how a control generally looks and behaves. Control definition functions are stored as resources of type `'CDEF'`. Various Control Manager functions call a control definition function whenever they need to perform some control-dependent action, such as drawing the control on the screen. For more information on how to create a control definition function, see `ControlDefProcPtr` (page 677).

A control definition function, in turn, can use a variation code to describe variations of the same basic control. For example, all pop-up arrows share the same basic control definition function, which is stored in a resource of type `'CDEF'` and has a resource ID of 12. The standard pop-up arrow is large and points to the right; it has a control definition ID of 192. A variation of this is a large, left-pointing arrow, which has a control definition ID of 193. Still another variation, in which the arrow points up, has a control definition ID of 194.

Your application can use the constant listed here in place of a control definition ID.

Pop-up Menu Title Constants

```
enum {
    popupTitleBold = 1 << 8,
    popupTitleItalic = 1 << 9,
    popupTitleUnderline = 1 << 10,
    popupTitleOutline = 1 << 11,
    popupTitleShadow = 1 << 12,
    popupTitleCondense = 1 << 13,
    popupTitleExtend = 1 << 14,
    popupTitleNoStyle = 1 << 15
};
```

Constants

`popupTitleBold`
Draw title in bold font style.
Available in Mac OS X v10.0 and later.
Declared in `HIPopupButton.h`.

`popupTitleItalic`
Draw title in italic font style.
Available in Mac OS X v10.0 and later.
Declared in `HIPopupButton.h`.

`popupTitleUnderline`
Draw title in underline font style.
Available in Mac OS X v10.0 and later.
Declared in `HIPopupButton.h`.

`popupTitleOutline`
Draw title in outline font style.
Available in Mac OS X v10.0 and later.
Declared in `HIPopupButton.h`.

`popupTitleShadow`
Draw title in shadow font style.
Available in Mac OS X v10.0 and later.
Declared in `HIPopupButton.h`.

`popupTitleCondense`
Draw title in condensed text font style.
Available in Mac OS X v10.0 and later.
Declared in `HIPopupButton.h`.

`popupTitleExtend`
Draw title in extended text font style.
Available in Mac OS X v10.0 and later.
Declared in `HIPopupButton.h`.

`popupTitleNoStyle`
Draw title in plain text font style.
Available in Mac OS X v10.0 and later.
Declared in `HIPopupButton.h`.

Discussion
When you define a pop-up menu control in a control resource, you can use one or more of these constants in the initial setting field to specify where and how to draw the pop-up menu control title.

Pop-up Menu Title Justification Constants

```
enum {
    popupTitleLeftJust = 0x00000000,
    popupTitleCenterJust = 0x00000001,
    popupTitleRightJust = 0x000000FF
};
```

Constants

`popupTitleLeftJust`
Place title to the left of the pop-up box.
Available in Mac OS X v10.0 and later.
Declared in `HIPopupButton.h`.

`popupTitleCenterJust`
Center title over the pop-up box.
Available in Mac OS X v10.0 and later.
Declared in `HIPopupButton.h`.

`popupTitleRightJust`
Place title to the right of the pop-up box.
Available in Mac OS X v10.0 and later.
Declared in `HIPopupButton.h`.

Discussion
When you define a pop-up menu control in a control resource, you can use one or more of these constants in the initial setting field to specify where and how to draw the pop-up menu control title.

Pop-up Arrow Control Definition ID Constants

```
enum {
    kControlPopupArrowEastProc = 192,
    kControlPopupArrowWestProc = 193,
    kControlPopupArrowNorthProc = 194,
    kControlPopupArrowSouthProc = 195,
    kControlPopupArrowSmallEastProc = 196,
    kControlPopupArrowSmallWestProc = 197,
    kControlPopupArrowSmallNorthProc = 198,
    kControlPopupArrowSmallSouthProc = 199
};
```

Constants

`kControlPopupArrowEastProc`

Resource ID: 12

Large, right-facing pop-up arrow. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HIPopupButton.h`.

`kControlPopupArrowWestProc`

Resource ID: 12

Large, left-facing pop-up arrow. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HIPopupButton.h`.

`kControlPopupArrowNorthProc`

Resource ID: 12

Large, up-facing pop-up arrow. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HIPopupButton.h`.

`kControlPopupArrowSouthProc`

Resource ID: 12

Large, down-facing pop-up arrow. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HIPopupButton.h`.

`kControlPopupArrowSmallEastProc`

Resource ID: 12

Small, right-facing pop-up arrow. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HIPopupButton.h`.

`kControlPopupArrowSmallWestProc`
Resource ID: 12

Small, left-facing pop-up arrow. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HIPopupButton.h`.

`kControlPopupArrowSmallNorthProc`
Resource ID: 12

Small, up-facing pop-up arrow. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HIPopupButton.h`.

`kControlPopupArrowSmallSouthProc`
Resource ID: 12

Small, down-facing pop-up arrow. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HIPopupButton.h`.

Discussion
When creating a control, your application supplies a control definition ID to one of the Control Manager control-creation functions or to the control resource; see `'CNTL'`. The control definition ID indicates the type of control to create. A control definition ID is an integer that contains the resource ID of a control definition function in its upper 12 bits and a variation code in its lower 4 bits. A control definition ID is derived as follows:

control definition ID = 16 * (`'CDEF'` resource ID) + variation code

A control definition function determines how a control generally looks and behaves. Control definition functions are stored as resources of type `'CDEF'`. Various Control Manager functions call a control definition function whenever they need to perform some control-dependent action, such as drawing the control on the screen. For more information on how to create a control definition function, see `ControlDefProcPtr` (page 677).

A control definition function, in turn, can use a variation code to describe variations of the same basic control. For example, all pop-up arrows share the same basic control definition function, which is stored in a resource of type `'CDEF'` and has a resource ID of 12. The standard pop-up arrow is large and points to the right; it has a control definition ID of 192. A variation of this is a large, left-pointing arrow, which has a control definition ID of 193. Still another variation, in which the arrow points up, has a control definition ID of 194.

Your application can use the constants listed here in place of control definition IDs.

Pop-up Button Control Data Tag Constants

```
enum {
    kControlPopupButtonMenuHandleTag = 'mhan',
    kControlPopupButtonMenuRefTag = 'mhan',
    kControlPopupButtonMenuIDTag = 'mnid'
};
```

Constants

`kControlPopupButtonMenuHandleTag`

Gets or sets the menu handle for a popup button. Available with Appearance Manager 1.0.1 and later.

Data type returned or set: `MenuHandle`

Available in Mac OS X v10.0 and later.

Declared in `HIPopupButton.h`.

`kControlPopupButtonMenuRefTag`

Gets or sets the menu reference assigned to a popup button. If setting the menu reference, the popup button does not own the menu, so you must dispose of it yourself. To allow the popup button to take ownership of the menu, use the `kControlPopupButtonOwnedMenuRefTag` tag (defined in "Mac OS 8.5 Pop-up Button Control Data Tag Constants" (page 789)) instead.

Data type returned or set: `MenuRef`

Available in Mac OS X v10.0 and later.

Declared in `HIPopupButton.h`.

`kControlPopupButtonMenuIDTag`

Gets or sets the menu ID for a popup button. Available with Appearance Manager 1.0.1 and later.

Data type returned or set: `SInt16`

Available in Mac OS X v10.0 and later.

Declared in `HIPopupButton.h`.

Discussion

You can use the control data tag constants to set or obtain data that is associated with a control. The control data tag constants are passed in the `inTagName` parameters of `SetControlData` (page 652) and `GetControlData` (page 594) to specify the piece of data in a control that you wish to set or get. You can also pass these constants in the `inTagName` parameter of `GetControlDataSize` (page 596) if you wish to determine the size of variable-length control data. These constants can also be used by custom control definition functions that return the feature bit `kControlSupportsDataAccess` in response to a `kControlMsgGetFeatures` message.

The data that your application sets or obtains can be of various types, dependent upon the control. Therefore, the descriptions of the control data tag constants list the data types for the information that you can set in the `inData` parameter to the `SetControlData` function and that you can get in the `inBuffer` parameter to the `GetControlData` function.

Pop-up Button Control Definition ID Constants

```
enum {
    kControlPopupButtonProc = 400,
    kControlPopupFixedWidthVariant = 1 << 0,
    kControlPopupVariableWidthVariant = 1 << 1,
    kControlPopupUseAddResMenuVariant = 1 << 2,
    kControlPopupUseWFontVariant = kControlUsesOwningWindowsFontVariant
};
```

Constants

`kControlPopupButtonProc`
Resource ID: 25

Appearance-compliant standard pop-up menu. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HIPopupButton.h`.

`kControlPopupFixedWidthVariant`
(+ `kControlPopupButtonProc`)

Resource ID: 25

Appearance-compliant fixed-width pop-up menu. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HIPopupButton.h`.

`kControlPopupVariableWidthVariant`
(+ `kControlPopupButtonProc`)

Resource ID: 25

Appearance-compliant variable-width pop-up menu. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HIPopupButton.h`.

`kControlPopupUseAddResMenuVariant`
(+ `kControlPopupButtonProc`)

Resource ID: 25

Appearance-compliant pop-up menu with a value of type `ResType` in the `contrlRfCon` field of the control structure. The Menu Manager adds resources of this type to the menu.

This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HIPopupButton.h`.

`kControlPopupUseWFontVariant`
(+ `kControlPopupButtonProc`)

Resource ID: 25

Appearance-compliant pop-up menu with control title in window font. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HIPopupButton.h`.

Discussion
When creating a control, your application supplies a control definition ID to one of the Control Manager control-creation functions or to the control resource; see `'CNTL'`. The control definition ID indicates the type of control to create. A control definition ID is an integer that contains the resource ID of a control definition function in its upper 12 bits and a variation code in its lower 4 bits. A control definition ID is derived as follows:

control definition ID = 16 * (`'CDEF'` resource ID) + variation code

A control definition function determines how a control generally looks and behaves. Control definition functions are stored as resources of type `'CDEF'`. Various Control Manager functions call a control definition function whenever they need to perform some control-dependent action, such as drawing the control on the screen. For more information on how to create a control definition function, see `ControlDefProcPtr` (page 677).

A control definition function, in turn, can use a variation code to describe variations of the same basic control. For example, all pop-up arrows share the same basic control definition function, which is stored in a resource of type `'CDEF'` and has a resource ID of 12. The standard pop-up arrow is large and points to the right; it has a control definition ID of 192. A variation of this is a large, left-pointing arrow, which has a control definition ID of 193. Still another variation, in which the arrow points up, has a control definition ID of 194.

Your application can use the constants listed here in place of control definition IDs.

Pop-up Width Constants

```
enum {
    popupFixedWidth = 1 << 0,
    popupVariableWidth = 1 << 1,
    popupUseAddResMenu = 1 << 2,
    popupUseWFont = 1 << 3
};
```

Pre–Appearance Control Definition ID Constants

```
enum {
    pushButProc = 0,
    checkBoxProc = 1,
    radioButProc = 2,
    scrollBarProc = 16,
    popupMenuProc = 1008
};
```

Constants

`pushButProc`

Resource ID: 0

Pre-Appearance push button.

`pushButProc` + `kControlUsesOwningWindowsFontVariant`:

Resource ID: 0

Pre-Appearance push button with its text in the window font.

Available in Mac OS X v10.0 and later.

Declared in `ControlDefinitions.h`.

`checkBoxProc`

Resource ID: 0

Pre-Appearance checkbox.

`checkBoxProc` + `kControlUsesOwningWindowsFontVariant`:

Resource ID: 0

Pre-Appearance checkbox with a control title in the window font.

Available in Mac OS X v10.0 and later.

Declared in `ControlDefinitions.h`.

`radioButProc`

Resource ID: 0

Pre-Appearance radio button.

`radioButProc` + `kControlUsesOwningWindowsFontVariant`:

Resource ID: 0

Pre-Appearance radio button with a title in the window font.

Available in Mac OS X v10.0 and later.

Declared in `ControlDefinitions.h`.

`scrollBarProc`

Resource ID: 0

Pre-Appearance scroll bar.

Available in Mac OS X v10.0 and later.

Declared in `ControlDefinitions.h`.

`popupMenuProc`

Resource ID: 63

Pre-Appearance standard pop-up menu.

`popupMenuProc + popupFixedWidth`:

Resource ID: 63; Control Definition ID: 1009

Pre-Appearance, fixed-width pop-up menu.

`popupMenuProc + popupVariableWidth`

Resource ID: 63; Control Definition ID: 1010

Pre-Appearance, variable-width pop-up menu.

`popupMenuProc + popupUseAddResMenu`

Resource ID: 63; Control Definition ID: 1012

Pre-Appearance pop-up menu with a value of type `ResType` in the `contrlRfCon` field of the control structure. The Menu Manager adds resources of this type to the menu.

`popupMenuProc + popupUseWFont`

Resource ID: 63; Control Definition ID: 1016

Pre-Appearance pop-up menu with a control title in the window font.

Available in Mac OS X v10.0 and later.

Declared in `ControlDefinitions.h`.

Discussion

When creating a control, your application supplies a control definition ID to one of the Control Manager control-creation functions or to the control resource; see `'CNTL'`. The control definition ID indicates the type of control to create. A control definition ID is an integer that contains the resource ID of a control definition function in its upper 12 bits and a variation code in its lower 4 bits. A control definition ID is derived as follows:

control definition ID = 16 * (`'CDEF'` resource ID) + variation code

A control definition function determines how a control generally looks and behaves. Control definition functions are stored as resources of type `'CDEF'`. Various Control Manager functions call a control definition function whenever they need to perform some control-dependent action, such as drawing the control on the screen. For more information on how to create a control definition function, see "Defining Your Own Control Definition Function".

A control definition function, in turn, can use a variation code to describe variations of the same basic control. For example, all pop-up arrows share the same basic control definition function, which is stored in a resource of type `'CDEF'` and has a resource ID of 12. The standard pop-up arrow is large and points to the right; it has a control definition ID of 192. A variation of this is a large, left-pointing arrow, which has a control definition ID of 193. Still another variation, in which the arrow points up, has a control definition ID of 194.

Your application can use the constants listed here in place of control definition IDs.

If your application contains code that uses the older, pre-Appearance control definition IDs or their constants, your application can use the Appearance Manager to map the old IDs to those for the new, updated controls introduced by the Appearance Manager. In particular, the control definition IDs for pre-Appearance checkboxes, buttons, scroll bars, radio buttons, and pop-up menus will be automatically mapped to Appearance-compliant equivalents.

Progress Bar Control Data Tag Constants

```
enum {
    kControlProgressBarIndeterminateTag = 'inde',
    kControlProgressBarAnimatingTag = 'anim'
};
```

Constants

`kControlProgressBarIndeterminateTag`

Gets or sets whether a progress indicator is determinate or indeterminate.

Data type returned or set: `Boolean`; if `true`, switches to an indeterminate progress indicator; if `false`, switches to an determinate progress indicator.

Available in Mac OS X v10.0 and later.

Declared in `HIProgressViews.h`.

Discussion

You can use this control data tag constant to set or obtain data that is associated with a control. This constant is passed in the `inTagName` parameters of `SetControlData` (page 652) and `GetControlData` (page 594) to specify the piece of data in a control that you wish to set or get. You can also pass this constant in the `inTagName` parameter of `GetControlDataSize` (page 596) if you wish to determine the size of variable-length control data. This constant can also be used by custom control definition functions that return the feature bit `kControlSupportsDataAccess` in response to a `kControlMsgGetFeatures` message.

The data that your application sets or obtains can be of various types, dependent upon the control. Therefore, the description of this control data tag constant lists the data type for the information that you can set in the `inData` parameter to the `SetControlData` function and that you can get in the `inBuffer` parameter to the `GetControlData` function.

Version Notes

This control data tag constant is available with Appearance Manager 1.0 and later.

Progress Bar Control Definition ID Constants

```
enum {
    kControlProgressBarProc = 80,
    kControlRelevanceBarProc = 81
};
```

Constants

`kControlProgressBarProc`

Resource ID: 5

Progress indicator. To make the control determinate or indeterminate, set the `kControlProgressBarIndeterminateTag` constant; see "Progress Bar Control Data Tag Constants" (page 804). Progress indicators are only horizontal in orientation; vertical progress indicators are not currently supported.

This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HIProgressViews.h`.

Discussion

When creating a control, your application supplies a control definition ID to one of the Control Manager control-creation functions or to the control resource; see `'CNTL'`. The control definition ID indicates the type of control to create. A control definition ID is an integer that contains the resource ID of a control definition function in its upper 12 bits and a variation code in its lower 4 bits. A control definition ID is derived as follows:

control definition ID = 16 * (`'CDEF'` resource ID) + variation code

A control definition function determines how a control generally looks and behaves. Control definition functions are stored as resources of type `'CDEF'`. Various Control Manager functions call a control definition function whenever they need to perform some control-dependent action, such as drawing the control on the screen. For more information on how to create a control definition function, see `ControlDefProcPtr` (page 677).

A control definition function, in turn, can use a variation code to describe variations of the same basic control. For example, all pop-up arrows share the same basic control definition function, which is stored in a resource of type `'CDEF'` and has a resource ID of 12. The standard pop-up arrow is large and points to the right; it has a control definition ID of 192. A variation of this is a large, left-pointing arrow, which has a control definition ID of 193. Still another variation, in which the arrow points up, has a control definition ID of 194.

Your application can use the constant listed here in place of a control definition ID.

Push Button Control Data Tag Constants

```
enum {
    kControlPushButtonDefaultTag = 'dflt',
    kControlPushButtonCancelTag = 'cncl'
};
```

Constants

`kControlPushButtonDefaultTag`

Tells Appearance-compliant button whether to draw a default ring, or returns whether the Appearance Manager draws a default ring for the button.

Data type returned or set: `Boolean`

Available in Mac OS X v10.0 and later.

Declared in `HIButtonViews.h`.

`kControlPushButtonCancelTag`

Gets or sets whether a given push button in a dialog or alert should be drawn with the theme-specific adornments for a Cancel button.

Data type retrieved or set: `Boolean`; default is `false`.

Available in Mac OS X v10.0 and later.

Declared in `HIButtonViews.h`.

Discussion

The Mac OS 8.5 Control Manager defines this new control data tag constant. This constant is passed in the `inTagName` parameters of the functions `SetControlData` and `GetControlData` to specify the piece of data in a control that you wish to set or get. You can also pass this constant in the `inTagName` parameter of the function `GetControlDataSize` if you wish to determine the size of variable-length control data. This constant can also be used by custom control definition functions that return the feature bit `kControlSupportsDataAccess` in response to a `kControlMsgGetFeatures` message.

The data that your application gets or sets can be of various types. The description here shows the data type for the information that you can set in the `inData` parameter to the `SetControlData` function and that you can get in the `inBuffer` parameter to the `GetControlData` function.

Radio Button Value Constants

```
enum {
    kControlRadioButtonUncheckedValue = 0,
    kControlRadioButtonCheckedValue = 1,
    kControlRadioButtonMixedValue = 2
};
```

Constants

`kControlRadioButtonUncheckedValue`

The radio button is unselected.

Available in Mac OS X v10.0 and later.

Declared in `HIButtonViews.h`.

`kControlRadioButtonCheckedValue`

The radio button is selected.

Available in Mac OS X v10.0 and later.

Declared in `HIButtonViews.h`.

`kControlRadioButtonMixedValue`
Mixed value. Indicates that a setting is on for some elements in a selection and off for others. This state only applies to standard Appearance-compliant radio buttons.

Available in Mac OS X v10.0 and later.

Declared in `HIButtonViews.h`.

Discussion
These constants specify the value of a standard radio button control and are passed in the `newValue` parameter of `SetControlValue` (page 661) and are returned by `GetControlValue` (page 606).

Version Notes
The radio button value constants are changed with Appearance Manager 1.0 to support mixed-value radio buttons.

Radio Group Control Definition ID Constant

```
enum {
    kControlRadioGroupProc = 416
};
```

Constants
`kControlRadioGroupProc`
Resource ID: 26

Radio group. Embedder control for controls that have set the feature bit `kControlHasRadioBehavior`.

This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HIButtonViews.h`.

Discussion
When creating a control, your application supplies a control definition ID to one of the Control Manager control-creation functions or to the control resource; see `'CNTL'`. The control definition ID indicates the type of control to create. A control definition ID is an integer that contains the resource ID of a control definition function in its upper 12 bits and a variation code in its lower 4 bits. A control definition ID is derived as follows:

control definition ID = 16 * (`'CDEF'` resource ID) + variation code

A control definition function determines how a control generally looks and behaves. Control definition functions are stored as resources of type `'CDEF'`. Various Control Manager functions call a control definition function whenever they need to perform some control-dependent action, such as drawing the control on the screen. For more information on how to create a control definition function, see `ControlDefProcPtr` (page 677).

A control definition function, in turn, can use a variation code to describe variations of the same basic control. For example, all pop-up arrows share the same basic control definition function, which is stored in a resource of type `'CDEF'` and has a resource ID of 12. The standard pop-up arrow is large and points to the right; it has a control definition ID of 192. A variation of this is a large, left-pointing arrow, which has a control definition ID of 193. Still another variation, in which the arrow points up, has a control definition ID of 194.

Your application can use the constant listed here in place of a control definition ID.

Scroll Bar Control Definition ID Constants

```
enum {
    kControlScrollBarProc = 384,
    kControlScrollBarLiveProc = 386
};
```

Constants

`kControlScrollBarProc`

Resource ID: 24

Appearance-compliant scroll bar. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HIScrollView.h`.

`kControlScrollBarLiveProc`

Resource ID: 24

Appearance-compliant scroll bar with live feedback. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HIScrollView.h`.

Discussion

When creating a control, your application supplies a control definition ID to one of the Control Manager control-creation functions or to the control resource; see `'CNTL'`. The control definition ID indicates the type of control to create. A control definition ID is an integer that contains the resource ID of a control definition function in its upper 12 bits and a variation code in its lower 4 bits. A control definition ID is derived as follows:

control definition ID = 16 * (`'CDEF'` resource ID) + variation code

A control definition function determines how a control generally looks and behaves. Control definition functions are stored as resources of type `'CDEF'`. Various Control Manager functions call a control definition function whenever they need to perform some control-dependent action, such as drawing the control on the screen. For more information on how to create a control definition function, see `ControlDefProcPtr` (page 677).

A control definition function, in turn, can use a variation code to describe variations of the same basic control. For example, all pop-up arrows share the same basic control definition function, which is stored in a resource of type `'CDEF'` and has a resource ID of 12. The standard pop-up arrow is large and points to the right; it has a control definition ID of 192. A variation of this is a large, left-pointing arrow, which has a control definition ID of 193. Still another variation, in which the arrow points up, has a control definition ID of 194.

Your application can use the constants listed here in place of control definition IDs.

Scrolling Text Box Control Data Tag Constants

```
enum {
    kControlScrollTextBoxDelayBeforeAutoScrollTag = 'stdl',
    kControlScrollTextBoxDelayBetweenAutoScrollTag = 'scdl',
    kControlScrollTextBoxAutoScrollAmountTag = 'samt',
    kControlScrollTextBoxContentsTag = 'tres',
    kControlScrollTextBoxAnimatingTag = 'anim'
};
```

Constants

`kControlScrollTextBoxDelayBeforeAutoScrollTag`

Gets or sets the number of ticks to delay before the initial scrolling of an auto-scrolling text box control begins.

Data type retrieved or set: `UInt32`

Available in Mac OS X v10.0 and later.

Declared in `ControlDefinitions.h`.

`kControlScrollTextBoxDelayBetweenAutoScrollTag`

Gets or sets the number of ticks to delay between each unit of scrolling, for an auto-scrolling text box control. (The unit of scrolling for the auto-scrolling text box control is one pixel at a time, unless your application changes this value by calling the `SetControlData` function.)

Data type retrieved or set: `UInt32`

Available in Mac OS X v10.0 and later.

Declared in `ControlDefinitions.h`.

`kControlScrollTextBoxAutoScrollAmountTag`

Gets or sets the number of pixels by which an auto-scrolling text box control scrolls; default is 1.

Data type retrieved or set: `UInt16`

Available in Mac OS X v10.0 and later.

Declared in `ControlDefinitions.h`.

`kControlScrollTextBoxContentsTag`

Sets the ID of a `'TEXT'` resource—and, optionally, a `'styl'` resource—to be used as the content in a scrolling or auto-scrolling text box control.

Data type set: `SInt16`

Available in Mac OS X v10.0 and later.

Declared in `ControlDefinitions.h`.

Discussion

The Mac OS 8.5 Control Manager defines these new control data tag constants. These constants are passed in the `inTagName` parameters of the functions `SetControlData` and `GetControlData` to specify the piece of data in a control that you wish to set or get. You can also pass these constants in the `inTagName` parameter of the function `GetControlDataSize` if you wish to determine the size of variable-length control data. These constants can also be used by custom control definition functions that return the feature bit `kControlSupportsDataAccess` in response to a `kControlMsgGetFeatures` message.

The data that your application gets or sets can be of various types. The descriptions here show the data types for the information that you can set in the `inData` parameter to the `SetControlData` function and that you can get in the `inBuffer` parameter to the `GetControlData` function.

Scrolling Text Box Control Definition ID Constants

```
enum {
    kControlScrollTextBoxProc = 432,
    kControlScrollTextBoxAutoScrollProc = 433
};
```

Constants

`kControlScrollTextBoxProc`

Identifies the standard variant of the scrolling text box (`'CDEF'` resource ID 27), which contains a scroll bar. Your application can use the `kControlScrollTextBoxProc` ID to create a scrolling box of non-editable text, such as is used for an "About" box. You must pass the `NewControl` function the ID of a `'TEXT'` resource—and, optionally, a `'styl'` resource—to be used for the initial value of the control. The minimum and maximum values are reserved for the `kControlScrollTextBoxProc` variant and should be set to 0.

Available in Mac OS X v10.0 and later.

Declared in `ControlDefinitions.h`.

`kControlScrollTextBoxAutoScrollProc`

Identifies the auto-scrolling variant of the scrolling text box (`'CDEF'` resource ID 27); this variant does not contain a scroll bar. Your application can use the `kControlScrollTextBoxAutoScrollProc` ID to create a scrolling box of non-editable text, such as is used for an "About" box. You must pass the `NewControl` function the ID of a `'TEXT'` resource—and, optionally, a `'styl'` resource—to be used for the initial value of the control. For the minimum value of the control, pass a value equal to the delay, in ticks, before the control begins scrolling this delay will also be used between when scrolling completes and when it begins again. For the maximum value of the control, pass a value equal to the delay, in ticks, between each unit of scrolling. The unit of scrolling for the auto-scrolling text box control is one pixel at a time, unless your application changes this value by calling the `SetControlData` function. Note that in order to advance the content of the text box—that is, to scroll the content—you must call the `IdleControls` function on a periodic basis, such as whenever you receive a null event.

Available in Mac OS X v10.0 and later.

Declared in `ControlDefinitions.h`.

Discussion

The Mac OS 8.5 Control Manager defines these new control definition IDs.

When creating a control, your application supplies a control definition ID to one of the Control Manager control-creation functions or to the control resource; see `'CNTL'`. The control definition ID indicates the type of control to create. A control definition ID is an integer that contains the resource ID of a control definition function in its upper 12 bits and a variation code in its lower 4 bits. A control definition ID is derived as follows:

control definition ID = 16 * (`'CDEF'` resource ID) + variation code

A control definition function determines how a control generally looks and behaves. Control definition functions are stored as resources of type `'CDEF'`. Various Control Manager functions call a control definition function whenever they need to perform some control-dependent action, such as drawing the control on the screen. For more information on how to create a control definition function, see `ControlDefProcPtr` (page 677).

A control definition function, in turn, can use a variation code to describe variations of the same basic control. For example, all pop-up arrows share the same basic control definition function, which is stored in a resource of type `'CDEF'` and has a resource ID of 12. The standard pop-up arrow is large and points to the right; it has a control definition ID of 192. A variation of this is a large, left-pointing arrow, which has a control definition ID of 193. Still another variation, in which the arrow points up, has a control definition ID of 194.

Your application can use the constants listed here in place of control definition IDs. These constants, and their associated IDs, are not supported unless the Appearance Manager is available.

Separator Line Control Definition ID Constant

```
enum {
    kControlSeparatorLineProc = 144
};
```

Constants

`kControlSeparatorLineProc`

Resource ID: 9

Separator line.

Available in Mac OS X v10.0 and later.

Declared in `HISeparator.h`.

Discussion

When creating a control, your application supplies a control definition ID to one of the Control Manager control-creation functions or to the control resource; see `'CNTL'`. The control definition ID indicates the type of control to create. A control definition ID is an integer that contains the resource ID of a control definition function in its upper 12 bits and a variation code in its lower 4 bits. A control definition ID is derived as follows:

control definition ID = 16 * (`'CDEF'` resource ID) + variation code

A control definition function determines how a control generally looks and behaves. Control definition functions are stored as resources of type `'CDEF'`. Various Control Manager functions call a control definition function whenever they need to perform some control-dependent action, such as drawing the control on the screen. For more information on how to create a control definition function, see `ControlDefProcPtr` (page 677).

A control definition function, in turn, can use a variation code to describe variations of the same basic control. For example, all pop-up arrows share the same basic control definition function, which is stored in a resource of type `'CDEF'` and has a resource ID of 12. The standard pop-up arrow is large and points to the right; it has a control definition ID of 192. A variation of this is a large, left-pointing arrow, which has a control definition ID of 193. Still another variation, in which the arrow points up, has a control definition ID of 194.

Your application can use the constant listed here in place of a control definition ID.

Slider Control Definition ID Constants

```
enum {
    kControlSliderProc = 48,
    kControlSliderLiveFeedback = (1 << 0),
    kControlSliderHasTickMarks = (1 << 1),
    kControlSliderReverseDirection = (1 << 2),
    kControlSliderNonDirectional = (1 << 3)
};
```

Constants

`kControlSliderProc`

Resource ID: 3

Slider. Your application calls the function `SetControlAction` (page 649) to set the last value for the control. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HISlider.h`.

`kControlSliderLiveFeedback`

(+ `kControlSliderProc`)

Resource ID: 3

Slider with live feedback. The value of the control is updated automatically by the Control Manager before your action function is called. If no application-defined action function is supplied, the slider draws an outline of the indicator as the user moves it. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HISlider.h`.

`kControlSliderHasTickMarks`

(+ `kControlSliderProc`)

Resource ID: 3

Slider with tick marks. The control rectangle must be large enough to include the tick marks. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HISlider.h`.

`kControlSliderReverseDirection`

(+ `kControlSliderProc`)

Resource ID: 3

Slider with a directional indicator. The indicator is positioned perpendicularly to the slider; that is, if the slider is horizontal, the indicator points up, and if the slider is vertical, the indicator points left. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HISlider.h`.

`kControlSliderNonDirectional`
(+ `kControlSliderProc`)

Resource ID: 3

Slider with a rectangular, non-directional indicator. This variant overrides the `kSliderReverseDirection` and `kSliderHasTickMarks` variants. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HISlider.h`.

Discussion
When creating a control, your application supplies a control definition ID to one of the Control Manager control-creation functions or to the control resource; see `'CNTL'`. The control definition ID indicates the type of control to create. A control definition ID is an integer that contains the resource ID of a control definition function in its upper 12 bits and a variation code in its lower 4 bits. A control definition ID is derived as follows:

control definition ID = 16 * (`'CDEF'` resource ID) + variation code

A control definition function determines how a control generally looks and behaves. Control definition functions are stored as resources of type `'CDEF'`. Various Control Manager functions call a control definition function whenever they need to perform some control-dependent action, such as drawing the control on the screen. For more information on how to create a control definition function, see `ControlDefProcPtr` (page 677).

A control definition function, in turn, can use a variation code to describe variations of the same basic control. For example, all pop-up arrows share the same basic control definition function, which is stored in a resource of type `'CDEF'` and has a resource ID of 12. The standard pop-up arrow is large and points to the right; it has a control definition ID of 192. A variation of this is a large, left-pointing arrow, which has a control definition ID of 193. Still another variation, in which the arrow points up, has a control definition ID of 194.

Your application can use the constants listed here in place of control definition IDs.

Static Text Control Data Tag Constants

```
enum {
 kControlStaticTextStyleTag = kControlFontStyleTag,
 kControlStaticTextTextTag = 'text',
 kControlStaticTextTextHeightTag = 'thei',
 kControlStaticTextTruncTag = 'trun',
 kControlStaticTextCFStringTag = 'cfst',
 kControlStaticTextIsMultilineTag = 'stim'
};
```

Constants
`kControlStaticTextTextTag`

Gets or sets text in a static text control.

Data type returned or set: character buffer.

Declared in `HITextViews.h`.

Available with Appearance Manager 1.0 and later.

`kControlStaticTextTextHeightTag`

Gets the height of text in a static text control. Available with Appearance Manager 1.0 and later.

Data type returned or set:`SInt16`

Available in Mac OS X v10.0 and later.

Declared in `HITextViews.h`.

`kControlStaticTextTruncTag`

Gets or sets the control's text truncation style. Truncation will not occur unless `kControlStaticTextIsMultilineTag` is set to `false`.

Data type returned or set: `TruncCode`. The value `truncEnd` indicates that characters are truncated off the end of the string; the value `truncMiddle` indicates that characters are truncated from the middle of the string. Default is a value of -1, which indicates that no truncation occurs and the text is wrapped instead.

Available with Appearance Manager 1.1 (Mac OS 8.5) and later.

Available in Mac OS X v10.0 and later.

Declared in `HITextViews.h`.

`kControlStaticTextCFStringTag`

Gets or sets the control's current text. When setting the text, the control retains the string, so you may release the string after calling `SetControlData`. If the string you set is mutable, the control will make a copy of the string, so any changes to the string after calling `SetControlData` will not affect the control. When retrieving the text, the control retains the string before returning it to you, so you must release the string after you are done with it.

Data type returned or set: `CFStringRef`

Available in CarbonLib 1.5 and later, and Mac OS X v10.0 and later.

Available in Mac OS X v10.0 and later.

Declared in `HITextViews.h`.

`kControlStaticTextIsMultilineTag`

Gets or sets a flag specifying whether the control draws its text in multiple lines if the text is too wide for the control bounds. If `false`, the control always draws the text in a single line.

Data type returned or set: `Boolean`

Declared in `HITextViews.h`.

Available in Mac OS X v10.1 and later.

Discussion

You can use the control data tag constants to set or obtain data that is associated with a control. The control data tag constants are passed in the `inTagName` parameters of `SetControlData` (page 652) and `GetControlData` (page 594) to specify the piece of data in a control that you wish to set or get. You can also pass these constants in the `inTagName` parameter of `GetControlDataSize` (page 596) if you wish to determine the size of variable-length control data. These constants can also be used by custom control definition functions that return the feature bit `kControlSupportsDataAccess` in response to a `kControlMsgGetFeatures` message.

The data that your application sets or obtains can be of various types, dependent upon the control. Therefore, the descriptions of the control data tag constants list the data types for the information that you can set in the `inData` parameter to the `SetControlData` function and that you can get in the `inBuffer` parameter to the `GetControlData` function.

Static Text Control Definition ID Constant

```
enum {
    kControlStaticTextProc = 288
};
```

Constants

`kControlStaticTextProc`

Resource ID: 18

Static text field. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HITextViews.h`.

Discussion

When creating a control, your application supplies a control definition ID to one of the Control Manager control-creation functions or to the control resource; see `'CNTL'`. The control definition ID indicates the type of control to create. A control definition ID is an integer that contains the resource ID of a control definition function in its upper 12 bits and a variation code in its lower 4 bits. A control definition ID is derived as follows:

control definition ID = 16 * (`'CDEF'` resource ID) + variation code

A control definition function determines how a control generally looks and behaves. Control definition functions are stored as resources of type `'CDEF'`. Various Control Manager functions call a control definition function whenever they need to perform some control-dependent action, such as drawing the control on the screen. For more information on how to create a control definition function, see `ControlDefProcPtr` (page 677).

A control definition function, in turn, can use a variation code to describe variations of the same basic control. For example, all pop-up arrows share the same basic control definition function, which is stored in a resource of type `'CDEF'` and has a resource ID of 12. The standard pop-up arrow is large and points to the right; it has a control definition ID of 192. A variation of this is a large, left-pointing arrow, which has a control definition ID of 193. Still another variation, in which the arrow points up, has a control definition ID of 194.

Your application can use the constant listed here in place of a control definition ID.

Text Proc Constants

```
enum {
    staticTextProc = 256,
    editTextProc = 272,
    iconProc = 288,
    userItemProc = 304,
    pictItemProc = 320
};
```

Tab Control Data Tag Constants

```
enum {
    kControlTabContentRectTag = 'rect',
    kControlTabEnabledFlagTag = 'enab',
    kControlTabFontStyleTag = kControlFontStyleTag
};
```

Constants

`kControlTabContentRectTag`

Gets the content rectangle of a tab control.

Data type returned: `Rect`

Available in Mac OS X v10.0 and later.

Declared in `HITabbedView.h`.

`kControlTabEnabledFlagTag`

Enables or disables a single tab in a tab control.

Data type returned or set: `Boolean`; if `true`, enabled; if `false`, disabled.

Available in Mac OS X v10.0 and later.

Declared in `HITabbedView.h`.

Discussion

You can use the control data tag constants to set or obtain data that is associated with a control. The control data tag constants are passed in the `inTagName` parameters of `SetControlData` (page 652) and `GetControlData` (page 594) to specify the piece of data in a control that you wish to set or get. You can also pass these constants in the `inTagName` parameter of `GetControlDataSize` (page 596) if you wish to determine the size of variable-length control data. These constants can also be used by custom control definition functions that return the feature bit `kControlSupportsDataAccess` in response to a `kControlMsgGetFeatures` message.

The data that your application sets or obtains can be of various types, dependent upon the control. Therefore, the descriptions of the control data tag constants list the data types for the information that you can set in the `inData` parameter to the `SetControlData` function and that you can get in the `inBuffer` parameter to the `GetControlData` function.

Version Notes

The control data tag constants are available with Appearance Manager 1.0 and later.

Tab Control Definition IDs

```
enum {
    kControlTabLargeProc = 128,
    kControlTabSmallProc = 129,
    kControlTabLargeNorthProc = 128,
    kControlTabSmallNorthProc = 129,
    kControlTabLargeSouthProc = 130,
    kControlTabSmallSouthProc = 131,
    kControlTabLargeEastProc = 132,
    kControlTabSmallEastProc = 133,
    kControlTabLargeWestProc = 134,
    kControlTabSmallWestProc = 135
};
```

Constants

`kControlTabLargeProc`

Resource ID: 8

Normal tab control. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HITabbedView.h`.

`kControlTabSmallProc`

Resource ID: none

Small tab control. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HITabbedView.h`.

Discussion

When creating a control, your application supplies a control definition ID to one of the Control Manager control-creation functions or to the control resource; see `'CNTL'`. The control definition ID indicates the type of control to create. A control definition ID is an integer that contains the resource ID of a control definition function in its upper 12 bits and a variation code in its lower 4 bits. A control definition ID is derived as follows:

control definition ID = 16 * (`'CDEF'` resource ID) + variation code

A control definition function determines how a control generally looks and behaves. Control definition functions are stored as resources of type `'CDEF'`. Various Control Manager functions call a control definition function whenever they need to perform some control-dependent action, such as drawing the control on the screen. For more information on how to create a control definition function, see `ControlDefProcPtr` (page 677).

A control definition function, in turn, can use a variation code to describe variations of the same basic control. For example, all pop-up arrows share the same basic control definition function, which is stored in a resource of type `'CDEF'` and has a resource ID of 12. The standard pop-up arrow is large and points to the right; it has a control definition ID of 192. A variation of this is a large, left-pointing arrow, which has a control definition ID of 193. Still another variation, in which the arrow points up, has a control definition ID of 194.

Your application can use the constants listed here in place of control definition IDs.

Tab Control Info Tag Constant

```
enum {
    kControlTabInfoTag = 'tabi'
};
```

Constants

`kControlTabInfoTag`

Gets or sets information for a tab in a tab control; see `ControlTabInfoRec` (page 710).

Data type returned or set: `ControlTabInfoRec`.

Available in Mac OS X v10.0 and later.

Declared in `HITabbedView.h`.

Discussion

You can use this control data tag constant to set or obtain data that is associated with a control. This constant is passed in the `inTagName` parameters of `SetControlData` (page 652) and `GetControlData` (page 594) to specify the piece of data in a control that you wish to set or get. You can also pass this constant in the `inTagName` parameter of `GetControlDataSize` (page 596) if you wish to determine the size of variable-length control data. This constant can also be used by custom control definition functions that return the feature bit `kControlSupportsDataAccess` in response to a `kControlMsgGetFeatures` message.

The data that your application sets or obtains can be of various types, dependent upon the control. Therefore, the description of this control data tag constant lists the data type for the information that you can set in the `inData` parameter to the `SetControlData` function and that you can get in the `inBuffer` parameter to the `GetControlData` function.

Version Notes

This control data tag constant is available with Appearance Manager 1.0.1 and later.

Triangle Control Data Tag Constant

```
enum {
    kControlTriangleLastValueTag = 'last'
};
```

Constants

`kControlTriangleLastValueTag`

Gets or sets the last value of a disclosure triangle. Used primarily for setting up a disclosure triangle properly when using the auto-toggle variant.

Data type returned or set: `SInt16`

Available in Mac OS X v10.0 and later.

Declared in `HIDisclosureViews.h`.

Discussion

You can use this control data tag constant to set or obtain data that is associated with a control. This constant is passed in the `inTagName` parameters of `SetControlData` (page 652) and `GetControlData` (page 594) to specify the piece of data in a control that you wish to set or get. You can also pass this constant in the `inTagName` parameter of `GetControlDataSize` (page 596) if you wish to determine the size of variable-length control data. This constant can also be used by custom control definition functions that return the feature bit `kControlSupportsDataAccess` in response to a `kControlMsgGetFeatures` message.

The data that your application sets or obtains can be of various types, dependent upon the control. Therefore, the description of this control data tag constant lists the data type for the information that you can set in the `inData` parameter to the `SetControlData` function and that you can get in the `inBuffer` parameter to the `GetControlData` function.

Version Notes
This control data tag constant is available with Appearance Manager 1.0 and later.

Triangle Control Definition ID Constants

```
enum {
    kControlTriangleProc = 64,
    kControlTriangleLeftFacingProc = 65,
    kControlTriangleAutoToggleProc = 66,
    kControlTriangleLeftFacingAutoToggleProc = 67
};
```

Constants

`kControlTriangleProc`
Resource ID: 4

Disclosure triangle. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HIDisclosureViews.h`.

`kControlTriangleLeftFacingProc`
Resource ID: 4

Left-facing disclosure triangle. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HIDisclosureViews.h`.

`kControlTriangleAutoToggleProc`
Resource ID: 4

Auto-tracking disclosure triangle. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HIDisclosureViews.h`.

`kControlTriangleLeftFacingAutoToggleProc`
Resource ID: 4

Left-facing, auto-tracking disclosure triangle. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HIDisclosureViews.h`.

Discussion
When creating a control, your application supplies a control definition ID to one of the Control Manager control-creation functions or to the control resource; see `'CNTL'`. The control definition ID indicates the type of control to create. A control definition ID is an integer that contains the resource ID of a control definition function in its upper 12 bits and a variation code in its lower 4 bits. A control definition ID is derived as follows:

control definition ID = 16 * ('CDEF' resource ID) + variation code

A control definition function determines how a control generally looks and behaves. Control definition functions are stored as resources of type 'CDEF'. Various Control Manager functions call a control definition function whenever they need to perform some control-dependent action, such as drawing the control on the screen. For more information on how to create a control definition function, see ControlDefProcPtr (page 677).

A control definition function, in turn, can use a variation code to describe variations of the same basic control. For example, all pop-up arrows share the same basic control definition function, which is stored in a resource of type 'CDEF' and has a resource ID of 12. The standard pop-up arrow is large and points to the right; it has a control definition ID of 192. A variation of this is a large, left-pointing arrow, which has a control definition ID of 193. Still another variation, in which the arrow points up, has a control definition ID of 194.

Your application can use the constants listed here in place of control definition IDs.

User Item and User Pane Control Data Tag Constants

```
enum {
    kControlUserItemDrawProcTag = 'uidp',
    kControlUserPaneDrawProcTag = 'draw',
    kControlUserPaneHitTestProcTag = 'hitt',
    kControlUserPaneTrackingProcTag = 'trak',
    kControlUserPaneIdleProcTag = 'idle',
    kControlUserPaneKeyDownProcTag = 'keyd',
    kControlUserPaneActivateProcTag = 'acti',
    kControlUserPaneFocusProcTag = 'foci',
    kControlUserPaneBackgroundProcTag = 'back'
};
```

Constants

kControlUserItemDrawProcTag
: Gets or sets an application-defined item drawing function. If an embedding hierarchy is established, a user pane drawing function should be used instead of an item drawing function.

  Data type returned or set: UserItemUPP

  Available in Mac OS X v10.0 and later.

  Declared in HIContainerViews.h.

kControlUserPaneDrawProcTag
: Gets or sets a user pane drawing function; see ControlUserPaneBackgroundProcPtr (page 688). Indicates that the Control Manager needs to draw a control.

  Data type returned or set: ControlUserPaneDrawingUPP

  Available in Mac OS X v10.0 and later.

  Declared in HIContainerViews.h.

kControlUserPaneHitTestProcTag
: Gets or sets a user pane hit-testing function. Indicates that the Control Manager needs to determine if a control part was hit; see ControlUserPaneBackgroundProcPtr (page 688).

  Data type returned or set: ControlUserPaneHitTestUPP

  Available in Mac OS X v10.0 and later.

  Declared in HIContainerViews.h.

`kControlUserPaneTrackingProcTag`
: Gets or sets a user pane tracking function, which will be called when a control definition function returns the `kControlHandlesTracking` feature bit in response to a `kControlMsgGetFeatures` message. Indicates that a user pane handles its own tracking; see `ControlUserPaneBackgroundProcPtr` (page 688).

  Data type returned or set: `ControlUserPaneTrackingUPP`

  Available in Mac OS X v10.0 and later.

  Declared in `HIContainerViews.h`.

`kControlUserPaneIdleProcTag`
: Gets or sets a user pane idle function, which will be called when a control definition function returns the `kControlWantsIdle` feature bit in response to a `kControlMsgGetFeatures` message. Indicates that a user pane performs idle processing; see `ControlUserPaneBackgroundProcPtr` (page 688).

  Data type returned or set: `ControlUserPaneIdleUPP`

  Available in Mac OS X v10.0 and later.

  Declared in `HIContainerViews.h`.

`kControlUserPaneKeyDownProcTag`
: Gets or sets a user pane key down function, which will be called when a control definition function returns the `kControlSupportsFocus` feature bit in response to a `kControlMsgGetFeatures` message. Indicates that a user pane performs keyboard event processing; see `ControlUserPaneBackgroundProcPtr` (page 688).

  Data type returned or set: `ControlUserPaneKeyDownUPP`

  Available in Mac OS X v10.0 and later.

  Declared in `HIContainerViews.h`.

`kControlUserPaneActivateProcTag`
: Gets or sets a user pane activate function, which will be called when a control definition function returns the `kControlWantsActivate`feature bit in response to a `kControlMsgGetFeatures` message. Indicates that a user pane wants to be informed of activate and deactivate events; see `ControlUserPaneBackgroundProcPtr` (page 688).

  Data type returned or set: `ControlUserPaneActivateUPP`

  Available in Mac OS X v10.0 and later.

  Declared in `HIContainerViews.h`.

`kControlUserPaneFocusProcTag`
: Gets or sets a user pane keyboard focus function, which will be called when a control definition function returns the `kControlSupportsFocus` feature bit in response to a `kControlMsgGetFeatures` message. Indicates that a user pane handles keyboard focus; see `ControlUserPaneBackgroundProcPtr` (page 688).

  Data type returned or set: `ControlUserPaneFocusUPP`

  Available in Mac OS X v10.0 and later.

  Declared in `HIContainerViews.h`.

`kControlUserPaneBackgroundProcTag`

Gets or sets a user pane background function, which will be called when a control definition function returns the `kControlHasSpecialBackground`and `kControlSupportsEmbedding`feature bits in response to a `kControlMsgGetFeatures` message. Indicates that a user pane can set its background color or pattern; see `ControlUserPaneBackgroundProcPtr` (page 688).

Data type returned or set: `ControlUserPaneBackgroundUPP`

Available in Mac OS X v10.0 and later.

Declared in `HIContainerViews.h`.

Discussion
You can use the control data tag constants to set or obtain data that is associated with a control. The control data tag constants are passed in the `inTagName` parameters of `SetControlData` (page 652) and `GetControlData` (page 594) to specify the piece of data in a control that you wish to set or get. You can also pass these constants in the `inTagName` parameter of `GetControlDataSize` (page 596) if you wish to determine the size of variable-length control data. These constants can also be used by custom control definition functions that return the feature bit `kControlSupportsDataAccess` in response to a `kControlMsgGetFeatures` message.

The data that your application sets or obtains can be of various types, dependent upon the control. Therefore, the descriptions of the control data tag constants list the data types for the information that you can set in the `inData` parameter to the `SetControlData` function and that you can get in the `inBuffer` parameter to the `GetControlData` function.

Version Notes
The control data tag constants are available with Appearance Manager 1.0 and later.

User Pane Control Definition ID Constant

```
enum {
    kControlUserPaneProc = 256
};
```

Constants
`kControlUserPaneProc`

Resource ID: 16

User pane. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in `HIContainerViews.h`.

Discussion
When creating a control, your application supplies a control definition ID to one of the Control Manager control-creation functions or to the control resource; see `'CNTL'`. The control definition ID indicates the type of control to create. A control definition ID is an integer that contains the resource ID of a control definition function in its upper 12 bits and a variation code in its lower 4 bits. A control definition ID is derived as follows:

control definition ID = 16 * (`'CDEF'` resource ID) + variation code

A control definition function determines how a control generally looks and behaves. Control definition functions are stored as resources of type 'CDEF'. Various Control Manager functions call a control definition function whenever they need to perform some control-dependent action, such as drawing the control on the screen. For more information on how to create a control definition function, see ControlDefProcPtr (page 677).

A control definition function, in turn, can use a variation code to describe variations of the same basic control. For example, all pop-up arrows share the same basic control definition function, which is stored in a resource of type 'CDEF' and has a resource ID of 12. The standard pop-up arrow is large and points to the right; it has a control definition ID of 192. A variation of this is a large, left-pointing arrow, which has a control definition ID of 193. Still another variation, in which the arrow points up, has a control definition ID of 194.

Your application can use the constant listed here in place of a control definition ID.

useWFont Constants

```
enum {
    useWFont = kControlUsesOwningWindowsFontVariant
};
```

Window Control Definition IDs

```
enum {
    kControlWindowHeaderProc = 336,
    kControlWindowListViewHeaderProc = 337
};
```

Constants

kControlWindowHeaderProc

Resource ID: 21

Window header. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in HIContainerViews.h.

kControlWindowListViewHeaderProc

Resource ID: 21

Window list view header. This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Available in Mac OS X v10.0 and later.

Declared in HIContainerViews.h.

Discussion

When creating a control, your application supplies a control definition ID to one of the Control Manager control-creation functions or to the control resource; see 'CNTL'. The control definition ID indicates the type of control to create. A control definition ID is an integer that contains the resource ID of a control definition function in its upper 12 bits and a variation code in its lower 4 bits. A control definition ID is derived as follows:

control definition ID = 16 * ('CDEF' resource ID) + variation code

A control definition function determines how a control generally looks and behaves. Control definition functions are stored as resources of type 'CDEF'. Various Control Manager functions call a control definition function whenever they need to perform some control-dependent action, such as drawing the control on the screen. For more information on how to create a control definition function, see ControlDefProcPtr (page 677).

A control definition function, in turn, can use a variation code to describe variations of the same basic control. For example, all pop-up arrows share the same basic control definition function, which is stored in a resource of type 'CDEF' and has a resource ID of 12. The standard pop-up arrow is large and points to the right; it has a control definition ID of 192. A variation of this is a large, left-pointing arrow, which has a control definition ID of 193. Still another variation, in which the arrow points up, has a control definition ID of 194.

Your application can use the constants listed here in place of control definition IDs.

Window Control Data List Header Tag Constant

```
enum {
    kControlWindowHeaderIsListHeaderTag = 'islh'
};
```

Constants

kControlWindowHeaderIsListHeaderTag

Set to true if the control is to draw as a list header. Available in Mac OS X v10.3 and later.

Data type returned or set: Boolean

Available in Mac OS X v10.3 and later.

Declared in HIContainerViews.h.

Result Codes

The table below lists the result codes returned by Control Manager functions.

Result Code	Value	Description
controlPropertyInvalid	-5603	You called SetControlProperty, GetControlProperty, or a similar function with an illegal property creator OSType. Available in Mac OS X v10.0 and later.
controlPropertyNotFoundErr	-5604	The property tag and creator combination does not exist for the specified control. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
errMessageNotSupported	-30580	In general, this return value means a control, window, or menu definition does not support the message/event that underlies an API call. For example, if you call GetControlFeatures on a control whose control definition function doesn't support "new messages" (the new group of CDEF messages that came into existence with the Appearance Manager on Mac OS 8), GetControlFeatures will return this error. Available in Mac OS X v10.0 and later.
errDataNotSupported	-30581	Returned from GetControlData and SetControlData if the control doesn't support the tag name and/or part code that is passed in. It can also be returned from other functions that are essentially wrappers around GetControlData and SetControlData (such as SetControlFontStyle). Available in Mac OS X v10.0 and later.
errControlDoesntSupportFocus	-30582	The control you passed to a focusing function (such as SetKeyboardFocus) doesn't support focus. On Mac OS X, you're likely to receive errCouldntSetFocus or eventNotHandledErr i instead. Available in Mac OS X v10.0 and later.
errWindowDoesntSupportFocus	-30583	The specified window does not support focus. Available in Mac OS X v10.0 and later.
errUnknownControl	-30584	This is a variant of (and serves the same purpose as) controlHandleInvalidErr. Various Control Manager functions return this error if one of the specified controls is NULL or otherwise invalid. Available in Mac OS X v10.0 and later.
errCouldntSetFocus	-30585	The focus couldn't be set to a given control or advanced through a hierarchy of controls. This could be because the control doesn't support focusing, the control isn't currently embedded in a window, or you are attempting to advance focus in a window that contains no focusable controls. Available in Mac OS X v10.0 and later.
errNoRootControl	-30586	No root control exists. Some examples of when you might receive this error include: 1) You called GetRootControl before anyone called CreateRootControl for a given non-compositing window. 2) You called a Control Manager function (such as ClearKeyboardFocus or AutoEmbedControl) that can only do its work if there's a root control in the window, yet there's no root control. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
errRootAlreadyExists	-30587	Returned by CreateRootControl if a root control already exists for the specified control. Available in Mac OS X v10.0 and later.
errInvalidPartCode	-30588	The ControlPartCode you passed to a Control Manager function is out of range, invalid, or otherwise unsupported. For example, GetControlRegion returns errInvalidPartCode if you pass kControlNoPart. Available in Mac OS X v10.0 and later.
errControlsAlreadyExist	-30589	You called CreateRootControl after creating one or more non-root controls in a window, which is illegal; if you want an embedding hierarchy on a given window, you must call CreateRootControl before creating any other controls for a given window.This is never returned on Mac OS X, because a root control is created automatically (if it doesn't already exist) the first time any nonroot control is created in a window. Available in Mac OS X v10.0 and later.
errControlIsNotEmbedder	-30590	The control does not support embedding. Returned, for example, by GetIndexedSubControl and EmbedControl. Available in Mac OS X v10.0 and later.
errDataSizeMismatch	-30591	You called GetControlData or SetControlData with a buffer whose size does not match the size of the data you are attempting to get or set. Available in Mac OS X v10.0 and later.
errControlHiddenOrDisabled	-30592	You called TrackControl, HandleControlClick, or a similar mouse tracking function, on a control that is invisible or disabled. You cannot track controls that are invisible or disabled. Available in Mac OS X v10.0 and later.
errWindowRegionCodeInvalid	-30593	The window region code is invalid. Available in Mac OS X v10.0 and later.
errCantEmbedIntoSelf	-30594	You called EmbedControl (or a similar function) with the same control in the parent and child parameters. In other words, you cannot embed a control into itself. Available in Mac OS X v10.0 and later.
errCantEmbedRoot	-30595	You attempted to embed the root control in another control. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
errItemNotControl	-30596	You called GetDialogItemAsControl on a dialog item (such as a kHelpDialogItem) that is not represented by a control. Available in Mac OS X v10.0 and later.
controlInvalidDataVersionErr	-30597	You called GetControlData or SetControlData with a buffer that represents a versioned structure, but the version is unsupported by the control definition. This can happen with the Tabs control and the kControlTabInfoTag. Available in Mac OS X v10.0 and later.
controlHandleInvalidErr	-30599	The control reference passed in was invalid. Available in Mac OS X v10.0 and later.

Dialog Manager Reference

Framework: Carbon/Carbon.h

Declared in Dialogs.h

Overview

Your application can use the Dialog Manager to alert users to unusual situations and to solicit information from users. For example, in some situations your application might not be able to carry out a command normally, and in other situations the user must specify multiple parameters before your application can execute a command. For circumstances like these, the Macintosh user interface includes these two features:

- alerts–including alert sounds and alert boxes–which warn the user whenever an unusual or potentially undesirable situation occurs within your application
- dialog boxes, which allow the user to provide additional information or to modify settings before your application carries out a command

Virtually all applications need to implement alerts and dialog boxes. To avoid needless development effort, use the Dialog Manager to implement alerts and to create most dialog boxes. It is possible, however–and sometimes desirable–to bypass the Dialog Manager and instead use Window Manager, Control Manager, QuickDraw, and Event Manager routines to create or respond to events in complex dialog boxes.

Carbon supports the majority of the Dialog Manager. However, your application must access Dialog Manager data structures only through the supplied accessor functions. Furthermore, your application must use the functions provided for creating and disposing of Dialog Manager data structures.

Functions by Task

Creating Alert Boxes

`StandardAlert` (page 887)
Displays a standard alert box.

`Alert` (page 834)
Displays an alert box and/or plays an alert sound.

`StopAlert` (page 889)
Displays an alert box with a stop icon and/or plays an alert sound.

`NoteAlert` (page 874)
Displays an alert box with a note icon and/or plays an alert sound.

`CautionAlert` (page 839)
Displays an alert box with a caution icon and/or plays an alert sound.

`GetAlertStage` (page 850)
Determines the stage of the last occurrence of an alert.

`ResetAlertStage` (page 876)
Resets the current alert stage to the first alert stage.

Creating and Disposing of Dialog Boxes

`GetNewDialog` (page 858)
Creates a dialog box from a resource-based description.

`NewFeaturesDialog` (page 871)
Creates a dialog box from information passed in memory.

`NewDialog` (page 870)
Creates a dialog box from information passed in memory.

`NewColorDialog` (page 868)
Creates a dialog box from information passed in memory.

`CloseDialog` (page 840)
Dismisses a dialog box without disposing of the dialog structure.

`DisposeDialog` (page 847)
Dismisses a dialog box for which the Dialog Manager supplies memory and disposes of the dialog structure.

Displaying Dialog Boxes and Items

`DrawDialog` (page 849)
Draws the entire contents of a specified dialog box.

`HideDialogItem` (page 861)
Makes an item in a dialog box invisible.

`ShowDialogItem` (page 885)
Redisplays an item that has been hidden by HideDialogItem.

Filtering Dialog Box Events

`GetModalDialogEventMask` (page 857)
Obtains the events to be received by the `ModalDialog` function.

`SetModalDialogEventMask` (page 884)
Specifies the events to be received by the `ModalDialog` function.

Handling Events in Dialog Boxes

`ModalDialog` (page 865)
Handles events while your application displays a modal or movable modal dialog box.

`IsDialogEvent` (page 863)
Determines whether a modeless dialog box or a movable modal dialog box is active when an event occurs.

`DialogSelect` (page 845)
Handles most of the events inside the dialog box after you have determined that an event related to an active modeless dialog box or an active movable modal dialog box has occurred.

`UpdateDialog` (page 890)
Redraws the update region of a specified dialog box.

Handling Text in Alert and Dialog Boxes

`ParamText` (page 875)
Replaces the text strings in the static text items of your alert or dialog boxes while your application is running.

`SetDialogItemText` (page 882)
Sets the text string for static text and editable text fields.

`GetDialogItemText` (page 854)
Obtains the text string contained in an edit text or a static text item.

`SelectDialogItemText` (page 877)
Selects and highlights text contained in an edit text item.

`DialogCut` (page 844)
Handles the Cut editing command when a dialog box containing an edit text item is active.

`DialogCopy` (page 844)
Handles the Copy editing command when a dialog box containing an edit text item is active.

`DialogPaste` (page 845)
Handles the Paste editing command when a dialog box containing an edit text item is active.

`DialogDelete` (page 845)
Handles the Delete editing command when a dialog box containing an edit text item is active.

Initializing the Dialog Manager

`SetDialogFont` (page 880)
Sets the font used in static and edit text items.

Manipulating Items in Dialog Boxes and Alert Boxes

`SetDialogCancelItem` (page 878)
Sets the cancel item for a dialog box.

`GetDialogCancelItem` (page 851)
Returns the item number of the cancel item previously set with `SetDialogCancelItem`.

`SetDialogDefaultItem` (page 879)
Sets the default item for a dialog box and draws an appropriate border around the default item.

`GetDialogDefaultItem` (page 851)
Returns the item number of the default item currently set for the standard filter function.

`GetDialogItemAsControl` (page 853)
Obtains the control handle for a dialog item in an embedding hierarchy.

`GetDialogItem` (page 852)
Obtains a handle to a dialog item.

`SetDialogItem` (page 881)
Sets or changes information for a dialog item.

`GetDialogKeyboardFocusItem` (page 854)
Returns the item number of the editable text item in a dialog box that has keyboard focus.

`SetDialogTracksCursor` (page 883)
Determines whether the Dialog Manager tracks the cursor's movements and changes the cursor to an I-beam whenever it is over an edit dialog box.

`FindDialogItem` (page 849)
Determines the item number of an item at a particular location in a dialog box.

`MoveDialogItem` (page 867)
Moves a dialog item to a specified location in a window.

`SizeDialogItem` (page 886)
Sizes a dialog item.

`AutoSizeDialog` (page 838)
Automatically resizes static text fields and their dialog boxes to accommodate changed static text.

`AppendDialogItemList` (page 835)
Adds items to an existing dialog box while your program is running.

`AppendDITL` (page 836)
Adds items to an existing dialog box while your application is running.

`ShortenDITL` (page 885)
Removes items from an existing dialog box while your application is running.

`CountDITL` (page 841)
Determines the number of items in a dialog box.

Simulating User Responses in Dialog Boxes

`GetDialogTimeout` (page 856)
Obtains the original countdown duration, the time remaining, and the item selection to be simulated for a specified modal dialog box.

`SetDialogTimeout` (page 882)
Simulates an item selection in a modal dialog box after a specified amount of time elapses.

Using the Standard Filter Function

`StdFilterProc` (page 888)
Handles standard event filtering for a dialog box.

`GetStdFilterProc` (page 860)
Returns a pointer to the standard filter function.

Miscellaneous

`CloseStandardSheet` (page 841)

`CreateStandardAlert` (page 842)
Creates an alert containing standard elements and using standard formatting rules.

`CreateStandardSheet` (page 842)
Creates an alert containing standard elements and using standard formatting rules, and prepares it to be displayed as a sheet.

`DisposeModalFilterUPP` (page 848)

`DisposeModalFilterYDUPP` (page 848)

`DisposeUserItemUPP` (page 849)

`GetDialogFromWindow` (page 852)

`GetDialogPort` (page 855)

`GetDialogTextEditHandle` (page 856)

`GetDialogWindow` (page 857)

`GetParamText` (page 859)

`GetStandardAlertDefaultParams` (page 860)
Fills out an `AlertStdCFStringAlertParamRec` with default values: - not movable - no help button - default button with title "OK" - no cancel or other buttons.

`InsertDialogItem` (page 862)

`InvokeModalFilterUPP` (page 862)

`InvokeModalFilterYDUPP` (page 863)

`InvokeUserItemUPP` (page 863)

`NewModalFilterUPP` (page 873)

`NewModalFilterYDUPP` (page 873)

`NewUserItemUPP` (page 873)

`RemoveDialogItems` (page 876)

`RunStandardAlert` (page 877)
Shows and runs a standard alert using a modal dialog loop.

`SetPortDialogPort` (page 884)

Functions

Alert

Displays an alert box and/or plays an alert sound.

```
DialogItemIndex Alert (
   SInt16 alertID,
   ModalFilterUPP modalFilter
);
```

Parameters

alertID
The resource ID of an alert resource and extended alert resource. If the alert resource is missing, the Dialog Manager returns to your application without creating the requested alert. See `'alrx'` for a description of the extended alert resource.

modalFilter
A universal procedure pointer for a filter function that responds to events not handled by the `ModalDialog` (page 865) function. If you set this parameter to `null`, the Dialog Manager uses the standard event filter function.

Return Value

If no alert box is to be drawn at the current alert stage or the `'ALRT'` resource is not found, `Alert` returns –1 otherwise, it creates and displays the alert box and returns the item number of the control selected by the user see "Alert Button Constants" (page 904). See the description of the `DialogItemIndex` data type.

Discussion

The `Alert` function displays an alert box or, if appropriate for the alert stage, plays an alert sound instead of or in addition to displaying the alert box. The `Alert` function creates the alert defined in the specified alert resource and its corresponding extended alert resource. The function calls the current alert sound function and passes it the sound number specified in the alert resource for the current alert stage. If no alert box is to be drawn at this stage, `Alert` returns –1 otherwise, it uses the `NewDialog` function to create and display the alert box. The default system window colors are used unless your application provides an alert color table resource with the same resource ID as the alert resource. The `Alert` function uses the `ModalDialog` (page 865) function to get and handle most events for you.

The `Alert` function does not display a default icon in the upper-left corner of the alert box you can leave this area blank, or you can specify your own icon in the alert's item list resource, which in turn is specified in the alert resource.

The `Alert` function continues calling `ModalDialog` until the user selects an enabled control (typically a button), at which time the `Alert` function removes the alert box from the screen and returns the item number of the selected control. Your application then responds as appropriate when the user clicks this item.

Your application should never draw its own default rings. Prior to Mac OS 8, the `Alert` function would only redraw the default button ring once and never redraw it on an update event. However, when Appearance is available, default rings do redraw when you call `Alert`.

See also the functions `NoteAlert` (page 874) , `CautionAlert` (page 839) , and `StopAlert` (page 889).

Special Considerations

If you need to display an alert box while your application is running in the background or is otherwise invisible to the user, call `AEInteractWithUser`

The Dialog Manager uses the system alert sound as the error sound unless you change it by calling the `ErrorSound` function .

Version Notes

This function was changed with Appearance Manager 1.0 to support the extended alert (`'alrx'`) resource.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Dialogs.h`

AppendDialogItemList

Adds items to an existing dialog box while your program is running.

```
OSErr AppendDialogItemList (
   DialogRef dialog,
   SInt16 ditlID,
   DITLMethod method
);
```

Parameters

dialog

A pointer to the dialog box to which the items in the item list resource specified in the `ditlID` parameter are to be appended.

ditlID

The resource ID of the item list resource whose items are to be appended to the dialog box specified in the `dialog` parameter.

method

The manner in which the new items are to be displayed in the dialog box.

If you use the `overlayDITL` constant, `AppendDialogItemList` superimposes the appended items over the dialog box by interpreting the coordinates of the display rectangles for the appended items (as specified in their item list resource) as local coordinates within the dialog box.

If you use the `appendDITLRight` constant, `AppendDialogItemList` appends the items to the right of the dialog box by positioning the display rectangles of the appended items relative to the upper-right coordinate of the dialog box. The `AppendDialogItemList` function automatically expands the dialog box to accommodate the new dialog items.

If you use the `appendDITLBottom` constant, `AppendDialogItemList` appends the items to the bottom of the dialog box by positioning the display rectangles of the appended items relative to the lower-left coordinate of the dialog box. The `AppendDialogItemList` function automatically expands the dialog box to accommodate the new dialog items.

You can append a list of items relative to an existing item by passing a negative number. The absolute value of this number is interpreted as the item in the dialog box relative to which the new items are to be positioned. For example, if you pass -2, the display rectangles of the appended items are offset relative to the upper-left corner of item number 2 in the dialog box.

Return Value
A result code. See "Dialog Manager Result Codes" (page 916).

Discussion
To be Appearance-compliant, your program should use the `AppendDialogItemList` function rather than the `AppendDITL` function. Unlike `AppendDITL`, the `AppendDialogItemList` function takes a `'DITL'` resource ID instead of a handle as the parameter describing the dialog item list to be appended, and it properly appends entries from a dialog font table (`'dftb'`) resource, if there is a `'dftb'` resource with the same resource ID as the `'DITL'` resource.

The `AppendDialogItemList` function adds the items in the item list resource specified in the parameter `ditlID` to the items of a dialog box. This is especially useful if several dialog boxes share a single item list resource, because you can use `AppendDialogItemList` to add items that are appropriate for individual dialog boxes. Your application can use the Resource Manager function `GetResource` to get a handle to the item list resource whose items you wish to add.

You typically create an invisible dialog box, call the `AppendDialogItemList` function, then make the dialog box visible by using the Window Manager function `ShowWindow`.

Version Notes
This function is available with Appearance Manager 1.0 and later.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

AppendDITL

Adds items to an existing dialog box while your application is running.

```
void AppendDITL (
   DialogRef theDialog,
   Handle theHandle,
   DITLMethod method
);
```

Parameters

theDialog

A pointer to a dialog structure. This is the dialog structure to which you will add the item list resource specified in the parameter theHandle.

theHandle

A handle to the item list resource whose items you want to append to the dialog box. To avoid item number conflicts, `AppendDITL` assigns new numbers to the items you are adding. For example, if you have a dialog with item numbers 1-5, and you use `AppendDITL` to add a `'DITL'` resource containing item numbers 1-3, those become item numbers 6-8 in the dialog.

method

The manner in which you want the new items to be displayed in the existing dialog box. You can pass a negative value to offset the appended items from a particular item in the existing dialog box. You can also pass one of the values defined by the `DITLMethod` constant. See "Dialog Item List Display Constants" (page 913) for possible values.

Discussion

The `AppendDITL` function adds the items specified in the `theHandle` parameter to the items of a dialog box (handle-based). This function is especially useful if several dialog boxes share a single item list resource, because you can use `AppendDITL` to add items that are appropriate for individual dialog boxes. Your application can use the Resource Manager function `GetResource` to get a handle to the item list resource whose items you wish to add.

In the parameter `method`, you specify how to append the new items, as follows:

- If you use the `overlayDITL` constant, `AppendDITL` superimposes the appended items over the dialog box. That is, `AppendDITL` interprets the coordinates of the display rectangles for the appended items (as specified in their item list resource) as local coordinates within the dialog box.
- If you use the `appendDITLRight` constant, `AppendDITL` appends the items to the right of the dialog box by positioning the display rectangles of the appended items relative to the upper-right coordinate of the dialog box. The `AppendDITL` function automatically expands the dialog box to accommodate the new dialog items.
- If you use the `appendDITLBottom` constant, `AppendDITL` appends the items to the bottom of the dialog box by positioning the display rectangles of the appended items relative to the lower-left coordinate of the dialog box. The `AppendDITL` function automatically expands the dialog box to accommodate the new dialog items.
- You can also append a list of items relative to an existing item by passing a negative number in the parameter `method`. The absolute value of this number is interpreted as the item in the dialog box relative to which the new items are to be positioned. For example, if you pass –2, the display rectangles of the appended items are offset relative to the upper-left corner of item number 2 in the dialog box.

You typically create an invisible dialog box, call the `AppendDITL` function, then make the dialog box visible by using the Window Manager function `ShowWindow`.

Special Considerations

The `AppendDITL` function modifies the contents of the dialog box (for instance, by enlarging it). To use an unmodified version of the dialog box at a later time, your application should use the Resource Manager function `ReleaseResource` to release the memory occupied by the appended item list resource. Otherwise, if your application calls `AppendDITL` to add items to that dialog box again, the dialog box remains modified by your previous call—for example, it will still be longer at the bottom if you previously used the `appendDITLBottom` constant.

Before calling `AppendDITL`, you should make sure that it is available by using the `Gestalt` function with the `gestaltDITLExtAttr` selector. Test the bit indicated by the `gestaltDITLExtPresent` constant in the `response` parameter. If the bit is set, then `AppendDITL` is available.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Dialogs.h`

AutoSizeDialog

Automatically resizes static text fields and their dialog boxes to accommodate changed static text.

```
OSErr AutoSizeDialog (
   DialogRef inDialog
);
```

Parameters

`inDialog`

A pointer to a dialog box.

Return Value

A result code. See "Dialog Manager Result Codes" (page 916).

Discussion

The `AutoSizeDialog` function is useful in situations such as localization, where the size of a static text field (and the dialog box that contains it) may need to be altered to accommodate a change in the size of the static text.

For each static text item `AutoSizeDialog` finds in the item list resource, it adjusts the static text field and the bottom of the dialog box window to accommodate the text. Any items below a static text field are moved down. If the dialog box is visible when this function is called, it is hidden, resized, and then shown. If the dialog box has enough room to show the text as is, no resizing is done.

Note that the `AutoSizeDialog` function does not process update events for your dialog box, so your program must call the `DrawDialog` function or the `DialogSelect` function to process the update event generated from showing the window.

Version Notes

This function is available with Appearance Manager 1.0 and later.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
Dialogs.h

CautionAlert

Displays an alert box with a caution icon and/or plays an alert sound.

```
DialogItemIndex CautionAlert (
   SInt16 alertID,
   ModalFilterUPP modalFilter
);
```

Parameters

alertID

The resource ID of an alert resource and extended alert resource. If the alert resource is missing, the Dialog Manager returns to your application without creating the requested alert. See 'alrx' for a description of the extended alert resource.

modalFilter

A universal procedure pointer for a filter function that responds to events not handled by the ModalDialog (page 865) function. If you set this parameter to null, the Dialog Manager uses the standard event filter function.

Return Value
If no alert box is to be drawn at the current alert stage, CautionAlert returns –1 otherwise, it uses NewDialog to create and display the alert box and returns the item hit; see "Alert Button Constants" (page 904). See the description of the DialogItemIndex data type.

Discussion
Displays an alert box with a caution icon in its upper-left corner or, if appropriate for the alert stage, to play an alert sound instead of or in addition to displaying the alert box.

The CautionAlert function is the same as the Alert (page 834) function except that, before drawing the items in the alert box, CautionAlert draws the caution icon in the upper-left corner. The caution icon has resource ID 2, which you can also specify with the constant kCautionIcon. By default, the Dialog Manager uses the standard caution icon from the System file. You can change this icon by providing your own 'ICON' resource with resource ID 2.

Use a caution alert to alert the user of an operation that may have undesirable results if it's allowed to continue. Give the user the choice of continuing the action (by clicking an OK button) or stopping it (by clicking a Cancel button).

Your application should never draw its own default rings or alert icons. Prior to Mac OS 8, the CautionAlert function would only redraw the alert icon and default button ring once and never redraw them on an update event. However, when Appearance is available, alert icons and default rings do redraw when you call CautionAlert.

See also the functions NoteAlert (page 874) and StopAlert (page 889).

Special Considerations

Version Notes
This function was changed with Appearance Manager 1.0 to support the extended alert ('alrx') resource.

Availability
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
`Dialogs.h`

CloseDialog

Dismisses a dialog box without disposing of the dialog structure.

```
void CloseDialog (
    DialogRef theDialog
);
```

Parameters

theDialog
A pointer to a dialog structure.

Return Value

Discussion
The `CloseDialog` function removes a dialog box from the screen and deletes it from the window list. The `CloseDialog` function releases the memory occupied by

- the data structures associated with the dialog box (such as its structure, content, and update regions)
- all the items in the dialog box (except for pictures and icons, which might be shared by other resources) and any data structures associated with them

Generally, you should provide memory for the dialog structure of modeless dialog boxes when you create them. (You can let the Dialog Manager provide memory for modal and movable modal dialog boxes.) You should then use `CloseDialog` to close a modeless dialog box when the user clicks the close box or chooses Close from the File menu.

Because `CloseDialog` does not dispose of the dialog resource or the item list resource, it is important to make these resources purgeable. Unlike `GetNewDialog` (page 858) , `NewColorDialog` (page 868) does not use a copy of the item list resource. Thus, if you use `NewColorDialog` to create a dialog box, you may want to use `CloseDialog` to keep the item list resource in memory even if you didn't supply a pointer to the memory.

Carbon Porting Notes

The `CloseDialog` function is not supported because developers do not allocate their own memory for dialog boxes in Carbon. Use the `DisposeDialog` function to dismiss a dialog box instead.

Declared In
`Dialogs.h`

CloseStandardSheet

```
OSStatus CloseStandardSheet (
   DialogRef inSheet,
   UInt32 inResultCommand
);
```

Parameters

inSheet

inResultCommand

Return Value

A result code. See "Dialog Manager Result Codes" (page 916).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Dialogs.h`

CountDITL

Determines the number of items in a dialog box.

```
DialogItemIndex CountDITL (
   DialogRef theDialog
);
```

Parameters

theDialog

A pointer to a dialog structure.

Return Value

The number of current items in a dialog box. See the description of the `DialogItemIndex` data type.

Discussion

You typically use `CountDITL` in conjunction with `ShortenDITL` (page 885) to remove items from a dialog box.

Special Considerations

Before calling `CountDITL`, you should make sure that it is available by using the `Gestalt` function with the `gestaltDITLExtAttr` selector. Test the bit indicated by the `gestaltDITLExtPresent` constant in the `response` parameter. If the bit is set, then `CountDITL` is available.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Dialogs.h`

CreateStandardAlert

Creates an alert containing standard elements and using standard formatting rules.

```
OSStatus CreateStandardAlert (
   AlertType alertType,
   CFStringRef error,
   CFStringRef explanation,
   const AlertStdCFStringAlertParamRec *param,
   DialogRef *outAlert
);
```

Parameters

alertType
The type of alert to create. For a list of possible values, see "Alert Type Constants" (page 907).

error
The error string to display.

explanation
The explanation string to display. May be `NULL` or empty to display no explanation.

param
The parameter block describing how to create the alert. May be `NULL`.

outAlert
A pointer to a variable that, on return, refers to the new alert.

Return Value

A result code. See "Dialog Manager Result Codes" (page 916).

Discussion

This function should be used in conjunction with `RunStandardAlert` (page 877). After `CreateStandardAlert` returns, the alert is still invisible. `RunStandardAlert` shows the alert and runs a modal dialog loop to process events in the alert.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

QTCarbonShell

Declared In

`Dialogs.h`

CreateStandardSheet

Creates an alert containing standard elements and using standard formatting rules, and prepares it to be displayed as a sheet.

```
OSStatus CreateStandardSheet (
   AlertType alertType,
   CFStringRef error,
   CFStringRef explanation,
   const AlertStdCFStringAlertParamRec *param,
   EventTargetRef notifyTarget,
   DialogRef *outSheet
);
```

Parameters

alertType

The type of alert to create. For a list of possible values, see "Alert Type Constants" (page 907).

error

The error string to display.

explanation

The explanation string to display. May be `NULL` or empty to display no explanation.

param

The parameter block describing how to create the alert. May be `NULL`.

notifyTarget

The event target to be notified when the user dismisses the sheet. The caller should install an event handler on this target for the `kEventProcessCommand` event. May be `NULL` if the caller does not need the command event to be sent to any target. For more information, see the Discussion below.

outSheet

A pointer to a variable that, on return, refers to the new alert.

Return Value

A result code. See "Dialog Manager Result Codes" (page 916).

Discussion

This function should be used in conjunction with `ShowSheetWindow` (page 1958). After `CreateStandardSheet` returns, the alert is still invisible. `ShowSheetWindow` will show the alert as a sheet and then return. Events in the sheet are handled asynchronously; the application should be prepared for the sheet window to be part of its window list while running its own event loop.

When a button in the sheet is pressed, the event target passed to `CreateStandardSheet` will receive a command event with one of the following commands: `kHICommandOK`, `kHICommandCancel`, or `kHICommandOther`. The system takes care of closing the sheet and releasing the alert. Therefore after using `ShowSheetWindow` (page 1958), you do not need to call `HideSheetWindow` (page 1872) or `DisposeDialog` (page 847).

Typically, the event target you pass in the *notifyTarget* parameter is the parent window of the sheet. A standard practice is to install a command event handler on the parent window just before showing the sheet window, and to remove the handler from the parent window after the sheet has been closed.

It is also possible to install a handler on the sheet window itself, in which case you would pass `NULL` in the *notifyTarget* parameter, since the command event is automatically sent to the sheet window already. If you install a handler on the sheet itself, make sure to return `eventNotHandledErr` from your handler, because `CreateStandardSheet` installs its own handler on the sheet and that handler must be allowed to run to close the sheet window and release the alert.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code
QTCarbonShell

Declared In
`Dialogs.h`

DialogCopy

Handles the Copy editing command when a dialog box containing an edit text item is active.

```
void DialogCopy (
   DialogRef theDialog
);
```

Parameters

`theDialog`

A pointer to a dialog structure.

Discussion
The `DialogCopy` function checks whether the dialog box has any edit text items and, if so, applies the TextEdit function `TECopy` to the selected text. Your application should test whether a dialog box is the frontmost window when handling mouse-down events in the Edit menu and then call this function when appropriate.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

DialogCut

Handles the Cut editing command when a dialog box containing an edit text item is active.

```
void DialogCut (
   DialogRef theDialog
);
```

Parameters

`theDialog`

On input, a pointer to a dialog structure.

Discussion
The `DialogCut` function checks whether the dialog box has any edit text items and, if so, applies the TextEdit function `TECut` to the selected text. Your application should test whether a dialog box is the frontmost window when handling mouse-down events in the Edit menu and then call this function when appropriate.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

DialogDelete

Handles the Delete editing command when a dialog box containing an edit text item is active.

```
void DialogDelete (
   DialogRef theDialog
);
```

Parameters

theDialog

A pointer to a dialog structure.

Discussion

The `DialogDelete` function checks whether the dialog box has any edit text items and, if so, applies the TextEdit function `TEDelete` to the selected text. Your application should test whether a dialog box is the frontmost window when handling mouse-down events in the Edit menu and then call this function when appropriate.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Dialogs.h`

DialogPaste

Handles the Paste editing command when a dialog box containing an edit text item is active.

```
void DialogPaste (
   DialogRef theDialog
);
```

Parameters

theDialog

On input, a pointer to a dialog structure.

Discussion

The `DialogPaste` function checks whether the dialog box has any edit text items and, if so, applies the TextEdit function `TEPaste` to the selected edit text item. Your application should test whether a dialog box is the frontmost window when handling mouse-down events in the Edit menu and then call this function when appropriate.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Dialogs.h`

DialogSelect

Handles most of the events inside the dialog box after you have determined that an event related to an active modeless dialog box or an active movable modal dialog box has occurred.

```
Boolean DialogSelect (
   const EventRecord *theEvent,
   DialogRef *theDialog,
   DialogItemIndex *itemHit
);
```

Parameters

theEvent

A pointer to an event structure returned by an Event Manager function such as `WaitNextEvent`.

theDialog

A pointer to a dialog structure for the dialog box where the event occurred.

itemHit

A pointer to a short integer. `DialogSelect` returns a number corresponding to the position of an item within the item list resource of the active dialog box.

Return Value

A Boolean value. If the event is an activate or update event for a dialog box, `DialogSelect` activates or updates it and returns `false`. If the event involves an enabled item, `DialogSelect` returns a function result of `true`.

Discussion

The `DialogSelect` function handles most of the events relating to a dialog box. Through its `itemHit` parameter, it returns the item number of the item selected by the user. Through the parameter `theDialog`, it returns a pointer to the dialog structure for the dialog box where the event occurred. In all other cases, the `DialogSelect` function returns `false`. When `DialogSelect` returns `true`, do whatever is appropriate as a response to the event involving that item in that particular dialog box; when it returns `false`, do nothing.

Generally, only controls should be enabled in a dialog box; therefore your application should normally respond only when `DialogSelect` returns `true` after the user clicks an enabled control, such as the OK button.

The `DialogSelect` function first obtains a pointer to the window containing the event. For update and activate events, the event structure contains the window pointer. For other types of events, `DialogSelect` calls the Window Manager function `FrontWindow`. The Dialog Manager then makes this window the current graphics port by calling the QuickDraw function `SetPort`. Then `DialogSelect` prepares to handle the event by setting up text information if there are any edit text items in the active dialog box.

When an item is a control defined in a control resource, the rectangle added to the update region is the rectangle defined in the control resource, not the display rectangle defined in the item list resource.

The `DialogSelect` function handles the event as follows:

- In response to an activate or update event for the dialog box, `DialogSelect` activates or updates its window and returns false.
- If a key-down event or an auto-key event occurs and there's an edit text item in the dialog box, `DialogSelect` uses `TextEdit` to handle text entry and editing, and `DialogSelect` returns true for a function result. Through its `itemHit` parameter, `DialogSelect` returns the item number.
- If a key-down event or an auto-key event occurs and there's no edit text item in the dialog box, `DialogSelect` returns `false`.
- If the user presses the mouse button while the cursor is in an edit text item, `DialogSelect` responds to the mouse activity as appropriate—that is, either by displaying an insertion point or by selecting text. If the edit text item is disabled, `DialogSelect` returns `false`. If the edit text item is enabled,

`DialogSelect` returns `true` and through its `itemHit` parameter returns the item number. Normally, edit text items are disabled, and you use the `GetDialogItemText` function to read the information in the items only after the OK button is clicked.

- If the user presses the mouse button while the cursor is in a control, `DialogSelect` tracks the control. If the user releases the mouse button while the cursor is in an enabled control, `DialogSelect` returns `true` for a function result and through its `itemHit` parameter returns the control's item number. Your application should respond appropriately—for example, by performing a command after the user clicks the OK button.
- If the user presses the mouse button while the cursor is in any other enabled item in the dialog box, `DialogSelect` returns `true` for a function result and through its `itemHit` parameter returns the item's number. Generally, only controls should be enabled. If your application creates a complex control—such as one that measures how far a dial is moved—your application must handle mouse events in that item before passing the event to `DialogSelect`.
- If the user presses the mouse button while the cursor is in a disabled item, or if it is in no item, or if any other event occurs, `DialogSelect` does nothing.
- If the event isn't one that `DialogSelect` specifically checks for (if it's a null event, for example), and if there's an edit text item in the dialog box, `DialogSelect` calls the TextEdit function `TEIdle` to make the insertion point blink.

Special Considerations

Because `DialogSelect` handles only mouse-down events in a dialog box and key-down events in a dialog box's edit text items, you should handle other events as appropriate before passing them to `DialogSelect`. Likewise, when `DialogSelect` calls the Control Manager function `TrackControl` , it does not allow you to specify any action function necessary for anything more complex than a button, radio button, or checkbox. If you need a more complex control (for example, one that measures how long the user holds down the mouse button or how far the user has moved an indicator), you can create your own control or a picture or an application-defined item that draws a control-like object in your dialog box. You must then test for and respond to those events yourself.

Within dialog boxes, use the functions `DialogCut` (page 844), `DialogCopy` (page 844), `DialogPaste` (page 845), and `DialogDelete` (page 845) to support Cut, Copy, Paste, and Clear commands in edit text boxes.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Dialogs.h`

DisposeDialog

Dismisses a dialog box for which the Dialog Manager supplies memory and disposes of the dialog structure.

```
void DisposeDialog (
   DialogRef theDialog
);
```

Parameters

`theDialog`

A pointer to a dialog structure.

Return Value

Discussion
The `DisposeDialog` function calls `CloseDialog` (page 840) and, in addition, releases the memory occupied by the dialog box's item list resource and the dialog structure. Call `DisposeDialog` when you're done with a dialog box if you pass `null` in the `dStorage` parameter to `GetNewDialog` (page 858) , `NewColorDialog` (page 868) , or `NewDialog` (page 870).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
ictbSample

Declared In
`Dialogs.h`

DisposeModalFilterUPP

```
void DisposeModalFilterUPP (
   ModalFilterUPP userUPP
);
```

Parameters
userUPP

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Dialogs.h`

DisposeModalFilterYDUPP

```
void DisposeModalFilterYDUPP (
   ModalFilterYDUPP userUPP
);
```

Parameters
userUPP

Carbon Porting Notes

This function is supported in Carbon because several QuickTime routines require it.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Dialogs.h`

DisposeUserItemUPP

```
void DisposeUserItemUPP (
   UserItemUPP userUPP
);
```

Parameters

userUPP

Availability

Available in Mac OS X v10.0 and later.

Declared In

Dialogs.h

DrawDialog

Draws the entire contents of a specified dialog box.

```
void DrawDialog (
   DialogRef theDialog
);
```

Parameters

theDialog

A pointer to a dialog structure.

Return Value

Discussion

The DrawDialog function draws all dialog items, calls the Control Manager function DrawOneControl to draw all controls, and calls the TextEdit function TEUpdate to update all static and edit text items and to draw their display rectangles. The DrawDialog function also calls the application-defined items' draw functions if the items' rectangles are within the update region.

DialogSelect (page 845) , ModalDialog (page 865) , Alert (page 834) , StopAlert (page 889) , NoteAlert (page 874) , and CautionAlert (page 839) use DrawDialog automatically. If you use GetNewDialog (page 858) to create a dialog box but don't use any of these other Dialog Manager functions when handling events in the dialog box, you can use DrawDialog to redraw the contents of the dialog box when it's visible. If the dialog box is invisible, first use the Window Manager function ShowWindow and then use DrawDialog.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

Dialogs.h

FindDialogItem

Determines the item number of an item at a particular location in a dialog box.

```
DialogItemIndexZeroBased FindDialogItem (
   DialogRef theDialog,
   Point thePt
);
```

Parameters

theDialog

A pointer to a dialog structure.

thePt

The point (in local coordinates) where the mouse-down event occurred.

Return Value

When an embedding hierarchy is established, the `FindDialogItem` function returns the deepest control selected by the user corresponding to the point specified in the `thePt` parameter. When an embedding hierarchy does not exist, `FindDialogItem` performs a linear search of the item list resource and returns a number corresponding to the hit item's position in the item list resource. For example, it returns 0 for the first item in the item list, 1 for the second, and 2 for the third. If the mouse is not over a dialog item, `FindDialogItem` returns –1. See the description of the `DialogItemIndexZeroBased` data type.

Discussion

The function `FindDialogItem` is useful for changing the cursor when the user moves the cursor over a particular item.

To get the proper item number before calling the `GetDialogItem` (page 852) function or the `SetDialogItem` (page 881) function, add 1 to the result of `FindDialogItem`, as shown here:

```
theItem = FindDialogItem(theDialog, thePoint) + 1;
```

Note that `FindDialogItem` returns the item number of disabled items as well as enabled items.

Version Notes

This function was changed with Appearance Manager 1.0 to support embedding hierarchies.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Dialogs.h`

GetAlertStage

Determines the stage of the last occurrence of an alert.

```
SInt16 GetAlertStage (
   void
);
```

Parameters

Return Value

A number from 0 to 3 as the stage of the last occurrence of an alert.

Discussion

You can use the `GetAlertStage` function to ensure that your application deactivates the active window only if an alert box is to be displayed at that stage.

Carbon Porting Notes

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

GetDialogCancelItem

Returns the item number of the cancel item previously set with `SetDialogCancelItem`.

```
SInt16 GetDialogCancelItem (
   DialogRef dialog
);
```

Parameters

dialog
On input, a pointer to the dialog structure for the dialog box whose cancel item you want to get.

Return Value
The item number of the cancel item previously set with the `SetDialogCancelItem` (page 878) function.

Discussion
If you don't explicitly call `GetDialogCanceltItem`, the standard filter function treats item 2 as the cancel item.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

GetDialogDefaultItem

Returns the item number of the default item currently set for the standard filter function.

```
SInt16 GetDialogDefaultItem (
   DialogRef dialog
);
```

Parameters

dialog
On input, a pointer to the dialog structure for the dialog box whose default item you want to get.

Return Value
The item number of the default item currently set for the standard filter function.

Discussion
If you don't explicitly call `GetDialogDefaultItem`, the standard filter function treats item 1 as the default item.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

GetDialogFromWindow

```
DialogRef GetDialogFromWindow (
   WindowRef window
);
```

Parameters

window

Return Value
See the description of the `DialogRef` data type.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

GetDialogItem

Obtains a handle to a dialog item.

```
void GetDialogItem (
   DialogRef theDialog,
   DialogItemIndex itemNo,
   DialogItemType *itemType,
   Handle *item,
   Rect *box
);
```

Parameters

theDialog
A pointer to the dialog box to examine.

itemNo
The position of the item in the dialog box's item list resource use `FindDialogItem` (page 849) to determine this value.

itemType
A pointer to a short value. On return, the value identifies the item type of the dialog item requested in the `itemNo` parameter.

item
A pointer to an item handle. On return the handle refers to the item specified in the `itemNo` parameter or, for application-defined draw functions, a pointer (coerced to a handle) to the draw function.

box

A pointer to a rectangle. On return, the rectangle specifies the display rectangle (described in coordinates local to the dialog box), for the item specified in the `itemNo` parameter.

Return Value

Discussion

The `GetDialogItem` function produces the item type, a handle to the item (or, for application-defined draw functions, the function pointer), and the display rectangle for a specified item in an item list resource. When a control hierarchy is present in the dialog box, `GetDialogItem` gets the appropriate information (for example, a text handle) from the controls. If you wish to get a control handle for a dialog item in an embedding hierarchy, see `GetDialogItemAsControl` (page 853).

You should call `GetDialogItem` before calling functions such as `SetDialogItemText` (page 882) that need a handle to a dialog item.

See also the function `SetDialogItem` (page 881).

Version Notes

This function was changed with Appearance Manager 1.0 to support retrieving item information from controls.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

ictbSample

Declared In

`Dialogs.h`

GetDialogItemAsControl

Obtains the control handle for a dialog item in an embedding hierarchy.

```
OSErr GetDialogItemAsControl (
   DialogRef inDialog,
   DialogItemIndex inItemNo,
   ControlRef *outControl
);
```

Parameters

inDialog

A pointer to the dialog box to examine.

inItemNo

The position of an item in the dialog box's item list.

outControl

A pointer to a control handle that, on return, refers to the embedded control.

Return Value

A result code. See "Dialog Manager Result Codes" (page 916). The Control Manager result code `errItemNotControl` indicates that the specified dialog item is not a control.

Discussion
When an embedding hierarchy is established, `GetDialogItemAsControl` produces a handle to the embedded controls (except Help items). It should be used instead of `GetDialogItem` (page 852) when an embedding hierarchy is established.

Version Notes
This function is available with Appearance Manager 1.0 and later.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

GetDialogItemText

Obtains the text string contained in an edit text or a static text item.

```
void GetDialogItemText (
   Handle item,
   Str255 text
);
```

Parameters

item
On input, a handle to an edit text or a static text item. To get this handle, call the "Alert Button Constants" (page 904) function.

text
On output, a string containing the text of the item that is specified by the `item` parameter.

Discussion
The `GetDialogItemText` function will only return the first 255 characters in an edit text item.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

GetDialogKeyboardFocusItem

Returns the item number of the editable text item in a dialog box that has keyboard focus.

```
SInt16 GetDialogKeyboardFocusItem (
   DialogRef dialog
);
```

Parameters

dialog
On input, a pointer to the dialog structure for the dialog box whose currently focused item you want to identify.

Return Value
The number of the editable text item in a dialog box that currently has keyboard focus.

Discussion
When the Appearance Manager is available and an embedding hierarchy is established, you should call the Control Manager function `GetKeyboardFocus` instead of `GetDialogKeyboardFocusItem` to return the item number of the item in a dialog box that has keyboard focus.

The `GetDialogKeyboardFocusItem` function accesses the `edit` field in the dialog structure. `GetDialogKeyboardFocusItem` should only be called when there is no embedding hierarchy in the dialog box.

Version Notes
This function is not recommended with Appearance Manager 1.0 and later.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

GetDialogPort

```
CGrafPtr GetDialogPort (
   DialogRef dialog
);
```

Parameters
dialog

Return Value
See the QuickDraw Manager documentation for a description of the `CGrafPtr` data type.

Discussion

Special Considerations

Version Notes

Carbon Porting Notes

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

GetDialogTextEditHandle

```
TEHandle GetDialogTextEditHandle (
   DialogRef dialog
);
```

Parameters

dialog

Return Value

See the TextEdit documentation for a description of the `TEHandle` data type.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Dialogs.h`

GetDialogTimeout

Obtains the original countdown duration, the time remaining, and the item selection to be simulated for a specified modal dialog box.

```
OSStatus GetDialogTimeout (
   DialogRef inDialog,
   DialogItemIndex *outButtonToPress,
   UInt32 *outSecondsToWait,
   UInt32 *outSecondsRemaining
);
```

Parameters

inDialog

A pointer to the dialog box to be examined.

outButtonToPress

On input, a pointer to a signed 16-bit integer. On return, a value representing the number within the item list of the item that is to be selected. You may pass `NULL` for the `outButtonToPress` parameter if you do not desire this information.

outSecondsToWait

On input, a pointer to an unsigned 32-bit integer. On return, a value specifying the number of seconds that were originally set to elapse before the Dialog Manager simulates an item selection. You may pass `NULL` for the `outSecondsToWait` parameter if you do not desire this information.

outSecondsRemaining

On input, a pointer to an unsigned 32-bit integer. On return, a value specifying the number of seconds remaining before the Dialog Manager simulates an item selection. You may pass `NULL` for the `outSecondsRemaining` parameter if you do not desire this information.

Return Value

A result code. See "Dialog Manager Result Codes" (page 916).

Discussion

Also see the function `SetDialogTimeout` (page 882).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

GetDialogWindow

```
WindowRef GetDialogWindow (
   DialogRef dialog
);
```

Parameters
dialog

Return Value
See the QuickDraw Manager documentation for a description of the `WindowRef` data type.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
QTCarbonShell

Declared In
`Dialogs.h`

GetModalDialogEventMask

Obtains the events to be received by the `ModalDialog` function.

```
OSStatus GetModalDialogEventMask (
   DialogRef inDialog,
   EventMask *outMask
);
```

Parameters
inDialog
A pointer to the dialog box for which you wish to obtain the event mask.

outMask
On input, a pointer to a unsigned 16-bit integer of type `EventMask`. On return, your application may test the bits of this value to determine the event(s) that the dialog box is currently set to receive.

Return Value
A result code. See "Dialog Manager Result Codes" (page 916).

Discussion
Also see the function `SetModalDialogEventMask` (page 884).

Version Notes
This function is available with Mac OS 8.5 and later.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

GetNewDialog

Creates a dialog box from a resource-based description.

```
DialogRef GetNewDialog (
   SInt16 dialogID,
   void *dStorage,
   WindowRef behind
);
```

Parameters

dialogID

The resource ID of a dialog resource and an extended dialog resource. The resource IDs for both resources must be identical. If the dialog resource is missing, the Dialog Manager returns to your application without creating the requested dialog box. See `'DLOG'` and `'dlgx'` for a description of the dialog resource and the extended dialog resource, respectively.

dStorage

A pointer to the memory for the dialog structure. If you set this parameter to `null`, the Dialog Manager automatically allocates a nonrelocatable block in your application heap.

behind

A pointer to the window behind which the dialog box is to be placed on the desktop. Set this parameter to the window pointer `(WindowPtr)-1L` to bring the dialog box in front of all other windows.

Return Value
Returns a pointer to a dialog box. If none was created, returns `null`. See the description of the `DialogRef` data type.

Discussion
The `GetNewDialog` function creates a dialog structure from information in a dialog resource and an extended dialog resource (if it exists) and returns a pointer to the dialog structure. You can use this pointer with Window Manager or QuickDraw functions to manipulate the dialog box. If the dialog resource specifies that the dialog box should be visible, the dialog box is displayed. If the dialog resource specifies that the dialog box should initially be invisible, use the Window Manager function `ShowWindow` to display the dialog box.

The dialog resource contains a resource ID that specifies both the dialog box's item list (`'DITL'`) resource and its dialog font table (`'dftb'`) resource. After calling the Resource Manager to read these resources into memory (if they are not already in memory), `GetNewDialog` makes a copy of the `'DITL'` resource and uses that copy; thus you may have several dialog boxes with identical items.

If you supply a dialog color table (`'dctb'`) resource with the same resource ID as the dialog resource, `GetNewDialog` uses `NewColorDialog` and returns a pointer to a color graphics port. If no dialog color table resource is present, `GetNewDialog` uses `NewDialog` to return a pointer to a black-and-white graphics port, although system software draws the window frame using the system's default colors. However, if the Appearance Manager is available and the `kDialogFlagsUseThemeBackground` feature bit of the extended dialog resource is set, then the `'dctb'` resource is ignored and a color graphics port is created.

Special Considerations

The `GetNewDialog` function doesn't release the memory occupied by the resources. Therefore, your application should mark all resources used for a dialog box as purgeable or you should release the resources yourself.

If either the dialog resource or the item list resource can't be read, the function result is `null`; your application should test to ensure that `null` is not returned before performing any more operations with the dialog box or its items.

As with all other windows, dialogs are created with an update region equal to their port rectangle. However, if the dialog's `'DLOG'` resource specifies that the dialog be made visible upon creation, the Dialog Manager draws the controls immediately and calls `ValidRgn` for each of their bounding rectangles. Other items are not drawn until the first update event for the dialog box is serviced.

If you need to display an alert box while your application is running in the background or is otherwise invisible to the user, call `AEInteractWithUser`

Version Notes

This function was changed with Appearance Manager 1.0 to support the extended dialog (`'dlgx'`) resource and the dialog font table (`'dftb'`) resource.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

ictbSample

Declared In

`Dialogs.h`

GetParamText

```
void GetParamText (
   StringPtr param0,
   StringPtr param1,
   StringPtr param2,
   StringPtr param3
);
```

Parameters

param0

param1

param2

param3

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Dialogs.h`

GetStandardAlertDefaultParams

Fills out an `AlertStdCFStringAlertParamRec` with default values: - not movable - no help button - default button with title "OK" - no cancel or other buttons.

```
OSStatus GetStandardAlertDefaultParams (
   AlertStdCFStringAlertParamPtr param,
   UInt32 version
);
```

Parameters

param

The parameter block to initialize.

version

The parameter block version; pass kStdCFStringAlertVersionOne.

Return Value

A result code. See "Dialog Manager Result Codes" (page 916).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

QTCarbonShell

Declared In

`Dialogs.h`

GetStdFilterProc

Returns a pointer to the standard filter function.

```
OSErr GetStdFilterProc (
   ModalFilterUPP *theProc
);
```

Parameters

theProc

A universal procedure pointer to a filter function. On output, the Dialog Manager provides a pointer to its standard filter function.

Return Value

A result code. See "Dialog Manager Result Codes" (page 916).

Discussion

The `GetStdFilterProc` function gets a pointer to the standard filter function. You must dispatch the function yourself using the CallModalFilterProc macro; see `ModalFilterProcPtr` (page 891).

You normally don't need to use `GetStdFilterProc` unless your development environment doesn't include the code required to support `StdFilterProc` (page 888).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
Dialogs.h

HideDialogItem

Makes an item in a dialog box invisible.

```
void HideDialogItem (
   DialogRef theDialog,
   DialogItemIndex itemNo
);
```

Parameters

theDialog

A pointer to a dialog structure.

itemNo

A number corresponding to the position of an item in the dialog box's item list resource.

Return Value

Discussion

The HideDialogItem function hides the item specified by itemNo by giving it a display rectangle that's off the screen. Specifically, if the left coordinate of the item's display rectangle is less than 8192 (hexadecimal 0x2000), HideDialogItem adds 16,384 (hexadecimal 0x4000) to both the left and right coordinates of the rectangle. If the item is already hidden (that is, if the left coordinate is greater than 8192), HideDialogItem does nothing. To redisplay an item that's been hidden by HideDialogItem, you can use the ShowDialogItem function.

Special Considerations

If your application needs to display a number of dialog boxes that are similar except for one or two items, it's generally easier to modify the common elements using the AppendDITL (page 836) and ShortenDITL (page 885) functions than to use the HideDialogItem and ShowDialogItem (page 885) functions.

If you hid an edit text item, the next visible edit text item will be highlighted.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

Dialogs.h

InsertDialogItem

```
OSStatus InsertDialogItem (
   DialogRef theDialog,
   DialogItemIndex afterItem,
   DialogItemType itemType,
   Handle itemHandle,
   const Rect *box
);
```

Parameters

theDialog

afterItem

itemType

itemHandle

box

Return Value

A result code. See "Dialog Manager Result Codes" (page 916).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

Dialogs.h

InvokeModalFilterUPP

```
Boolean InvokeModalFilterUPP (
   DialogRef theDialog,
   EventRecord *theEvent,
   DialogItemIndex *itemHit,
   ModalFilterUPP userUPP
);
```

Parameters

theDialog

theEvent

itemHit

userUPP

Availability

Available in Mac OS X v10.0 and later.

Declared In

Dialogs.h

InvokeModalFilterYDUPP

```
Boolean InvokeModalFilterYDUPP (
   DialogRef theDialog,
   EventRecord *theEvent,
   short *itemHit,
   void *yourDataPtr,
   ModalFilterYDUPP userUPP
);
```

Parameters

theDialog

theEvent

itemHit

yourDataPtr

userUPP

Carbon Porting Notes

This function is supported in Carbon because several QuickTime routines require it.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Dialogs.h`

InvokeUserItemUPP

```
void InvokeUserItemUPP (
   DialogRef theDialog,
   DialogItemIndex itemNo,
   UserItemUPP userUPP
);
```

Parameters

theDialog

itemNo

userUPP

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Dialogs.h`

IsDialogEvent

Determines whether a modeless dialog box or a movable modal dialog box is active when an event occurs.

```
Boolean IsDialogEvent (
   const EventRecord *theEvent
);
```

Parameters

theEvent

A pointer to an event structure returned by an Event Manager function such as `WaitNextEvent`.

Return Value

A Boolean value. If any event, including a null event, occurs when your dialog box is active, `IsDialogEvent` returns `true`; otherwise, it returns `false`.

Discussion

When `IsDialogEvent` returns `false`, pass the event to the rest of your event-handling code. When `IsDialogEvent` returns `true`, pass the event to `DialogSelect` (page 845) after testing for the events that `DialogSelect` does not handle.

A dialog structure includes a window structure. When you use the `GetNewDialog` (page 858) , `NewDialog` (page 870) , `NewFeaturesDialog` (page 871) , or `NewColorDialog` (page 868) functions to create a dialog box, the Dialog Manager sets the `windowKind` field in the window structure to `dialogKind`. To determine whether the active window is a dialog box, `IsDialogEvent` checks the `windowKind` field.

Before passing the event to `DialogSelect`, you should perform the following tests whenever `IsDialogEvent` returns `true` :

- Check whether the event is a key-down event for the Return, Enter, Esc, or Command-period keystrokes. When the user presses the Return or Enter key, your application should respond as if the user had clicked the default button; when the user presses Esc or Command-period, your application should respond as if the user had clicked the Cancel button. Use the Control Manager function `HiliteControl` to highlight the applicable button for 8 ticks.
- At this point, you may also want to check for and respond to any special events that you do not wish to pass to `DialogSelect` (page 845) or that require special processing before you pass them to `DialogSelect`. You would need to do this, for example, if the dialog box needs to respond to disk-inserted events.
- Check whether the event is an update event for a window other than the dialog box and, if it is, update your window.
- For complex items that you create, such as pictures or application-defined items that emulate complex controls, test for and respond to mouse events inside those items as appropriate. When `DialogSelect` calls the Control Manager function `TrackControl`, it does not allow you to specify the action function necessary for anything more complex than a button, radio button, or checkbox. If you need a more complex control (for example, one that measures how long the user holds down the mouse button or how far the user has moved an indicator), you can create your own control or a picture or an application-defined item that draws a control-like object in your dialog box. You must then test for and respond to those events yourself.

If your application uses `IsDialogEvent` to help handle events when you display a movable modal dialog box, perform the following additional tests before passing events to `DialogSelect` :

- Test for mouse-down events in the title bar of the movable modal dialog box and respond by dragging the dialog box accordingly.

- Test for and respond to mouse-down events in the Apple menu and, if the movable modal dialog box includes edit text items, in the Edit menu. (You should disable all other menus when you display a movable modal dialog box.)
- Play the system alert sound for every other mouse-down event outside the movable modal dialog box.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

ModalDialog

Handles events while your application displays a modal or movable modal dialog box.

```
void ModalDialog (
   ModalFilterUPP modalFilter,
   DialogItemIndex *itemHit
);
```

Parameters

`modalFilter`

A universal procedure pointer for an event filter function. For modal dialog boxes, you can specify `null` if you want to use the standard event-handling function. For movable modal dialog boxes, you should specify your own event filter function.

`itemHit`

A pointer to a short integer. After receiving an event involving an enabled item, `ModalDialog` produces a number representing the position of the selected item in the active dialog box's item list resource.

Return Value

Discussion
Call the `ModalDialog` function immediately after displaying a modal or movable modal dialog box. Your application should continue calling `ModalDialog` until the user dismisses your dialog.

For modal dialogs, the `ModalDialog` function repeatedly handles events until an event involving an enabled dialog box item—such as a click in a radio button, for example—occurs. If the event is a mouse-down event outside the content region of the dialog box, `ModalDialog` plays the system alert sound and gets the next event.

For movable modal dialogs, if the `kDialogFlagsHandleMovableModal` feature bit in the extended dialog resource is set, the `ModalDialog` function will handle all standard movable modal user interactions, such as dragging a dialog box by its title bar and allowing the user to switch into another application. However, a difference between the `ModalDialog` function's behavior with movable modal and modal dialogs is that, with movable modal dialogs, your event filter function receives all events. If you want the Dialog Manager to assist you in handling events in movable modal dialog boxes, call `GetStdFilterProc` and `StdFilterProc`.

For events inside the dialog box, `ModalDialog` passes the event to the event filter function pointed to in the `modalFilter` parameter before handling the event. When the event filter returns `false`, `ModalDialog` handles the event. If the event filter function handles the event, returning `true`, `ModalDialog` performs no more event handling.

If you set the `modalFilter` parameter to `null`, the standard event filter function is executed. The standard event filter function checks whether

- the user has pressed the Enter or Return key and, if so, returns the item number of the default button
- the user has pressed the Escape key or Command-period and, if so, returns the item number of the Cancel button
- the cursor is over an editable text box, and optionally changes the cursor to an I-beam whenever this is the case

If you set the `modalFilter` parameter to point to your own event filter function, that function can use the standard filter function to accomplish the above tasks. (To do so, you can call `GetStdFilterProc`, and dispatch the event to the standard filter function yourself, or you can call `StdFilterProc`, which obtains a `ModalFilterUPP` for the standard filter function and then dispatches the function.) Additionally, your own event filter function should also

- handle update events, so that background processes can receive processor time, and return `false`
- return `false` for all events that your event filter function doesn't handle

You can also use your event filter function to test for and respond to keyboard equivalents and more complex events—for instance, the user dragging the cursor within an application-defined item. You can use your same event filter function in most or all of your alert and modal dialog boxes.

If the event filter function does not handle the event (returning `false`), `ModalDialog` handles the event as follows:

- In response to an activate or update event for the dialog box, `ModalDialog` activates or updates its window.
- If the user presses the mouse button while the cursor is in an editable text item, `ModalDialog` responds to the mouse activity as appropriate—that is, either by displaying an insertion point or by selecting text. If a key-down event occurs and there's an editable text item, `ModalDialog` uses TextEdit to handle text entry and editing automatically. If the editable text item is enabled, `ModalDialog` produces its item number after it receives either the mouse-down or key-down event. Normally, editable text items are disabled, and you use the `GetDialogItemText` function to read the information in the items only after the user clicks the OK button.
- If the user presses the mouse button while the cursor is in a control, `ModalDialog` calls the Control Manager function `TrackControl`. If the user releases the mouse button while the cursor is in an enabled control, `ModalDialog` produces the control's item number. Your application should respond appropriately—for example, by performing a command after the user clicks the OK button.
- If the user presses the mouse button while the cursor is in any other enabled item in the dialog box, `ModalDialog` produces the item's number, and your application should respond appropriately. Generally, only controls should be enabled. If your application creates a control more complex than a button, radio button, or checkbox, your application must handle events inside that item with your event filter function.
- If the user presses the mouse button while the cursor is in a disabled item or in no item, or if any other event occurs, `ModalDialog` does nothing.

Special Considerations

The `ModalDialog` function traps all events. This prevents your event loop from receiving activate events for your windows. Thus, if one of your application's windows is active when you use `GetNewDialog` to create a modal dialog box, you must explicitly deactivate that window before displaying the modal dialog box.

When `ModalDialog` calls the Control Manager function `TrackControl`, it does not allow you to specify the action function necessary for anything more complex than a button, radio button, or checkbox. If you need a more complex control, you can create your own control, a picture, or an application-defined item that draws a control-like object in your dialog box. You must then provide an event filter function that appropriately handles events in that item.

Version Notes
This function was changed with Appearance Manager 1.0 to handle events for movable modal dialogs.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
ictbSample

Declared In
`Dialogs.h`

MoveDialogItem

Moves a dialog item to a specified location in a window.

```
OSErr MoveDialogItem (
   DialogRef inDialog,
   DialogItemIndex inItemNo,
   SInt16 inHoriz,
   SInt16 inVert
);
```

Parameters

inDialog
A pointer to the dialog box containing the item to move.

inItemNo
The position of the item in the dialog box's item list resource use `FindDialogItem` (page 849) to determine this value.

inHoriz
The new horizontal coordinate for the dialog item.

inVert
The new vertical coordinate for the dialog item.

Return Value
A result code. See "Dialog Manager Result Codes" (page 916).

Discussion
The `MoveDialogItem` function moves a dialog item to a specified location in a window. `MoveDialogItem` ensures that if the item is a control, the control rectangle and the dialog item rectangle (maintained by the Dialog Manager) are always the same.

Version Notes
This function is available with Appearance Manager 1.0 and later.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

NewColorDialog

Creates a dialog box from information passed in memory.

```
DialogRef NewColorDialog (
   void *dStorage,
   const Rect *boundsRect,
   ConstStr255Param title,
   Boolean visible,
   SInt16 procID,
   WindowRef behind,
   Boolean goAwayFlag,
   SRefCon refCon,
   Handle items
);
```

Parameters

dStorage

On input, a pointer to the memory for the dialog structure. If you set this parameter to `null`, the Dialog Manager automatically allocates a nonrelocatable block in your application heap.

boundsRect

On input, a pointer to a rectangle, given in global coordinates, that determines the size and position of the dialog box; these coordinates specify the upper-left and lower-right corners of the dialog box.

title

On input, a text string used for the title of a modeless or movable modal dialog box. You can specify an empty string (not `null`) for a title bar that contains no text.

visible

On input, a flag that specifies whether the dialog box should be drawn on the screen immediately. If you set this parameter to `false`, the dialog box is not drawn until your application uses the Window Manager function `ShowWindow` to display it.

procID

On input, the window definition ID for the type of dialog box, specified with constants defined by the Window Manager. Use the `kWindowModalDialogProc` constant to specify modal dialog boxes, the `kWindowDocumentProc` constant to specify modeless dialog boxes, and the `kWindowMovableModalDialogProc` constant to specify movable modal dialog boxes.

behind

On input, a pointer to the window behind which the dialog box is to be placed on the desktop. Set this parameter to the window pointer `(WindowPtr)-1L` to bring the dialog box in front of all other windows.

goAwayFlag

On input, a flag to specify whether a modeless dialog box can have a close box in its title bar when the dialog box is active. If you set this parameter to `true`, the modeless dialog box has a close box in its title bar when the window is active.

refCon

On input, a value that the Dialog Manager uses to set the `refCon` field of the dialog box's window structure. Your application may store any value here for any purpose. For example, your application can store a number that represents a dialog box type, or it can store a handle to a structure that maintains state information about the dialog box. You can use the Window Manager function `SetWRefCon` at any time to change this value in the dialog structure for a dialog box, and you can use the `GetWRefCon` function to determine its current value.

items

On input, a handle to an item list resource for the dialog box. You can get the handle by calling the Resource Manager function `GetResource` to read the item list resource into memory. Use the Memory Manager function `HNoPurge` to make the handle unpurgeable while you use it or use the Operating System utility function `HandToHand` to make a copy of the handle and use the copy.

Return Value

A pointer to the new dialog box. If the function doesn't create a new dialog box, returns `null`. See the description of the `DialogRef` data type.

Discussion

The `NewColorDialog` function creates a dialog box as specified by its parameters. The first eight parameters (`dStorage` through `refCon`) are passed to the Window Manager function `NewWindow`, which creates the dialog box. You can use this pointer with Window Manager or QuickDraw functions to manipulate the dialog box.

The Dialog Manager uses the default window colors for the dialog box. By using the system's default colors, you ensure that your application's interface is consistent with that of the Finder and other applications. However, if you absolutely feel compelled to break from this consistency, you can use the Window Manager function `SetWinColor` to use your own dialog color table resource that specifies colors other than the default colors. Be aware, however, that nonstandard colors in your alert and dialog boxes may initially confuse your users.

The Window Manager creates an auxiliary window structure for the color dialog box. You can access this structure with the Window Manager function `GetAuxWin`. If the dialog box's content color isn't white, it's a good idea to call `NewColorDialog` with the `visible` flag set to `false`. After the color table and color item list resource are installed, use the Window Manager function `ShowWindow` to display the dialog box if it's the frontmost window. If the dialog box is a modeless dialog box that is not in front, use the Window Manager function `ShowHide` to display it.

The `NewColorDialog` function generates an update event for the entire window contents. Thus, with the exception of controls, items aren't drawn immediately. The Dialog Manager calls the Control Manager to draw controls, and the Control Manager draws them immediately. So that the controls won't be drawn twice, the Dialog Manager calls the Window Manager function `ValidRect` for the enclosing rectangle of each control. If you find that there is too great a lag between the drawing of controls and the drawing of other items, try making the dialog box initially invisible and then calling the Window Manager function `ShowWindow` to show it.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Dialogs.h`

NewDialog

Creates a dialog box from information passed in memory.

```
DialogRef NewDialog (
   void *dStorage,
   const Rect *boundsRect,
   ConstStr255Param title,
   Boolean visible,
   SInt16 procID,
   WindowRef behind,
   Boolean goAwayFlag,
   SRefCon refCon,
   Handle items
);
```

Parameters

dStorage

On input, a pointer to the memory for the dialog structure. If you set this parameter to `null`, the Dialog Manager automatically allocates a nonrelocatable block in your application heap.

boundsRect

On input, a pointer to a rectangle, given in global coordinates, that determines the size and position of the dialog box; these coordinates specify the upper-left and lower-right corners of the dialog box.

title

On input, a text string used for the title of a modeless or movable modal dialog box. You can specify an empty string (not `null`) for a title bar that contains no text.

visible

On input, a flag that specifies whether the dialog box should be drawn on the screen immediately. If you set this parameter to `false`, the dialog box is not drawn until your application uses the Window Manager function `ShowWindow` to display it.

procID

On input, the window definition ID for the type of dialog box, specified with constants defined by the Window Manager. Use the `kWindowModalDialogProc` constant to specify modal dialog boxes, the `kWindowDocumentProc` constant to specify modeless dialog boxes, and the `kWindowMovableModalDialogProc` constant to specify movable modal dialog boxes.

behind

On input, a pointer to the window behind which the dialog box is to be placed on the desktop. Set this parameter to the window pointer `(WindowPtr)-1L` to bring the dialog box in front of all other windows.

goAwayFlag

On input, a flag to specify whether a modeless dialog box can have a close box in its title bar when the dialog box is active. If you set this parameter to `true`, the modeless dialog box has a close box in its title bar when the window is active.

refCon

On input, a value that the Dialog Manager uses to set the `refCon` field of the dialog box's window structure. Your application may store any value here for any purpose. For example, your application can store a number that represents a dialog box type, or it can store a handle to a structure that maintains state information about the dialog box. You can use the Window Manager function `SetWRefCon` at any time to change this value in the dialog structure for a dialog box, and you can use the `GetWRefCon` function to determine its current value.

items

On input, a handle to an item list resource for the dialog box. You can get the handle by calling the Resource Manager function `GetResource` to read the item list resource into memory. Use the Memory Manager function `HNoPurge` to make the handle unpurgeable while you use it or use the Operating System utility function `HandToHand` to make a copy of the handle and use the copy.

Return Value
A pointer to the new dialog box. If the function doesn't create a new dialog box, returns `null`. See the description of the `DialogRef` data type.

Discussion
The `NewDialog` function is identical to the `NewColorDialog` function, except that `NewDialog` returns a pointer to a black-and-white graphics port. See the discussion of `NewColorDialog` (page 868) for descriptions of the parameters that you also pass to `NewDialog`.

The `NewDialog` function creates a dialog box as specified by its parameters and returns a pointer to a black-and-white graphics port for the new dialog box. The first eight parameters (`wStorage` through `refCon`) are passed to the Window Manager function `NewWindow`, which creates the dialog box.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

NewFeaturesDialog

Creates a dialog box from information passed in memory.

```
DialogRef NewFeaturesDialog (
   void *inStorage,
   const Rect *inBoundsRect,
   ConstStr255Param inTitle,
   Boolean inIsVisible,
   SInt16 inProcID,
   WindowRef inBehind,
   Boolean inGoAwayFlag,
   SRefCon inRefCon,
   Handle inItemListHandle,
   UInt32 inFlags
);
```

Parameters

inStorage

A pointer to the memory for the dialog box. If you set this parameter to `null`, the Dialog Manager automatically allocates a nonrelocatable block in your application heap.

inBoundsRect

A pointer to a rectangle, given in global coordinates, that determines the size and position of the dialog box; these coordinates specify the upper-left and lower-right corners of the dialog box.

inTitle

A pointer to a text string used for the title of a modeless or movable modal dialog box. You can specify an empty string (not `null`) for a title bar that contains no text.

inIsVisible

A flag that specifies whether the dialog box should be drawn on the screen immediately. If you set this parameter to `false`, the dialog box is not drawn until your application uses the Window Manager function `ShowWindow` to display it.

inProcID

The window definition ID for the type of dialog box, specified with constants defined by the Window Manager. Use the `kWindowModalDialogProc` constant to specify modal dialog boxes, the `kWindowDocumentProc` constant to specify modeless dialog boxes, and the `kWindowMovableModalDialogProc` constant to specify movable modal dialog boxes.

inBehind

A pointer to the window behind which the dialog box is to be placed on the desktop. Set this parameter to the window pointer `(WindowPtr)-1L` to bring the dialog box in front of all other windows.

inGoAwayFlag

A Boolean value. If `true`, specifies that an active modeless dialog box has a close box in its title bar.

inRefCon

A value that the Dialog Manager uses to set the `refCon` field of the dialog box's window structure. Your application may store any value here for any purpose. For example, your application can store a number that represents a dialog box type, or it can store a handle to a structure that maintains state information about the dialog box. You can use the Window Manager function `SetWRefCon` at any time to change this value in the dialog structure for a dialog box, and you can use the `GetWRefCon` function to determine its current value.

inItemListHandle

A handle to an item list resource for the dialog box. You can get the handle by calling the Resource Manager function `GetResource` to read the item list resource into memory.

inFlags

An unsigned 32-bit mask specifying the dialog box's Appearance-compliant feature flags see "Dialog Feature Flag Constants" (page 909). To establish an embedding hierarchy in a dialog box, pass `kDialogFlagsUseControlHierarchy` in the `inFlags` parameter.

Return Value

A pointer to the newly created dialog box. If `NewFeaturesDialog` doesn't create a new dialog box, it returns `null`. See the description of the `DialogRef` data type.

Discussion

The `NewFeaturesDialog` function creates a dialog box without using `'DLOG'` or `'dlgx'` resources. Although the `inItemListHandle` parameter specifies an item list (`'DITL'`) resource for the dialog box, the corresponding dialog font table (`'dftb'`) resource is not automatically accessed. You must explicitly set the dialog box's control font style(s) individually.

Version Notes

This function is available with Appearance Manager 1.0 and later.

Carbon Porting Notes

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Dialogs.h`

NewModalFilterUPP

```
ModalFilterUPP NewModalFilterUPP (
   ModalFilterProcPtr userRoutine
);
```

Parameters

userRoutine

Return Value
See the description of the `ModalFilterUPP` data type.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Dialogs.h`

NewModalFilterYDUPP

```
ModalFilterYDUPP NewModalFilterYDUPP (
   ModalFilterYDProcPtr userRoutine
);
```

Parameters

userRoutine

Return Value
See the description of the `ModalFilterYDUPP` data type.

Carbon Porting Notes

This function is supported in Carbon because several QuickTime routines require it.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Dialogs.h`

NewUserItemUPP

```
UserItemUPP NewUserItemUPP (
   UserItemProcPtr userRoutine
);
```

Parameters

userRoutine

Return Value
See the description of the `UserItemUPP` data type.

Discussion

Special Considerations

Version Notes

Carbon Porting Notes

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Dialogs.h`

NoteAlert

Displays an alert box with a note icon and/or plays an alert sound.

```
DialogItemIndex NoteAlert (
   SInt16 alertID,
   ModalFilterUPP modalFilter
);
```

Parameters

alertID

The resource ID of an alert resource and extended alert resource. If the alert resource is missing, the Dialog Manager returns to your application without creating the requested alert. See `'alrx'` for a description of the extended alert resource.

modalFilter

A universal procedure pointer for a filter function that responds to events not handled by the `ModalDialog` (page 865) function. If you set this parameter to `null`, the Dialog Manager uses the standard event filter function.

Return Value
If no alert box is to be drawn at the current alert stage, `NoteAlert` returns –1 otherwise, it creates and displays the alert box and returns the item number of the control selected by the user see "Alert Button Constants" (page 904). See the description of the `DialogItemIndex` data type.

Discussion
The `NoteAlert` function displays an alert box with a note icon in its upper-left corner or, if appropriate for the alert stage, plays an alert sound instead of or in addition to displaying the alert box.

The `NoteAlert` function is the same as the `Alert` (page 834) function except that, before drawing the items in the alert box, `NoteAlert` draws the note icon in the upper-left corner. The note icon has resource ID 1, which you can also specify with the constant `noteIcon`. By default, the Dialog Manager uses the standard note icon from the System file. You can change this icon by providing your own `'ICON'` resource with resource ID 1.

Use a note alert to inform users of a minor mistake that won't have any disastrous consequences if left as is. Usually this type of alert simply offers information, and the user responds by clicking an OK button. Occasionally, a note alert may ask a simple question and provide a choice of responses.

Your application should never draw its own default rings or alert icons. Prior to Mac OS 8, the `NoteAlert` function would only redraw the alert icon and default button ring once and never redraw them on an update event. However, when Appearance is available, alert icons and default rings do redraw when you call `NoteAlert`.

See also the functions `CautionAlert` (page 839) and `StopAlert` (page 889).

Version Notes
This function was changed with Appearance Manager 1.0 to support the extended alert (`'alrx'`) resource.

Carbon Porting Notes

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

ParamText

Replaces the text strings in the static text items of your alert or dialog boxes while your application is running.

```
void ParamText (
   ConstStr255Param param0,
   ConstStr255Param param1,
   ConstStr255Param param2,
   ConstStr255Param param3
);
```

Parameters

param0
A text string to substitute for the special string `^0` in the static text items of all subsequently created alert and dialog boxes.

param1
A text string to substitute for the special string `^1` in the static text items of all subsequently created alert and dialog boxes.

param2
A text string to substitute for the special string `^2` in the static text items of all subsequently created alert and dialog boxes.

param3
A text string to substitute for the special string `^3` in the static text items of all subsequently created alert and dialog boxes.

Discussion
The `ParamText` function replaces the special strings `^0` through `^3` in the static text items of all subsequently created alert and dialog boxes with the text strings you pass as parameters. Pass empty strings (not `null`) for parameters not used.

Special Considerations

If the user launches a desk accessory (such as a driver) in your application's partition and the desk accessory calls `ParamText`, it may change the text in your application's dialog box.

You should be very careful about using `ParamText` in modeless dialog boxes. If a modeless dialog box using `ParamText` is onscreen and you display another dialog box or alert box that also uses `ParamText`, both boxes will be affected by the latest call to `ParamText`.

Note that you should try to store text strings in resource files to facilitate translation into other languages; therefore, `ParamText` is best used for supplying text strings, such as document names, that the user specifies. To avoid problems with grammar and sentence structure when you localize your application, you should use `ParamText` to supply only one text string per screen message.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

RemoveDialogItems

```
OSStatus RemoveDialogItems (
   DialogRef theDialog,
   DialogItemIndex itemNo,
   DialogItemIndex amountToRemove,
   Boolean disposeItemData
);
```

Parameters
theDialog
itemNo
amountToRemove
disposeItemData

Return Value
A result code. See "Dialog Manager Result Codes" (page 916).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

ResetAlertStage

Resets the current alert stage to the first alert stage.

```
void ResetAlertStage (
   void
);
```

Parameters

Return Value

Discussion
The `ResetAlertStage` function resets every alert to a first-stage alert.

Availability
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
`Dialogs.h`

RunStandardAlert

Shows and runs a standard alert using a modal dialog loop.

```
OSStatus RunStandardAlert (
   DialogRef inAlert,
   ModalFilterUPP filterProc,
   DialogItemIndex *outItemHit
);
```

Parameters

inAlert
The alert to display. On return, the alert you pass in this parameter has been released and is no longer valid. You should not call `DisposeDialog` (page 847) on this alert.

filterProc
An event filter function for handling events that do not apply to the alert. May be `NULL`.

outItemHit
On exit, contains the item index of the button that was pressed to close the alert.

Return Value
A result code. See "Dialog Manager Result Codes" (page 916).

Discussion
This function displays and runs an alert created by `CreateStandardAlert` (page 842). `RunStandardAlert` handles all user interaction with the alert.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
QTCarbonShell

Declared In
`Dialogs.h`

SelectDialogItemText

Selects and highlights text contained in an edit text item.

```
void SelectDialogItemText (
   DialogRef theDialog,
   DialogItemIndex itemNo,
   SInt16 strtSel,
   SInt16 endSel
);
```

Parameters

theDialog

On input, a pointer to a dialog structure.

itemNo

On input, a number corresponding to the position of an edit text item in the dialog box's item list resource.

strtSel

On input, a number representing the position of the first character to begin selecting.

endSel

On input, a number representing one position past the last character to be selected.

Discussion

If the item in the `itemNo` parameter is an edit text item that contains text, the `SelectDialogItemText` function sets the text selection range to extend from the character position specified in the `strtSel` parameter up to but not including the character position specified in the `endSel` parameter. The selection range is highlighted unless `strtSel` equals `endSel`, in which case a blinking vertical bar is displayed to indicate an insertion point at that position. If the edit text item doesn't contain text, `SelectDialogItemText` displays the insertion point.

You can select the entire text by specifying the number 0 in the `strtSel` parameter and the number 32767 in the `endSel` parameter.

For example, if the user makes an unacceptable entry in the edit text item, your application can display an alert box reporting the problem and then use `SelectDialogItemText` to select the entire text so it can be replaced by a new entry. Without this function, the user would have to select the item before making the new entry.

Carbon Porting Notes

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Dialogs.h`

SetDialogCancelItem

Sets the cancel item for a dialog box.

```
OSErr SetDialogCancelItem (
   DialogRef theDialog,
   DialogItemIndex newItem
);
```

Parameters

theDialog

On input, a pointer to the dialog structure for the dialog box whose cancel item you want to set.

newItem

On input, the item number of the item you want to set as the cancel item; see "Alert Button Constants" (page 904).

Return Value
A result code. See "Dialog Manager Result Codes" (page 916).

Discussion
If you intend to use the standard filter function, you can first use the functions `SetDialogDefaultItem` and `SetDialogCancelItem` to set the items that the standard filter function will treat as the default and cancel items. You can use `GetDialogDefaultItem` and `GetDialogCancelItem` to determine the dialog item numbers that the standard filter function will treat as the default and cancel items.

If you call the `SetDialogCancelItem` function before you call the standard filter function, the standard filter function automatically interprets Escape and Command-period keypresses to mean that the specified cancel item has been selected.

If you don't explicitly call `SetDialogCancelItem`, the standard filter function treats item 2 as the cancel item.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

SetDialogDefaultItem

Sets the default item for a dialog box and draws an appropriate border around the default item.

```
OSErr SetDialogDefaultItem (
   DialogRef theDialog,
   DialogItemIndex newItem
);
```

Parameters

theDialog

On input, a pointer to the dialog structure for the dialog box whose default item you want to set.

newItem

On input, the item number of the item you want to set as the default item.

Return Value
A result code. See "Dialog Manager Result Codes" (page 916).

Discussion
If you call the `SetDialogDefaultItem` function before you call the standard filter function, the standard filter function automatically interprets Return and Enter keypresses to mean that the specified default item has been selected.

If you don't explicitly call `SetDialogDefaultItem`, the standard filter function treats item 1 as the default item.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

SetDialogFont

Sets the font used in static and edit text items.

```
void SetDialogFont (
   SInt16 fontNum
);
```

Parameters

`fontNum`

A font ID number. Do not rely on font number constants. Instead, use the Font Manager function `GetFNum` to find the font number to pass in this parameter.

Discussion
For subsequently created dialog and alert boxes, `SetDialogFont` sets the font of the dialog or alert box's graphics port to the specified font. If you don't call this function, the system font is used. The `SetDialogFont` function does not affect titles of controls, which are always displayed in the system font.

Special Considerations

There are a number of caveats regarding the SetDialogFont function:

1. Most importantly, your application will be much easier to localize if you always use the system font in your alert and dialog boxes and never use `SetDialogFont`.
2. The Standard File Package does not always properly calculate the position of the standard file dialog box once this function has been called; for example, the standard file dialog box may be partially obscured by a menu bar.
3. Be aware that this function affects all static text and edit text items in all of the alert and dialog boxes you subsequently display.
4. `SetDialogFont` does not change the font for control titles.
5. You can't use `SetDialogFont` to change the font size or font style.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
Dialogs.h

SetDialogItem

Sets or changes information for a dialog item.

```
void SetDialogItem (
   DialogRef theDialog,
   DialogItemIndex itemNo,
   DialogItemType itemType,
   Handle item,
   const Rect *box
);
```

Parameters

theDialog
A pointer to the dialog box containing the dialog item.

itemNo
The position of the item in the dialog box's item list resource use FindDialogItem (page 849) to determine this value.

itemType
A short value. Pass an item type constant identifying the dialog item specified in the itemNo parameter. When an embedding hierarchy is established, only the kItemDisableBit item type constant is honored.

item
Either a handle to the dialog item specified in the itemNo parameter or, for a custom dialog item, a pointer (coerced to a handle) to an application-defined item drawing function. When an embedding hierarchy is established, the item parameter is ignored unless you pass a universal procedure pointer to an application-defined item draw function.

box
A pointer to the display rectangle (in local coordinates) for the item specified in the itemNo parameter. If you set the control rectangle on an item when an embedding hierarchy is present, SetDialogItem will move and resize the item appropriately for you, on return.

Return Value

Discussion
The SetDialogItem function sets the item specified by the itemNo parameter for the specified dialog box. If an embedding hierarchy exists, however, you cannot change the type or handle of an item, although application-defined item drawing functions can still be set.

See also the function GetDialogItem (page 852).

Version Notes
This function was changed with Appearance Manager 1.0 to work with embedding hierarchies.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
Dialogs.h

SetDialogItemText

Sets the text string for static text and editable text fields.

```
void SetDialogItemText (
   Handle item,
   ConstStr255Param text
);
```

Parameters

item

A handle to an editable text field or static text field. When embedding is on, you should pass in the control handle produced by a call to the function . If embedding is not on, pass in the handle produced by the "Alert Button Constants" (page 904) function.

text

A pointer to a string containing the text to display in the field.

Discussion

The `SetDialogItemText` function sets and redraws text strings for static text and editable text fields. `SetDialogItemText` is useful for supplying a default text string—such as a document name—for an editable text field while your application is running.

Version Notes

This function was changed with Appearance Manager 1.0 to support embedding hierarchies.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Dialogs.h`

SetDialogTimeout

Simulates an item selection in a modal dialog box after a specified amount of time elapses.

```
OSStatus SetDialogTimeout (
   DialogRef inDialog,
   DialogItemIndex inButtonToPress,
   UInt32 inSecondsToWait
);
```

Parameters

inDialog

A pointer to the dialog box for which an item selection is to be simulated.

inButtonToPress

A signed 16-bit integer. Pass a value representing the number (within the item list) of the item that is to be selected.

inSecondsToWait

An unsigned 32-bit integer. Pass a value specifying the number of seconds that are to elapse before the Dialog Manager simulates an item selection. Pass 0 to clear a preexisting timeout value and cease the countdown in progress.

Return Value
A result code. See "Dialog Manager Result Codes" (page 916).

Discussion
Your application calls the `SetDialogTimeout` function each time you wish to start a countdown of the specified duration for a given modal dialog box. When the amount of time specified in the `inSecondsToWait` parameter has elapsed, the Dialog Manager simulates a click on the button specified in the `inButtonToPress` parameter. If your application calls `SetDialogTimeout` again, or if any event is received for the dialog box, the countdown is restarted.

In order to use `SetDialogTimeout` with a given modal dialog box, your application must handle events for the dialog box through the `ModalDialog` function. The Dialog Manager will not simulate an item selection for the dialog box until `ModalDialog` processes an event (including null events).

Also see the function `GetDialogTimeout` (page 856).

Version Notes
This function is available with Mac OS 8.5 and later.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

SetDialogTracksCursor

Determines whether the Dialog Manager tracks the cursor's movements and changes the cursor to an I-beam whenever it is over an edit dialog box.

```
OSErr SetDialogTracksCursor (
   DialogRef theDialog,
   Boolean tracks
);
```

Parameters

`theDialog`
On input, a pointer to the dialog structure for the dialog box containing one or more edit text items for which you want the Dialog Manager to track the cursor.

`tracks`
On input, a Boolean value. A value of `true` indicates you want the Dialog Manager to track the cursor's movements and change it to an I-beam whenever the cursor is over an edit dialog box a value of `false` indicates you don't want the Dialog Manager to track the cursor in this manner.

Return Value
A result code. See "Dialog Manager Result Codes" (page 916).

Discussion
You should call SetDialogTracksCursor before you call the standard filter function.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

SetModalDialogEventMask

Specifies the events to be received by the `ModalDialog` function.

```
OSStatus SetModalDialogEventMask (
   DialogRef inDialog,
   EventMask inMask
);
```

Parameters

inDialog
A pointer to the dialog box for which you wish to set the event mask.

inMask
The desired mask value(s) for the event(s) you wish the dialog box to receive.

Return Value
A result code. See "Dialog Manager Result Codes" (page 916).

Discussion
Your application can use the `SetModalDialogEventMask` function to specify the events received by the `ModalDialog` function for a given modal dialog box. This allows your application to specify additional events that are not by default received by `ModalDialog`, such as disk-inserted events and operating-system events. If you use `SetModalDialogEventMask` to change the `ModalDialog` function's event mask, you should pass `ModalDialog` a pointer to your own event filter function to handle any added events.

Also see the function `GetModalDialogEventMask` (page 857).

Version Notes
This function is available with Mac OS 8.5 and later.

Carbon Porting Notes

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

SetPortDialogPort

```
void SetPortDialogPort (
   DialogRef dialog
);
```

Parameters

dialog

Availability
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
`Dialogs.h`

ShortenDITL

Removes items from an existing dialog box while your application is running.

```
void ShortenDITL (
   DialogRef theDialog,
   DialogItemIndex numberItems
);
```

Parameters

theDialog
A pointer to a dialog structure.

numberItems
The number of items to remove (starting from the last item in the item list resource).

Discussion
The `ShortenDITL` function removes the specified number of items from the dialog box. This function is especially useful if several dialog boxes share a single item list resource, because you can use `ShortenDITL` to remove items as necessary for individual dialog boxes.

You typically create an invisible dialog box, call the `ShortenDITL` function, then make the dialog box visible by using the Window Manager function `ShowWindow`. Note that `ShortenDITL` does not automatically resize the dialog box; you can use `AutoSizeDialog` (page 838) or the Window Manager function `SizeWindow` if you need to resize the dialog box.

Special Considerations

The `ShortenDITL` function is available in System 7 and in earlier versions of the Communications Toolbox. Before calling `ShortenDITL`, you should make sure that it is available by using the `Gestalt` function with the `gestaltDITLExtAttr` selector. Test the bit indicated by the `gestaltDITLExtPresent` constant in the `response` parameter. If the bit is set, then `ShortenDITL` is available.

Carbon Porting Notes

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

ShowDialogItem

Redisplays an item that has been hidden by HideDialogItem.

```
void ShowDialogItem (
   DialogRef theDialog,
   DialogItemIndex itemNo
);
```

Parameters

theDialog

On input, a pointer to a dialog structure.

itemNo

On input, a number corresponding to the position of an item in the dialog box's item list resource.

Return Value

Discussion

The `ShowDialogItem` function redisplays the item specified in the `itemNo` parameter by restoring the display rectangle the item had prior to `HideDialogItem` (page 861). If the left coordinate of the item's display rectangle is greater than 8192, `ShowDialogItem` subtracts 16,384 from both the left and right coordinates of the rectangle. If the item is already visible (that is, if the left coordinate is less than 8192), `ShowDialogItem` does nothing.

The `ShowDialogItem` function adds the rectangle that contained the item to the update region so that it will be drawn. Note that if the item is a control you define in a control (' `CNTL` ') resource, the rectangle added to the update region is the rectangle defined in the control resource, not the display rectangle defined in the item list resource. If the item is an edit text item, `ShowDialogItem` activates it by calling the TextEdit function `TEActivate`.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Dialogs.h`

SizeDialogItem

Sizes a dialog item.

```
OSErr SizeDialogItem (
   DialogRef inDialog,
   DialogItemIndex inItemNo,
   SInt16 inWidth,
   SInt16 inHeight
);
```

Parameters

inDialog

A pointer to the dialog box containing the item to be resized.

inItemNo

The position of the item in the dialog box's item list resource use `FindDialogItem` (page 849) to determine this value.

inWidth

The new width (in pixels) of the dialog item's control rectangle.

inHeight
The new height (in pixels) of the dialog item's control rectangle.

Return Value
A result code. See "Dialog Manager Result Codes" (page 916).

Discussion
The `SizeDialogItem` function resizes a dialog item to a specified size. If the dialog item is a control, the control rectangle and the dialog item rectangle (maintained by the Dialog Manager) are always the same.

Version Notes
This function is available with Appearance Manager 1.0 and later.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

StandardAlert

Displays a standard alert box.

```
OSErr StandardAlert (
   AlertType inAlertType,
   ConstStr255Param inError,
   ConstStr255Param inExplanation,
   const AlertStdAlertParamRec *inAlertParam,
   SInt16 *outItemHit
);
```

Parameters
inAlertType
A constant indicating the type of alert box you wish to create; see "Alert Type Constants" (page 907).

inError
A pointer to a Pascal string containing the primary error text you wish to display.

inExplanation
A pointer to a Pascal string containing the secondary text you wish to display; secondary text is displayed in the small system font. Pass `null` to indicate no secondary text.

inAlertParam
A pointer to the standard alert structure; see `AlertStdAlertParamRec` (page 897). Pass `null` to specify that you do not wish to your alert box to incorporate any of the features that the standard alert structure provides.

outItemHit
A pointer to a signed 16-bit integer value. On return, the value indicates the alert button pressed; see "Alert Button Constants" (page 904).

Return Value
A result code. See "Dialog Manager Result Codes" (page 916).

Discussion
The `StandardAlert` function displays an alert box based on the values you pass it. You can pass the error text you wish displayed in the `error` and `explanation` parameters, and customize the alert button text by filling in the appropriate fields of the standard alert structure passed in the `inAlertParam` parameter.

`StandardAlert` automatically resizes the height of a dialog box to fit all static text. It ignores alert stages and therefore provides no corresponding alert sounds.

Special Considerations

This function is available with Appearance Manager 1.0 and later.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
BSDLLCTest
QTMetaData

Declared In
`Dialogs.h`

StdFilterProc

Handles standard event filtering for a dialog box.

```
Boolean StdFilterProc (
   DialogRef theDialog,
   EventRecord *event,
   DialogItemIndex *itemHit
);
```

Parameters

`theDialog`
On input, a pointer to a dialog structure for an alert box or a modal dialog box.

`event`
On output, a pointer to an event structure returned by an Event Manager function such as `WaitNextEvent`.

`itemHit`
On output, a pointer to a short integer. `StdFilterProc` returns a number corresponding to the position of an item in the item list resource for the alert or modal dialog box.

Return Value
A Boolean value representing whether the standard filter proc handled the event. `true` means handled; otherwise `false`.

Discussion
To use the standard filter function from within your own filter function, you can call `GetStdFilterProc` (page 860) , then dispatch the event to the standard filter function yourself; or you can call `StdFilterProc`, which performs both steps for you. Calling `StdFilterProc` is equivalent to calling `GetStdFilterProc` (page 860) and then calling `ModalFilterProcPtr` (page 891).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

StopAlert

Displays an alert box with a stop icon and/or plays an alert sound.

```
DialogItemIndex StopAlert (
   SInt16 alertID,
   ModalFilterUPP modalFilter
);
```

Parameters

alertID

The resource ID of an alert resource and extended alert resource. The resource ID of both types of resources must be identical. If the alert resource is missing, the Dialog Manager returns to your application without creating the requested alert. See `'alrx'` for a description of the extended alert resource.

modalFilter

A universal procedure pointer for a filter function that responds to events not handled by the `ModalDialog` (page 865) function. If you set this parameter to `null`, the Dialog Manager uses the standard event filter function.

Return Value
If no stop alert box is to be drawn at the current alert stage, `StopAlert` returns –1 otherwise, it creates and displays the alert box and returns the item number of the control selected by the user see "Alert Button Constants" (page 904). See the description of the `DialogItemIndex` data type.

Discussion
The `StopAlert` function displays an alert box with a stop icon in its upper-left corner or, if appropriate for the alert stage, plays an alert sound instead of or in addition to displaying the alert box.

The `StopAlert` function is the same as the `Alert` (page 834) function except that, before drawing the items in the alert box, `StopAlert` draws the stop icon in the upper-left corner. The stop icon has resource ID 0, which you can also specify with the constant `stopIcon` . By default, the Dialog Manager uses the standard stop icon from the System file. You can change this icon by providing your own `'ICON'` resource with resource ID 0.

Use a stop alert to inform the user that a problem or situation is so serious that the action cannot be completed. Stop alerts typically have only a single button (OK), because all the user can do is acknowledge that the action cannot be completed.

Your application should never draw its own default rings or alert icons. Prior to Mac OS 8, the `StopAlert` function would only redraw the alert icon and default button ring once and never redraw them on an update event. However, when Appearance is available, alert icons and default rings do redraw when you call `StopAlert`.

See also the functions `NoteAlert` (page 874) and `CautionAlert` (page 839).

Version Notes
This function was changed with Appearance Manager 1.0 to support the extended alert (`'alrx'`) resource.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

UpdateDialog

Redraws the update region of a specified dialog box.

```
void UpdateDialog (
   DialogRef theDialog,
   RgnHandle updateRgn
);
```

Parameters

theDialog
A pointer to a dialog structure.

updateRgn
A handle to the window region that needs to be updated.

Discussion
The `UpdateDialog` function redraws only the region in a dialog box specified in the `updateRgn` parameter. Your application generally should not use `UpdateDialog`. The Dialog Manager generally handles update events in alert and dialog boxes. `Alert` (page 834). `StopAlert` (page 889) , `NoteAlert` (page 874) , and `CautionAlert` (page 839) handle update events on their own.

Instead of drawing the entire contents of the specified dialog box, `UpdateDialog` draws only the items in the specified update region. You can use `UpdateDialog` in response to an update event, and you should usually bracket it by calls to the Window Manager functions `BeginUpdate` and `EndUpdate`. `UpdateDialog` uses the QuickDraw function `SetPort` to make the dialog box the current graphics port.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

Callbacks by Task

Accessing and Modifying Low-Memory Data

`UserItemProcPtr` (page 895)

ModalFilterProcPtr (page 891)

SoundProcPtr (page 894)
Defines a pointer to your sound callback function.

Miscellaneous

ModalFilterYDProcPtr (page 893)

QTModelessCallbackProcPtr (page 894)

Callbacks

ModalFilterProcPtr

```
typedef Boolean (*ModalFilterProcPtr)
(
    DialogRef theDialog,
    EventRecord * theEvent,
    DialogItemIndex * itemHit
);
```

If you name your function `MyModalFilterProc`, you would declare it like this:

```
Boolean MyModalFilterProc (
    DialogRef theDialog,
    EventRecord * theEvent,
    DialogItemIndex * itemHit
);
```

Parameters

theDialog
A pointer to a dialog structure for an alert box or a modal dialog box.

theEvent
A pointer to an event structure returned by an Event Manager function such as `WaitNextEvent`.

itemHit
A pointer to a short integer. Your event filter function should return a number corresponding to the position of an item in the item list resource for the alert or modal dialog box.

Return Value

A Boolean value. After receiving an event that it does not handle, your function should return `false`. When your function returns `false`, `ModalDialog` handles the event, which you pass in the parameter `theEvent`. (Your function can also change the event to simulate a different event and return `false`, which passes the event to the Dialog Manager for handling.) If your function does handle the event, your function should return `true`, and through the `itemHit` parameter return the number of the item that it handled.

Discussion
To supplement the Dialog Manager's ability to handle events in the Mac OS multitasking environment, you should provide an event filter function that the Dialog Manager calls whenever it displays alert boxes and modal dialog boxes. This function can receive all events that are sent to your application.

The `ModalDialog` (page 865) function and, in turn, the `Alert` (page 834) , `NoteAlert` (page 874) , `StopAlert` (page 889) , and `CautionAlert` (page 839) functions return the item number that your event filter function returns in the `itemHit` parameter in their own itemHit parameters.

For alert and modal dialog boxes, the Dialog Manager provides a standard event filter function that checks whether

- the user has pressed the Enter or Return key and, if so, returns the item number of the default button
- the user has pressed the Escape key or Command-period and, if so, returns the item number of the Cancel button
- the cursor is over edit text in a dialog box, and optionally changes the cursor to an I-beam whenever this is the case

If the dialog box is movable modal and the `kDialogHandleMovable` bit is set, your filter function will receive all events (including apple events and update events) that your application receives.

Your own filter function should use the standard filter function to accomplish these tasks. To do so, you can call `GetStdFilterProc` (page 860) , and dispatch the event to the standard filter function yourself; or you can call `StdFilterProc` (page 888) , which obtains a `ModalFilterUPP` for the standard filter function and then dispatches the function.

Your event filter function should also perform the following tasks:

- update your windows in response to update events and return `false`. If you do not handle update events for all the windows in your application, other processes won't get time.
- return `false` for all events that your event filter function doesn't handle

You can also use the event filter function to test for and respond to keyboard equivalents and more complex events—for instance, the user dragging the cursor in an application- defined item. For example, if you provide an application-defined item that requires you to measure how long the user holds down the mouse button or how far the user drags the cursor, use the event filter function to handle events inside that item.

Movable modal dialog boxes receive all events (not just those masked by the Event message mask).

In all alert and dialog boxes, any buttons that are activated by key sequences should highlight to indicate which item has been selected. Use the Control Manager function `HiliteControl` to highlight a button for 8 ticks, long enough to be noticeable but not so long as to be annoying. The Control Manager performs this action whenever users click a button, and your application should do this whenever the user presses the keyboard equivalent of a button click.

For modal dialog boxes that contain edit text items, your application should handle menu bar access to allow use of your Edit menu and its Cut, Copy, Paste, Clear, and Undo commands. Your event filter function should then test for and handle clicks in your Edit menu and keyboard equivalents for the appropriate commands in your Edit menu. Your application should respond by using the functions `DialogCut` (page 844) , `DialogCopy` (page 844) , `DialogPaste` (page 845) , and `DialogDelete` (page 845) to support the Cut, Copy, Paste, and Clear commands.

For an alert box, you specify a universal procedure pointer to your event filter function in a parameter that you pass to the `Alert` (page 834) , `StopAlert` (page 889) , `CautionAlert` (page 839) , and `NoteAlert` (page 874) functions. For a modal dialog box, specify a pointer to your event filter function in a parameter that you pass to `UpdateDialog`.

The Dialog Manager defines the data type `ModalFilterUPP` to identify this application-defined function:

```
typedef UniversalProcPtr ModalFilterUPP;
```

You typically use the `NewModalFilterProc` macro like this:

```
ModalFilterUPP myEventFilterProc;

myEventFilterProc = NewModalFilterProc(MyEventFilter);
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Dialogs.h`

ModalFilterYDProcPtr

```
typedef Boolean (*ModalFilterYDProcPtr)
(
    DialogRef theDialog,
    EventRecord * theEvent,
    short * itemHit,
    void * yourDataPtr
);
```

If you name your function `MyModalFilterYDProc`, you would declare it like this:

```
Boolean MyModalFilterYDProc (
    DialogRef theDialog,
    EventRecord * theEvent,
    short * itemHit,
    void * yourDataPtr
);
```

Parameters

`theDialog`

`theEvent`

`itemHit`

`yourDataPtr`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Dialogs.h`

QTModelessCallbackProcPtr

```
typedef void (*QTModelessCallbackProcPtr)
(
    EventRecord *theEvent,
    DialogRef theDialog,
    DialogItemIndex itemHit
);
```

If you name your function `MyQTModelessCallbackProc`, you would declare it like this:

```
void MyQTModelessCallbackProc (
    EventRecord *theEvent,
    DialogRef theDialog,
    DialogItemIndex itemHit
);
```

Parameters

theEvent

theDialog

itemHit

Carbon Porting Notes

This QuickTime function for manipulating dialog boxes is not supported in Carbon.

SoundProcPtr

Defines a pointer to your sound callback function.

```
typedef void (*SoundProcPtr) (
    SInt16 soundNumber
);
```

You should provide a sound callback function if you want the Dialog Manager to play sounds other than the system alert sound. If you name your function `MySoundProc`, you would declare it like this:

```
void MySoundProc (
    SInt16 soundNumber
);
```

Parameters

soundNumber

An integer from 0 to 3, representing the four possible alert stages.

Return Value

Discussion

For each of the four alert stages that can be reported in the `soundNumber` parameter, your function can emit any sound that you define. When the Dialog Manager calls your function, it passes 0 as the sound number for alert sounds specified by the silent constant in the alert resource. The Dialog Manager passes 1 for sounds specified by the `sound1` constant, 2 for sounds specified by the `sound2` constant, and 3 for sounds specified by the `sound3` constant.

The Dialog Manager defines the universal procedure pointer `SoundUPP` to identify this application-defined function:

```
typedef UniversalProcPtr SoundUPP; /
```

You typically use the `NewSoundProc` macro like this:

SoundUPP mySoundProc;

mySoundProc = NewSoundProc(MyAlertSound)

Special Considerations

When the Dialog Manager detects a click outside an alert box or a modal dialog box, it uses the Sound Manager function SysBeep to play the system alert sound. By changing settings in the Sound control panel, the user can select which sound to play as the system alert sound. For consistency with system software and other Macintosh applications, your sound function should call SysBeep whenever your sound function receives sound number 1 (which you can represent with the sound1 constant).

Version Notes
Not recommended with Appearance Manager 1.0 and later.

Carbon Porting Notes

Using custom sounds in dialog boxes is not supported in Carbon.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Dialogs.h`

UserItemProcPtr

```
typedef void (*UserItemProcPtr) (
    DialogRef theDialog,
    DialogItemIndex itemNo
);
```

If you name your function `MyUserItemProc`, you would declare it like this:

```
void MyUserItemProc (
    DialogRef theDialog,
    DialogItemIndex itemNo
);
```

Parameters

`theDialog`

On input, a pointer to the dialog structure for the dialog box containing an application-defined item. If your function can draw in more than one dialog box, this parameter tells your function which one to draw in.

itemNo

On input, a number corresponding to the position of an item in the item list resource for the specified dialog box. If your function draws more than one item, this parameter tells your function which one to draw.

Return Value

Discussion

When the Appearance Manager is available and an embedding hierarchy is established in a dialog box, you should provide the Control Manager user pane drawing function `MyUserPaneDrawCallback` instead of the user item drawing function `MyUserItemCallback` to draw an application-defined control (a dialog item becomes a control in a dialog box with an embedding hierarchy).

You can provide other user pane application-defined functions to hit test, track, perform idle processing, handle keyboard, activate, and deactivate event processing, handle keyboard focus, and set the background color or pattern in a user pane control.

When calling your draw function, the Dialog Manager sets the current port to the dialog box's graphics port. Normally, you create an invisible dialog box and then use the Window Manager function `ShowWindow` to display the dialog box.

Before you display the dialog box, use `SetDialogItem` (page 881) to install this function in the dialog structure. Before using `SetDialogItem`, you must first use `GetDialogItem` to obtain a handle to an item of type `userItem`.

If you enable the application-defined item that you draw with this function, `UpdateDialog` and `StdFilterProc` (page 888) return the item's number when the user clicks that item. If your application needs to respond to a user action more complex than this (for example, if your application needs to measure how long the user holds down the mouse or how far the user drags the cursor), your application must track the cursor itself. If you use `ModalDialog`, your event filter function must handle events inside the item; if you use `DialogSelect`, your application must handle events inside the item before handing events to `DialogSelect`.

The Dialog Manager defines the data type `UserItemUPP` to identify the universal procedure pointer for this application-defined function:

```
typedef UniversalProcPtr UserItemUPP;
```

You typically use the `NewUserItemProc` macro like this:

```
UserItemUPP myItemProc;

myItemProc = NewUserItemProc (MyItem);
```

Version Notes

This function is not recommended with Appearance Manager 1.0 and later.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Dialogs.h`

Data Types

AlertStdAlertParamRec

```
struct AlertStdAlertParamRec {
    Boolean movable;
    Boolean helpButton;
    ModalFilterUPP filterProc;
    ConstStringPtr defaultText;
    ConstStringPtr cancelText;
    ConstStringPtr otherText;
    SInt16 defaultButton;
    SInt16 cancelButton;
    UInt16 position;
};
typedef struct AlertStdAlertParamRec AlertStdAlertParamRec;
typedef AlertStdAlertParamRec * AlertStdAlertParamPtr;
```

Fields

`movable`
A Boolean value indicating whether or not the alert box is movable.

`helpButton`
A Boolean value indicating whether or not the alert includes a Help button.

`filterProc`
If the value in the `movable` field is `true` (alert is movable), then specify in this parameter a universal procedure pointer to an application-defined filter function that responds to events not handled by `ModalDialog` (page 865). If you do, all events will get routed to your application-defined event filter function for handling, even when your alert box window is in the background. If you set this parameter to `null`, the Dialog Manager uses the standard event filter function.

`defaultText`
Text for button in OK position; see "Alert Default Text Constants" (page 904). The button automatically sizes and positions itself in the alert box. To specify that the default button names should be used, pass `-1`. To indicate that no button should be displayed, pass `null`.

`cancelText`
Text for button in Cancel position; see "Alert Default Text Constants" (page 904). The button automatically sizes and positions itself in the alert box. To specify that the default button names should be used, pass `-1`. To indicate that no button should be displayed, pass `null`.

`otherText`
Text for button in leftmost position; see "Alert Default Text Constants" (page 904). The button automatically sizes and positions itself in the alert box. To specify that the default button names should be used, pass `-1`. To indicate that no button should be displayed, pass `null`.

`defaultButton`
Specifies which button acts as the default button; see "Alert Button Constants" (page 904).

`cancelButton`
Specifies which button acts as the Cancel button (can be 0); see "Alert Button Constants" (page 904).

`position`
The alert box position, as defined by a window positioning constant. In this structure, the constant `kWindowDefaultPosition` is equivalent to the constant `kWindowAlertPositionParentWindowScreen`.

Discussion
A standard alert structure of type `AlertStdAlertParamRec` can be used when you call the function `StandardAlert` (page 887) to customize the alert box. The `AlertStdAlertParamRec` type is available with Appearance Manager 1.0 and later.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Dialogs.h`

AlertStdCFStringAlertParamRec

Defines an alert or sheet.

```
struct AlertStdCFStringAlertParamRec {
    UInt32 version;
    Boolean movable;
    Boolean helpButton;
    CFStringRef defaultText;
    CFStringRef cancelText;
    CFStringRef otherText;
    SInt16 defaultButton;
    SInt16 cancelButton;
    UInt16 position;
    OptionBits flags;
};
typedef struct AlertStdCFStringAlertParamRec AlertStdCFStringAlertParamRec;
typedef AlertStdCFStringAlertParamRec * AlertStdCFStringAlertParamPtr;
```

Fields

`version`
The version of this parameter record. Set this field to `kStdCFStringAlertVersionOne`.

`movable`
A Boolean value indicating whether or not the alert is movable.

`helpButton`
A Boolean value indicating whether or not the alert contains a Help button.

`defaultText`
Text for button in the OK position. The button automatically sizes and positions itself in the alert box. To specify that the default button names should be used, pass `-1` (see "Alert Default Text Constants" (page 904) for values). To indicate that no button should be displayed, pass `null`

`cancelText`
Text for button in the Cancel position; see "Alert Default Text Constants" (page 904). The button automatically sizes and positions itself in the alert box. To specify that the default button names should be used, pass `-1` see "Alert Default Text Constants" (page 904) for values). To indicate that no button should be displayed, pass `null`.

`otherText`
Text for button in the other (leftmost) position; see "Alert Default Text Constants" (page 904). The button automatically sizes and positions itself in the alert box. To specify that the default button names should be used, pass `-1`. To indicate that no button should be displayed, pass `null`

`defaultButton`
Specifies which button acts as the default button; see "Alert Button Constants" (page 904).

`cancelButton`
Specifies which button acts as the default button; see "Alert Button Constants" (page 904).

`position`
The alert box position, as defined by a window positioning constant. In this structure, the constant `kWindowDefaultPosition` is equivalent to the constant `kWindowAlertPositionParentWindowScreen`. See the Window Manager Reference for other possible positioning constants.

`flags`
Options for this alert. See "Standard Alert and Sheet Option Flags" (page 915) for possible values.

Discussion
You pass this structure when calling `CreateStandardAlert` (page 842) or `CreateStandardSheet` (page 842).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Dialogs.h`

AlertTemplate

```
struct AlertTemplate {
    Rect boundsRect;
    SInt16 itemsID;
    StageList stages;
};
typedef struct AlertTemplate AlertTemplate;
typedef AlertTemplate * AlertTPtr;
```

Fields
`boundsRect`
`itemsID`
`stages`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Dialogs.h`

AlertType

```
typedef SInt16 AlertType;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Dialogs.h`

DialogItemIndex

```
typedef SInt16 DialogItemIndex;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Dialogs.h`

DialogItemIndexZeroBased

```
typedef SInt16 DialogItemIndexZeroBased;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Dialogs.h`

DialogItemType

```
typedef SInt16 DialogItemType;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Dialogs.h`

DialogPeek

```
typedef DialogRecord* DialogPeek;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Dialogs.h`

DialogRecord

```
struct DialogRecord {
    WindowRecord window;
    Handle items;
    TEHandle textH;
    SInt16 editField;
    SInt16 editOpen;
    SInt16 aDefItem;
};
typedef struct DialogRecord DialogRecord;
```

Fields

Discussion

A dialog structure of type `DialogRecord` is created whenever you call the functions `Alert` (page 834) or `GetNewDialog` (page 858). These functions incorporate information from your item list resource and your alert resource or dialog resource into this structure. Your application generally should not create a dialog structure or directly access its fields.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Dialogs.h`

DialogRef

```
typedef DialogPtr DialogRef;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Dialogs.h`

DialogTemplate

```
struct DialogTemplate {
    Rect boundsRect;
    SInt16 procID;
    Boolean visible;
    Boolean filler1;
    Boolean goAwayFlag;
    Boolean filler2;
    SInt32 refCon;
    SInt16 itemsID;
    Str255 title;
};
typedef struct DialogTemplate DialogTemplate;
typedef DialogTemplate * DialogTPtr;
```

Fields

`boundsRect`

`procID`

`visible`

`filler1`

`goAwayFlag`

`filler2`

`refCon`

`itemsID`

`title`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Dialogs.h`

ModalFilterUPP

```
typedef ModalFilterProcPtr ModalFilterUPP;
```

Discussion
For more information, see the description of the ModalFilterUPP () callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Dialogs.h`

ModalFilterYDUPP

```
typedef ModalFilterYDProcPtr ModalFilterYDUPP;
```

Discussion
For more information, see the description of the ModalFilterYDUPP () callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Dialogs.h`

QTModelessCallbackUPP

```
typedef QTModelessCallbackProcPtr QTModelessCallbackUPP;
```

SoundUPP

```
typedef SoundProcPtr SoundUPP;
```

Discussion
For more information, see the description of the SoundUPP () callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Dialogs.h`

StageList

```
typedef SInt16 StageList;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Dialogs.h`

UserItemUPP

```
typedef UserItemProcPtr UserItemUPP;
```

Discussion
For more information, see the description of the UserItemUPP () callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Dialogs.h`

Constants

Alert Button Constants

Define standard button types for alerts and sheets.

```
enum {
    kAlertStdAlertOKButton = 1,
    kAlertStdAlertCancelButton = 2,
    kAlertStdAlertOtherButton = 3,
    kAlertStdAlertHelpButton = 4
};
```

Constants

`kAlertStdAlertOKButton`
The OK button. The default text for this button is “OK”.
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`kAlertStdAlertCancelButton`
The Cancel button (optional). The default text for this button is “Cancel”.
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`kAlertStdAlertOtherButton`
A third button (optional). The default text for this button is “Don’t Save”.
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`kAlertStdAlertHelpButton`
The Help button (optional).
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

Discussion

You can use these constants in the `defaultButton` and `cancelButton` fields in the standard alert structure to specify which buttons act as the default and Cancel buttons in the standard alert structure. These constants are also returned in the `itemHit` parameter of `StandardAlert` (page 887). Alert button constants are available with Appearance Manager 1.0 and later.

Alert Default Text Constants

Defines the default text for alerts and sheets.

```
enum {
    kAlertDefaultOKText = -1,
    kAlertDefaultCancelText = -1,
    kAlertDefaultOtherText = -1
};
```

Constants

`kAlertDefaultOKText`

The default text for the default (right) button is "OK" on an English system. The text will vary depending upon the localization of the user's system. Use this constant in the `defaultText` field of the standard alert structure.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`kAlertDefaultCancelText`

The default text for the Cancel (middle) button is "Cancel" on an English system. The text will vary depending upon the localization of your system. Use this constant in the `cancelText` field of the standard alert structure.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`kAlertDefaultOtherText`

The default text for the third (leftmost) button is "Don't Save" for an English system. The text will vary depending upon the localization of the user's system. Use this constant in the `otherText` field of the standard alert structure.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

Discussion

You can use these constants in the `defaultText`, `cancelText`, and `otherText` fields of the standard alert structure to specify the default text for the OK, Cancel, and Don't Save buttons. Alert default text constants are available with Appearance Manager 1.0 and later.

Alert Feature Flag Constants

```
enum {
    kAlertFlagsUseThemeBackground = (1 << 0),
    kAlertFlagsUseControlHierarchy = (1 << 1),
    kAlertFlagsAlertIsMovable = (1 << 2),
    kAlertFlagsUseThemeControls = (1 << 3)
};
```

Constants

`kAlertFlagsUseThemeBackground`

If this bit (bit 0) is set, the Dialog Manager sets the alert box's background color or pattern.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`kAlertFlagsUseControlHierarchy`
If this bit (bit 1) is set, the Dialog Manager creates a root control in the alert box and establishes an embedding hierarchy. Any alert items become controls once the embedding hierarchy is established.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`kAlertFlagsAlertIsMovable`
If this bit (bit 2) is set, the alert box is movable modal. The Dialog Manager handles movable modal behavior such as dragging the alert box by its title bar or switching out of the application by clicking in another one.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`kAlertFlagsUseThemeControls`
If this bit (bit 3) is set, the Dialog Manager creates Appearance-compliant controls in your alert box. Otherwise, push buttons, checkboxes, and radio buttons will be displayed in their pre-Appearance forms when systemwide Appearance is off.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

Discussion
You can set these bits in the alert flags field of the extended alert resource to specify the alert box's Appearance-compliant features. Alert feature flag constants are available with Appearance Manager 1.0 and later.

Alert Icon Resource ID Constants

```
enum {
    kStopIcon = 0,
    kNoteIcon = 1,
    kCautionIcon = 2,
    stopIcon = kStopIcon,
    noteIcon = kNoteIcon,
    cautionIcon = kCautionIcon
};
```

Constants
`kStopIcon`
Resource ID for the standard stop icon.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`kNoteIcon`
Resource ID for the standard note icon.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`kCautionIcon`
Resource ID for the standard caution icon.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`stopIcon`
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`noteIcon`
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`cautionIcon`
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

Discussion
You can pass these constants in the `alertID` parameter of `StopAlert` (page 889) , `NoteAlert` (page 874), and `CautionAlert` (page 839) to specify the resource ID of the alert box icon you wish displayed.

Alert Type Constants

```
enum {
    kAlertStopAlert = 0,
    kAlertNoteAlert = 1,
    kAlertCautionAlert = 2,
    kAlertPlainAlert = 3
};
```

Constants
`kAlertStopAlert`
Stop alert box.
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`kAlertNoteAlert`
Note alert box.
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`kAlertCautionAlert`
Caution alert box.
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`kAlertPlainAlert`
Alert box with no icon.
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

Discussion
You can pass constants of type `AlertType` in the `inAlertType` parameter of `StandardAlert` (page 887) to specify the type of alert box you wish to create. Alert type constants are available with Appearance Manager 1.0 and later.

ctrlItem

```
enum {
    ctrlItem = 4,
    btnCtrl = 0,
    chkCtrl = 1,
    radCtrl = 2,
    resCtrl = 3,
    statText = 8,
    editText = 16,
    iconItem = 32,
    picItem = 64,
    userItem = 0,
    itemDisable = 128
};
```

Constants

`ctrlItem`
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`btnCtrl`
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`chkCtrl`
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`radCtrl`
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`resCtrl`
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`statText`
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`editText`
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`iconItem`
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`picItem`
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`userItem`
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`itemDisable`
Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

Dialog Feature Flag Constants

```
enum {
    kDialogFlagsUseThemeBackground = (1 << 0),
    kDialogFlagsUseControlHierarchy = (1 << 1),
    kDialogFlagsHandleMovableModal = (1 << 2),
    kDialogFlagsUseThemeControls = (1 << 3)
};
```

Constants

`kDialogFlagsUseThemeBackground`
If this bit (bit 0) is set, the Dialog Manager sets the dialog box's background color or pattern.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`kDialogFlagsUseControlHierarchy`
If this bit (bit 1) is set, the Dialog Manager creates a root control in the dialog box and establishes an embedding hierarchy. Any dialog items become controls once the embedding hierarchy is established.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`kDialogFlagsHandleMovableModal`
If this bit (bit 2) is set, and the dialog box is a movable modal (specify the `kWindowMovableModalDialogProc` window definition ID), the Dialog Manager handles movable modal behavior such as dragging a dialog box by its title bar or switching out of the application by clicking in another one.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`kDialogFlagsUseThemeControls`
If this bit (bit 3) is set, the Dialog Manager creates Appearance-compliant controls in the dialog box directly. Otherwise, push buttons, checkboxes, and radio buttons will be displayed in their pre-Appearance forms when systemwide Appearance is off.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

Discussion
You can set these bits in the dialog flags field of the extended dialog resource or pass them in the `inFlags` parameter of `NewFeaturesDialog` (page 871) to specify the dialog box's Appearance-compliant features. Dialog feature flag constants are available with Appearance Manager 1.0 and later.

Dialog Font Flag Constants

```
enum {
    kDialogFontNoFontStyle = 0,
    kDialogFontUseFontMask = 0x0001,
    kDialogFontUseFaceMask = 0x0002,
    kDialogFontUseSizeMask = 0x0004,
    kDialogFontUseForeColorMask = 0x0008,
    kDialogFontUseBackColorMask = 0x0010,
    kDialogFontUseModeMask = 0x0020,
    kDialogFontUseJustMask = 0x0040,
    kDialogFontUseAllMask = 0x00FF,
    kDialogFontAddFontSizeMask = 0x0100,
    kDialogFontUseFontNameMask = 0x0200,
    kDialogFontAddToMetaFontMask = 0x0400
};
```

Constants

`kDialogFontNoFontStyle`
If the `kDialogFontNoFontStyle` constant is used, no font style information is applied.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`kDialogFontUseFontMask`
If the `kDialogFontUseFontMask` flag (bit 0) is set, the font ID specified in the Font ID field of the dialog font table is applied.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`kDialogFontUseFaceMask`
If the `kDialogFontUseFaceMask` flag (bit 1) is set, the font style specified in the Style field of the dialog font table is applied.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`kDialogFontUseSizeMask`
If the `kDialogFontUseSizeMask` flag (bit 2) is set, the font size specified in the Font Size field of the dialog font table is applied.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`kDialogFontUseForeColorMask`
If the `kDialogFontUseForeColorMask` flag (bit 3) is set, the text color specified in the Text Color field of the dialog font table is applied. This flag only applies to static text controls.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`kDialogFontUseBackColorMask`
If the `kDialogFontUseBackColorMask` flag (bit 4) is set, the background color specified in the Background Color field of the dialog font table is applied. This flag only applies to static text controls.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`kDialogFontUseModeMask`
If the `kDialogFontUseModeMask` flag (bit 5) is set, the text mode specified in the Text Mode field of the dialog font table is applied.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`kDialogFontUseJustMask`
If the `kDialogFontUseJustMask` flag (bit 6) is set, the text justification specified in the Justification field of the dialog font table is applied.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`kDialogFontUseAllMask`
If the `kDialogFontUseAllMask` constant is used, all flags in this mask will be set except `kDialogFontAddFontSizeMask` and `kDialogFontUseFontNameMask`.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`kDialogFontAddFontSizeMask`
If the `kDialogFontAddFontSizeMask` flag (bit 8) is set, the Dialog Manager will add a specified font size to the existing font size indicated in the Font Size field of the dialog font table resource.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`kDialogFontUseFontNameMask`
If the `kDialogFontUseFontNameMask` flag (bit 9) is set, the Dialog Manager will use the string in the Font Name field for the font name instead of a font ID.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`kDialogFontAddToMetaFontMask`
Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

Discussion
You can set the following bits in the dialog font table resource to specify fields in the dialog font table that should be used. Dialog font flag constants are available with Appearance Manager 1.0 and later.

Dialog Item Constants

```
enum {
    kControlDialogItem = 4,
    kButtonDialogItem = kControlDialogItem | 0,
    kCheckBoxDialogItem = kControlDialogItem | 1,
    kRadioButtonDialogItem = kControlDialogItem | 2,
    kResourceControlDialogItem = kControlDialogItem | 3,
    kStaticTextDialogItem = 8,
    kEditTextDialogItem = 16,
    kIconDialogItem = 32,
    kPictureDialogItem = 64,
    kUserDialogItem = 0,
    kHelpDialogItem = 1,
    kItemDisableBit = 128
};
```

Constants

`kControlDialogItem`
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`kButtonDialogItem`
Standard button control.
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`kCheckBoxDialogItem`
Standard checkbox control.
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`kRadioButtonDialogItem`
Standard radio button control.
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`kResourceControlDialogItem`
Control defined in control resource.
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`kStaticTextDialogItem`
Static text item.
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`kEditTextDialogItem`
Edit text item.
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`kIconDialogItem`
Icon.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`kPictureDialogItem`
QuickDraw picture.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`kUserDialogItem`
Application-defined item.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`kHelpDialogItem`
Help balloon, as defined by the Help Manager.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`kItemDisableBit`
Add to disable any other constant, except `kHelpDialogItem`.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

Discussion
These constants are returned in the `itemType` parameter of `GetDialogItem` (page 852) and can be passed to `SetDialogItem` (page 881) and the dialog item list resource to specify dialog item type.

Dialog Item List Display Constants

Specify methods of appending new items to a dialog.

```
typedef SInt16 DITLMethod;
enum {
    overlayDITL = 0,
    appendDITLRight = 1,
    appendDITLBottom = 2
};
```

Constants
`overlayDITL`
Superimpose the appended items over the dialog box.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`appendDITLRight`
Position the items to the right of the dialog box and relative to its upper-right coordinate.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`appendDITLBottom`
Position the items below the dialog box and relative to its lower-left coordinate.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

Discussion
You can pass a constant value of type `DITLMethod` to the function `AppendDITL` (page 836) to specify how you want appended dialog items displayed.

kDialogFontUseThemeFontIDMask

```
enum {
    kDialogFontUseThemeFontIDMask = 0x0080
};
```

Constants
`kDialogFontUseThemeFontIDMask`
Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

kHICommandOther

```
enum {
    kHICommandOther = 'othr'
};
```

Constants
`kHICommandOther`
Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

kOkItemIndex

```
enum {
    kOkItemIndex = 1,
    kCancelItemIndex = 2
};
```

Constants
`kOkItemIndex`
Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`kCancelItemIndex`
Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

Standard Alert and Sheet Option Flags

Define flags used in the `AlertStdCFStringAlertParamRec` (page 898) structure.

```
enum {
    kStdAlertDoNotDisposeSheet = 1 << 0,
    kStdAlertDoNotAnimateOnDefault = 1 << 1,
    kStdAlertDoNotAnimateOnCancel = 1 << 2,
    kStdAlertDoNotAnimateOnOther = 1 << 3,
    kStdAlertDoNotCloseOnHelp     = 1 << 4
};
```

Constants

`kStdAlertDoNotDisposeSheet`

Do not dispose of the sheet window after closing it. This option allows the sheet to be used again when calling the Window Manager function `ShowSheetWindow`.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`kStdAlertDoNotAnimateOnDefault`

Do not animate hiding the sheet window when the user presses the default button.

Available in Mac OS X v10.1 and later.

Declared in `Dialogs.h`.

`kStdAlertDoNotAnimateOnCancel`

Do not animate hiding the sheet window when the user presses the Cancel button.

Available in Mac OS X v10.1 and later.

Declared in `Dialogs.h`.

`kStdAlertDoNotAnimateOnOther`

Do not animate hiding the sheet window when the user presses the other button.

Available in Mac OS X v10.1 and later.

Declared in `Dialogs.h`.

`kStdAlertDoNotCloseOnHelp`

Specifies that the alert stay up even after the user clicks the Help button. Normally, it would close immediately. It is not necessary to set this option for sheets, as they merely send the `HICommandHelp` command to the event target provided. When you specify this option, `RunStandardAlert` (page 877) returns with the Help button item in the `outItemHit` parameter, but the alert remains up. You can then perform whatever help function you wish and then call `RunStandardAlert` again.

Declared in `Dialogs.h`.

Available in Mac OS X 10.4 or later.

Standard Alert Structure Version Constant

Indicates the version of the `AlertStdCFStringAlertParamRec` (page 898) structure.

```
enum {
    kStdCFStringAlertVersionOne = 1
};
```

Constants

`kStdCFStringAlertVersionOne`
First version. Pass this into the `version` field of the `AlertStdCFStringAlertParamRec` structure.
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

kStdOkItemIndex

```
enum {
    kStdOkItemIndex = 1,
    kStdCancelItemIndex = 2,
    ok = kStdOkItemIndex,
    cancel = kStdCancelItemIndex
};
```

Constants

`kStdOkItemIndex`
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`kStdCancelItemIndex`
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`ok`
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`cancel`
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

Result Codes

The result codes defined for the Dialog Manager are listed below.

Result Code	Value	Description
`dialogNoTimeoutErr`	-5640	No timeout has been set for this dialog. Available in Mac OS X v10.0 and later.

Gestalt Constants

You can check for version and feature availability information by using the Dialog Manager selectors defined in the Gestalt Manager. For more information, see *Gestalt Manager Reference*.

Drag Manager Reference

Framework: Carbon/Carbon.h

Declared in Drag.h

Overview

The Drag Manager facilitates dragging objects within the Macintosh user interface. The Drag Manager provides functions that handle the user interface for dragging an object from, within, or to one of your application's windows. The Drag Manager can be used whenever an object is dragged within your application.

Use the Drag Manager if you want your users to be able to drag items within your own application's windows or between those of your application and other applications. You can also use the Drag Manager to allow the user of your application to drag selections of your documents to the Finder to create "clippings" from your documents and to allow selections from other applications to be dragged directly into your documents.

Functions by Task

Installing and Removing Drag Handlers

`InstallReceiveHandler` (page 940) Deprecated in Mac OS X v10.5
Installs a receive handler function for one or all of your application's windows.

`InstallTrackingHandler` (page 941) Deprecated in Mac OS X v10.5
Installs a tracking handler function for one or all of your application's windows.

`RemoveReceiveHandler` (page 947) Deprecated in Mac OS X v10.5
Removes a receive handler function from one or all of your application's windows.

`RemoveTrackingHandler` (page 947) Deprecated in Mac OS X v10.5
Removes a tracking handler function from one or all of your application's windows.

Creating and Disposing of Drag References

`NewDrag` (page 944)
Creates a new drag reference for your application to use with the Drag Manager.

`DisposeDrag` (page 926)
Disposes of a drag reference and its associated data.

Adding Drag Item Flavors

`AddDragItemFlavor` (page 923) Deprecated in Mac OS X v10.5
Adds a flavor to a drag item, creating a new item if necessary.

`SetDragItemFlavorData` (page 953) Deprecated in Mac OS X v10.5
Sets the data or part of the data contained within an existing flavor.

Providing Drag Callback Functions

`SetDragInputProc` (page 952)
Sets the drag input function for the Drag Manager to use with a particular drag.

`SetDragDrawingProc` (page 948) Deprecated in Mac OS X v10.5
Sets the drag drawing function for the Drag Manager to use with a particular drag.

`SetDragSendProc` (page 955) Deprecated in Mac OS X v10.5
Sets the send data function for the Drag Manager to use with a particular drag.

Setting the Drag Image

`SetDragImageWithCGImage` (page 951)
Associates a Core Graphics image with a drag reference.

`SetDragImage` (page 950) Deprecated in Mac OS X v10.4
Associates an image with a drag reference. (Deprecated. Use `SetDragImageWithCGImage` (page 951) instead.)

Altering the Behavior of a Drag

`ChangeDragBehaviors` (page 924)
Changes the behavior of a drag.

Performing a Drag

`TrackDrag` (page 958)
Drags an item or collection of items from your application.

Getting Drag Item Information

`GetDragItemBounds` (page 932)
Gets the bounding rectangle of a drag item.

`SetDragItemBounds` (page 952)
Sets the bounding rectangle of a drag item.

`CountDragItemFlavors` (page 925) Deprecated in Mac OS X v10.5
Gets the number of flavors that are contained within a drag item.

`CountDragItems` (page 925) Deprecated in Mac OS X v10.5
Gets the number of drag items that are contained in a drag reference.

`GetDragItemReferenceNumber` (page 933) Deprecated in Mac OS X v10.5
Gets the reference number of a specific item in a drag reference.

`GetFlavorData` (page 936) Deprecated in Mac OS X v10.5
Gets all or part of the data for a specific flavor in a drag item.

`GetFlavorDataSize` (page 937) Deprecated in Mac OS X v10.5
Gets the size of the data for a specific flavor in a drag item.

`GetFlavorFlags` (page 937) Deprecated in Mac OS X v10.5
Gets the flags for a specific flavor in a drag item.

`GetFlavorType` (page 938) Deprecated in Mac OS X v10.5
Gets the type of a specific flavor in a drag item.

Getting and Setting the Drop Location

`GetDropLocation` (page 935) Deprecated in Mac OS X v10.5
Gets the Apple Event descriptor of the drop location.

`GetStandardDropLocation` (page 939) Deprecated in Mac OS X v10.5
Gets the standard drop location set by the receiver of a drag.

`SetDropLocation` (page 956) Deprecated in Mac OS X v10.5
Sets the Apple Event descriptor for the drop location of a drag.

`SetStandardDropLocation` (page 956) Deprecated in Mac OS X v10.5
Used by the receiver of a drag to set the standard drop location for a drag.

Getting Drag Status Information

`GetDragAttributes` (page 930)
Gets the current set of drag attribute flags.

`GetDragMouse` (page 934)
Gets the current mouse and pinned mouse locations.

`SetDragMouse` (page 954)
Sets the current pinned mouse location.

`GetDragOrigin` (page 934)
Gets the `mouseDown` parameter location that started the given drag.

`GetDragModifiers` (page 933)
Gets the current set of keyboard modifiers.

Accessing Drag Actions

`GetDragAllowableActions` (page 929)
Returns the actions that the drag receiver may take on the data within a drag.

`SetDragAllowableActions` (page 948)
Sets the actions that the drag receiver may take on the data within a drag.

`GetDragDropAction` (page 931)
Returns the action performed by the receiver of the drag.

`SetDragDropAction` (page 949)
Sets the action performed by the receiver of the drag.

Highlighting a Drag

`DragPostScroll` (page 928) Deprecated in Mac OS X v10.5
Restores the drag highlight after scrolling part of your window.

`DragPreScroll` (page 929) Deprecated in Mac OS X v10.5
Prepares your window or pane for scrolling.

`GetDragHiliteColor` (page 931) Deprecated in Mac OS X v10.5
Returns the drag highlight color for a window.

`HideDragHilite` (page 939) Deprecated in Mac OS X v10.5
Removes highlighting created with the `ShowDragHilite` function.

`ShowDragHilite` (page 957) Deprecated in Mac OS X v10.5
Highlights an area of your window during a drag.

`UpdateDragHilite` (page 959) Deprecated in Mac OS X v10.5
Updates the portion of the drag highlight that was drawn over by your application.

Drag Manager Utilities

`WaitMouseMoved` (page 960)
Returns `true` if a mouse movement is the beginning of a drag.

`ZoomRects` (page 960) Deprecated in Mac OS X v10.5
Animates a rectangle into a second rectangle.

`ZoomRegion` (page 961) Deprecated in Mac OS X v10.5
Animates a region's outline from one screen location to another.

Creating, Calling, and Deleting Universal Procedure Pointers

`NewDragInputUPP` (page 945)
Creates a new universal procedure pointer (UPP) to a drag input callback.

`DisposeDragInputUPP` (page 927)
Disposes of the universal procedure pointer (UPP) to a drag input callback.

`InvokeDragInputUPP` (page 942)
Calls your drag input callback.

`DisposeDragDrawingUPP` (page 926) Deprecated in Mac OS X v10.5
Disposes of the universal procedure pointer (UPP) to a drag drawing callback.

`DisposeDragReceiveHandlerUPP` (page 927) Deprecated in Mac OS X v10.5
Disposes of the universal procedure pointer (UPP) to a drag receive handler.

`DisposeDragSendDataUPP` (page 927) Deprecated in Mac OS X v10.5
Disposes of the universal procedure pointer (UPP) to a drag send data callback.

DisposeDragTrackingHandlerUPP (page 928) Deprecated in Mac OS X v10.5
: Disposes of the universal procedure pointer (UPP) to a drag tracking handler.

InvokeDragDrawingUPP (page 942) Deprecated in Mac OS X v10.5
: Calls your drag drawing callback.

InvokeDragReceiveHandlerUPP (page 943) Deprecated in Mac OS X v10.5
: Calls your drag receive handler.

InvokeDragSendDataUPP (page 943) Deprecated in Mac OS X v10.5
: Calls your drag send data callback.

InvokeDragTrackingHandlerUPP (page 944) Deprecated in Mac OS X v10.5
: Calls your drag tracking handler.

NewDragDrawingUPP (page 944) Deprecated in Mac OS X v10.5
: Creates a new universal procedure pointer (UPP) to a drag drawing callback.

NewDragReceiveHandlerUPP (page 945) Deprecated in Mac OS X v10.5
: Creates a new universal procedure pointer (UPP) to a drag receive handler.

NewDragSendDataUPP (page 946) Deprecated in Mac OS X v10.5
: Creates a new universal procedure pointer (UPP) to a drag send data callback.

NewDragTrackingHandlerUPP (page 946) Deprecated in Mac OS X v10.5
: Creates a new universal procedure pointer (UPP) to a drag tracking handler.

Functions

AddDragItemFlavor

Adds a flavor to a drag item, creating a new item if necessary. (Deprecated in Mac OS X v10.5.)

```
OSErr AddDragItemFlavor (
   DragRef theDrag,
   DragItemRef theItemRef,
   FlavorType theType,
   const void *dataPtr,
   Size dataSize,
   FlavorFlags theFlags
);
```

Parameters

theDrag
: A drag reference.

theItemRef
: The drag item to add the flavor to. You create a new drag item by providing a unique item reference number here. You add a flavor to an existing item by using the same item reference number as in a previous call. You may use any item reference number when adding a flavor to an item. Item reference numbers do not need to be specified in order, nor must they be sequential. In many cases it is easiest to use index numbers as item reference numbers (1, 2, 3...). Item reference numbers are only used as unique "key" numbers for each item. Depending on your application, it might be easier to use your own internal memory addresses as item reference numbers (as long as each item being dragged has a unique item reference number).

theType

The data type of the flavor to add. This may be any four-character scrap type. You may use your application's signature for a unique type for internal use. You must add all of the drag item flavors to a drag item before calling the `TrackDrag` function. Once the `TrackDrag` function is called, receiving applications may not operate properly if new drag items or drag item flavors are added.

dataPtr

A pointer to the flavor data to add. Pass `NULL` to defer the creation of a particular data type until a receiver has specifically requested it. If you pass `NULL`, a promise is added to the drag; when the flavor is requested, the Drag Manager calls the drag's send data function to get the data from your application.

Note that this method of setting promises differs from the method of setting Scrap Manager promises. See the Scrap Manager function `PutScrapFlavor` for more information.

dataSize

The size, in bytes, of the flavor data to add. If you pass `NULL` in the `dataPtr` parameter, the value in this parameter is ignored.

theFlags

The set of attributes to set for this flavor.

Return Value
A result code. See "Drag Manager Result Codes" (page 986).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Drag.h`

ChangeDragBehaviors

Changes the behavior of a drag.

```
OSErr ChangeDragBehaviors (
   DragRef theDrag,
   DragBehaviors inBehaviorsToSet,
   DragBehaviors inBehaviorsToClear
);
```

Parameters

theDrag

A drag reference.

inBehaviorsToSet

A value indicating the new behavior of the drag. See "Drag Behaviors" (page 974) for a description of the values you can use in this parameter.

inBehaviorsToClear

A value indicating which existing behavior, if any, should be cleared. See "Drag Behaviors" (page 974) for a description of the values you can use in this parameter.

Return Value
A result code. See "Drag Manager Result Codes" (page 986).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Drag.h`

CountDragItemFlavors

Gets the number of flavors that are contained within a drag item. (Deprecated in Mac OS X v10.5.)

```
OSErr CountDragItemFlavors (
   DragRef theDrag,
   DragItemRef theItemRef,
   UInt16 *numFlavors
);
```

Parameters

theDrag
The drag reference.

theItemRef
An item reference number.

numFlavors
On return, a pointer to the number of flavors in the specified drag item. When the `CountDragItemFlavors` function is called by an application other than the sender, the flavors that are marked with the `flavorSenderOnly` flag are not included in the count.

Return Value
A result code. See "Drag Manager Result Codes" (page 986).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Drag.h`

CountDragItems

Gets the number of drag items that are contained in a drag reference. (Deprecated in Mac OS X v10.5.)

```
OSErr CountDragItems (
   DragRef theDrag,
   UInt16 *numItems
);
```

Parameters

theDrag
The drag reference.

numItems
On return, a pointer to the number of drag items in the specified drag reference.

Return Value
A result code. See "Drag Manager Result Codes" (page 986).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Drag.h`

DisposeDrag

Disposes of a drag reference and its associated data.

```
OSErr DisposeDrag (
   DragRef theDrag
);
```

Parameters

theDrag
The drag reference of the drag object to dispose of. If the drag reference contains any drag item flavors, the memory associated with the drag item flavors is disposed of as well.

Return Value
A result code. See "Drag Manager Result Codes" (page 986).

Discussion
You should call the `DisposeDrag` function after a drag has been performed using the `TrackDrag` function or if a drag reference was created but is no longer needed.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Drag.h`

DisposeDragDrawingUPP

Disposes of the universal procedure pointer (UPP) to a drag drawing callback. (Deprecated in Mac OS X v10.5.)

```
void DisposeDragDrawingUPP (
   DragDrawingUPP userUPP
);
```

Parameters

userUPP
The UPP to dispose of.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`Drag.h`

DisposeDragInputUPP

Disposes of the universal procedure pointer (UPP) to a drag input callback.

```
void DisposeDragInputUPP (
   DragInputUPP userUPP
);
```

Parameters

userUPP
The UPP to dispose of.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Drag.h`

DisposeDragReceiveHandlerUPP

Disposes of the universal procedure pointer (UPP) to a drag receive handler. (Deprecated in Mac OS X v10.5.)

```
void DisposeDragReceiveHandlerUPP (
   DragReceiveHandlerUPP userUPP
);
```

Parameters

userUPP
The UPP to dispose of.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`Drag.h`

DisposeDragSendDataUPP

Disposes of the universal procedure pointer (UPP) to a drag send data callback. (Deprecated in Mac OS X v10.5.)

```
void DisposeDragSendDataUPP (
   DragSendDataUPP userUPP
);
```

Parameters

userUPP
The UPP to dispose of.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`Drag.h`

DisposeDragTrackingHandlerUPP

Disposes of the universal procedure pointer (UPP) to a drag tracking handler. (Deprecated in Mac OS X v10.5.)

```
void DisposeDragTrackingHandlerUPP (
   DragTrackingHandlerUPP userUPP
);
```

Parameters

userUPP
The UPP to dispose of.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`Drag.h`

DragPostScroll

Restores the drag highlight after scrolling part of your window. (Deprecated in Mac OS X v10.5.)

```
OSErr DragPostScroll (
   DragRef theDrag
);
```

Parameters

theDrag
The drag reference.

Return Value
A result code. See "Drag Manager Result Codes" (page 986).

Discussion
The `DragPostScroll` function restores the drag highlight after scrolling part of your window. This function must be called following each call to the `DragPreScroll` function and any subsequent scrolling.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Drag.h`

DragPreScroll

Prepares your window or pane for scrolling. (Deprecated in Mac OS X v10.5.)

```
OSErr DragPreScroll (
   DragRef theDrag,
   SInt16 dH,
   SInt16 dV
);
```

Parameters

theDrag

The drag reference.

dH

The horizontal distance you intend to scroll.

dV

The vertical distance you intend to scroll.

Return Value

A result code. See "Drag Manager Result Codes" (page 986).

Discussion

Removes any drag highlighting that would be scrolled away from the `hiliteFrame` specified to the `ShowDragHilite` function when scrolling part of your window while drag highlighting is showing. Use this function if you plan to scroll part of your window using the `ScrollRect` or `CopyBits` functions.

Scrolling part of your window may inadvertently move part of the drag highlighting with it. The `DragPreScroll` function is optimized to remove from the screen only the parts of the highlighting that will be scrolled away from the `hiliteFrame` region. After calling the `DragPreScroll` function with the `dH` and `dV` that you are going to scroll, you can then scroll your window followed by a call to the `DragPostScroll` function which redraws any necessary highlighting after the scroll.

If you use an offscreen port to draw your window into while scrolling, it is recommended that you simply use the `HideDragHilite` and `ShowDragHilite` functions to preserve drag highlighting in your offscreen port. The `DragScroll` functions are optimized for onscreen scrolling.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Drag.h`

GetDragAllowableActions

Returns the actions that the drag receiver may take on the data within a drag.

```
OSStatus GetDragAllowableActions (
   DragRef theDrag,
   DragActions *outActions
);
```

Parameters

theDrag

The drag reference.

outActions

A pointer to a field that specifies, on return, the allowable drag actions. See "Drag Actions" (page 979) for a description of the values that may be returned here.

Return Value

A result code. See "Drag Manager Result Codes" (page 986).

Discussion

The drag actions returned by the `GetDragAllowableActions` function are not actually requirements; they are highly recommended suggestions for operations that the drag receiver may perform. The drag sender sets the recommended actions for a drag using the `SetDragAllowableActions` (page 948) function. The drag actions returned by `GetDragAllowableActions` are always local to the caller's process.

Availability

Available in Mac OS X v10.1 and later.

Not available to 64-bit applications.

Declared In

`Drag.h`

GetDragAttributes

Gets the current set of drag attribute flags.

```
OSErr GetDragAttributes (
   DragRef theDrag,
   DragAttributes *flags
);
```

Parameters

theDrag

A drag reference.

flags

On return, a pointer to the drag attribute flags for the specified drag reference. See "Drag Attributes" (page 973) for a description of the values that may be returned here.

Return Value

A result code. See "Drag Manager Result Codes" (page 986).

Discussion

If the `GetDragAttributes` function is called during a drag, the current set of drag attributes is returned. If the `GetDragAttributes` function is called after a drag, the set of drag attributes that were set at drop time is returned.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
Drag.h

GetDragDropAction

Returns the action performed by the receiver of the drag.

```
OSStatus GetDragDropAction (
   DragRef theDrag,
   DragActions *outAction
);
```

Parameters

theDrag

The drag reference from which to retrieve the drop action.

outAction

A pointer to a field that, on return, specifies the action performed by the drag receiver. More than one action may be performed. See "Drag Actions" (page 979) for a description of the values that may be returned here.

Return Value
A result code. See "Drag Manager Result Codes" (page 986).

Availability
Available in Mac OS X v10.1 and later.
Not available to 64-bit applications.

Declared In
Drag.h

GetDragHiliteColor

Returns the drag highlight color for a window. (Deprecated in Mac OS X v10.5.)

```
OSErr GetDragHiliteColor (
   WindowRef window,
   RGBColor *color
);
```

Parameters

window

The window for which to return the drag highlight color.

color

On return, a pointer to the highlight color.

Return Value
A result code. See "Drag Manager Result Codes" (page 986).

Discussion
Use the `GetDragHiliteColor` function to determine the color the Drag Manager will use for a particular window. `GetDragHiliteColor` can safely be called when the `gestaltDragMgrHasImageSupport` bit is set in the `Gestalt` response to the selector `gestaltDragMgrAttr`. For more information on the `gestaltDragMgrAttr` selector, see *Inside Mac OS X: Gestalt Manager Reference*.

The Drag Manager chooses an appropriate color for highlighting, based on the color used for drag highlighting in the current Appearance Manager theme.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Drag.h`

GetDragItemBounds

Gets the bounding rectangle of a drag item.

```
OSErr GetDragItemBounds (
   DragRef theDrag,
   DragItemRef theItemRef,
   Rect *itemBounds
);
```

Parameters
`theDrag`
A drag reference.
`theItemRef`
The reference number of the drag item whose bounds you wish to obtain.
`itemBounds`
On return, a pointer to the bounding rectangle (relative to the current pinned mouse position) of the specified item in global coordinates.

Return Value
A result code. See "Drag Manager Result Codes" (page 986).

Discussion
You can use the `GetDragItemBounds` function in your tracking or receive handlers to determine the current or dropped location of each item in the drag.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Drag.h`

GetDragItemReferenceNumber

Gets the reference number of a specific item in a drag reference. (Deprecated in Mac OS X v10.5.)

```
OSErr GetDragItemReferenceNumber (
   DragRef theDrag,
   UInt16 index,
   DragItemRef *theItemRef
);
```

Parameters

theDrag

The drag reference.

index

The index of an item in a drag for which to get the reference.

theItemRef

On return, a pointer to the reference number of the item with the specified index.

Return Value

A result code. See "Drag Manager Result Codes" (page 986). If `index` is 0 or larger than the number of items in the drag, `GetDragItemReferenceNumber` returns the `badDragItemErr` result code.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Drag.h`

GetDragModifiers

Gets the current set of keyboard modifiers.

```
OSErr GetDragModifiers (
   DragRef theDrag,
   SInt16 *modifiers,
   SInt16 *mouseDownModifiers,
   SInt16 *mouseUpModifiers
);
```

Parameters

theDrag

A drag reference.

modifiers

A pointer to a variable that, on return, contains the current keyboard modifiers. You may pass `NULL` if you wish to disregard this value. The value will be 0 if the drag has not been started.

mouseDownModifiers

A pointer to a variable that, on return, contains the keyboard modifiers at the `mouseDown` parameter time. You may pass `NULL` if you wish to disregard this value. The value will be 0 if the drag has not been started.

mouseUpModifiers

A pointer to a variable that, on return, contains the keyboard modifiers at the `mouseUp` parameter time. You may pass `NULL` if you wish to disregard this value. The value will be 0 if the drag has not been completed.

Return Value
A result code. See "Drag Manager Result Codes" (page 986).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Drag.h`

GetDragMouse

Gets the current mouse and pinned mouse locations.

```
OSErr GetDragMouse (
   DragRef theDrag,
   Point *mouse,
   Point *globalPinnedMouse
);
```

Parameters

theDrag

A drag reference.

mouse

A pointer to a variable containing, on return, the current mouse location in global screen coordinates. You may pass `NULL` if you wish to ignore this value. The value will be (0, 0) if the drag is not yet used. After a drag completes, the drop location is returned.

globalPinnedMouse

A pointer to a variable containing, on return, the current pinned mouse location in global screen coordinates. You may pass `NULL` if you wish to ignore this value. The value will be (0, 0) if the drag is not yet used. After a drag completes, the drop location is returned.The pinned mouse location is the mouse location that is used to draw the drag region on the screen. The pinned mouse location is different from the mouse location when the cursor is being constrained in either dimension by a tracking handler.

Return Value
A result code. See "Drag Manager Result Codes" (page 986).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Drag.h`

GetDragOrigin

Gets the `mouseDown` parameter location that started the given drag.

```
OSErr GetDragOrigin (
   DragRef theDrag,
   Point *globalInitialMouse
);
```

Parameters

theDrag

A drag reference.

globalInitialMouse

A pointer to a variable that contains, on return, the `mouseDown` parameter location that started the drag, in global coordinates. The `mouseDown` location is returned whether or not the drag has completed.

Return Value
A result code. See "Drag Manager Result Codes" (page 986).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Drag.h`

GetDropLocation

Gets the Apple Event descriptor of the drop location. (Deprecated in Mac OS X v10.5.)

```
OSErr GetDropLocation (
   DragRef theDrag,
   AEDesc *dropLocation
);
```

Parameters

theDrag

A drag reference.

dropLocation

On return, a pointer to the Apple Event descriptor of the drop location. The drop location is only valid after the receiver has set the drop location by calling the `SetDropLocation` function. If the destination is in the Finder, the drop location will be an alias to the location in the file system that received the drag. If the receiver of the drag has not set a drop location by calling the `SetDropLocation` function, `typeNull` will be returned.

Return Value
A result code. See "Drag Manager Result Codes" (page 986).

Discussion
The `GetDropLocation` function may be called both during a drag as well as after a drag has completed.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Drag.h`

GetFlavorData

Gets all or part of the data for a specific flavor in a drag item. (Deprecated in Mac OS X v10.5.)

```
OSErr GetFlavorData (
   DragRef theDrag,
   DragItemRef theItemRef,
   FlavorType theType,
   void *dataPtr,
   Size *dataSize,
   UInt32 dataOffset
);
```

Parameters

theDrag

A drag reference.

theItemRef

The reference number of the drag item containing the flavor data.

theType

The flavor type of the flavor to get the data from.

dataPtr

A pointer to a data buffer. On return, the buffer contains the requested flavor data. Your application is responsible for allocating the memory for the flavor data and for setting the *dataSize* parameter to the number of bytes that you have allocated for the data.

dataSize

On input, a pointer to the size of the data (in bytes) that you have allocated memory for and wish to receive from the flavor. On return, a pointer to the actual number of bytes copied into the buffer specified by the `dataPtr` parameter.

If you specify a `dataSize` that is smaller than the amount of data in the flavor, the data is copied into your buffer and the `dataSize` parameter is unchanged. If you specify a `dataSize` that is larger than the amount of data in the flavor, only the amount of data in the flavor is copied into your buffer and the `dataSize` parameter contains, on return, the actual number of bytes copied. You have reached the end of the flavor's data when the `dataSize` parameter points to a number of bytes lower than you provided.

If you wish to receive the flavor data in smaller pieces than the entire size of the data, you can set the `dataSize` parameter to be as large as your buffer and call the `GetFlavorData` function multiple times while incrementing the `dataOffset` parameter by the size of your buffer. If the `dataOffset` parameter is larger than the amount of data contained within the flavor, 0 (zero) will be returned in the number pointed to by the `dataSize` parameter indicating that no data was copied into your buffer.

dataOffset

A pointer to the offset (in bytes) within the flavor structure at which to begin copying data.

Return Value
A result code. See "Drag Manager Result Codes" (page 986).

Discussion
You can first determine the size of a flavor's data by calling the `GetFlavorDataSize` function.

Note that calling the `GetFlavorData` function on a flavor that requires translation will force that translation to occur in order to return the data.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Drag.h`

GetFlavorDataSize

Gets the size of the data for a specific flavor in a drag item. (Deprecated in Mac OS X v10.5.)

```
OSErr GetFlavorDataSize (
   DragRef theDrag,
   DragItemRef theItemRef,
   FlavorType theType,
   Size *dataSize
);
```

Parameters

theDrag
A drag reference.

theItemRef
The reference number of the drag item containing the flavor.

theType
The flavor type for which to get the size of the data.

dataSize
On return, a pointer to the size of the data for the specified drag item flavor.

Return Value
A result code. See "Drag Manager Result Codes" (page 986).

Discussion
Note that calling the `GetFlavorDataSize` function on a flavor that requires translation will force that translation to be performed in order to determine the data size. Since translation may require a significant amount of time and memory during processing, call the `GetFlavorDataSize` function only when necessary.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Drag.h`

GetFlavorFlags

Gets the flags for a specific flavor in a drag item. (Deprecated in Mac OS X v10.5.)

```
OSErr GetFlavorFlags (
   DragRef theDrag,
   DragItemRef theItemRef,
   FlavorType theType,
   FlavorFlags *theFlags
);
```

Parameters

theDrag

A drag reference.

theItemRef

The reference number of the drag item containing the flavor.

theType

The flavor type for which to get the attributes.

theFlags

On return, a pointer to the attributes of the specified flavor. If a flavor is marked with the `flavorSenderOnly` flag, it is not returned to any application other than the sender.

Return Value

A result code. See "Drag Manager Result Codes" (page 986).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Drag.h`

GetFlavorType

Gets the type of a specific flavor in a drag item. (Deprecated in Mac OS X v10.5.)

```
OSErr GetFlavorType (
   DragRef theDrag,
   DragItemRef theItemRef,
   UInt16 index,
   FlavorType *theType
);
```

Parameters

theDrag

A drag reference.

theItemRef

The reference number of the drag item containing the flavor.

index

The index of the desired flavor.

theType

On return, a pointer to the type of the specified flavor. If a flavor is marked with the `flavorSenderOnly` flag, it is not returned to any application other than the sender.

Return Value
A result code. See "Drag Manager Result Codes" (page 986). If `index` is 0 or larger than the number of flavors in the item, `GetFlavorType` returns the `badDragFlavorErr` result code.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Drag.h`

GetStandardDropLocation

Gets the standard drop location set by the receiver of a drag. (Deprecated in Mac OS X v10.5.)

```
OSStatus GetStandardDropLocation (
   DragRef theDrag,
   StandardDropLocation *outDropLocation
);
```

Parameters

`theDrag`
The drag reference.

`outDropLocation`
A pointer to a value that, on return, represents the location where the drag was dropped. You can use the `GetStandardDropLcoation` function to easily determine whether a drag landed in the trash; if the drop location is the trash, the value of this parameter is `kDragStandardDropLocationTrash`. Otherwise, the value returned here is `kDragStandardDropLocationUnknown`. See "Standard Drop Locations" (page 982) for more information on these values.

Return Value
A result code. See "Drag Manager Result Codes" (page 986).

Availability
Available in Mac OS X v10.2 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Drag.h`

HideDragHilite

Removes highlighting created with the `ShowDragHilite` function. (Deprecated in Mac OS X v10.5.)

```
OSErr HideDragHilite (
   DragRef theDrag
);
```

Parameters

theDrag

The drag reference that is currently showing a drag highlight.

Return Value

A result code. See "Drag Manager Result Codes" (page 986).

Discussion

The `HideDragHilite` function assumes that the highlighting should be erased from the current port. Your application should make sure that the correct port is set before calling the `HideDragHilite` function. Also, highlighting erased by the `HideDragHilite` function is clipped to the current port. Make sure that the port's clip region is appropriately sized to remove the highlighting.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Drag.h`

InstallReceiveHandler

Installs a receive handler function for one or all of your application's windows. (Deprecated in Mac OS X v10.5.)

```
OSErr InstallReceiveHandler (
   DragReceiveHandlerUPP receiveHandler,
   WindowRef theWindow,
   void *handlerRefCon
);
```

Parameters

receiveHandler

A pointer to a receive handler function. Installing a receive handler function allows your application to accept a drag by getting drag item flavor data from the Drag Manager when the user releases the mouse button while dragging over one of your application's windows.

theWindow

A reference to the window for which to install the receive handler. When a drop occurs over this window, the Drag Manager calls your receive handler function to allow your application to accept the drag. If you pass `NULL`, the receive handler function is installed in the default handler space for your application. Receive handler functions installed in this way are called when a drop occurs over any window that belongs to your application. You may install more than one receive handler function on a single window.

handlerRefCon

A pointer to a reference constant that will be forwarded to your receive handler function when it is called by the Drag Manager. Use this constant to pass any data you wish to forward to your drag receive handler.

Return Value
A result code. See "Drag Manager Result Codes" (page 986).

Discussion
The Drag Manager sequentially calls all of the receive handler functions installed on a window when a drop occurs in that window.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Drag.h`

InstallTrackingHandler

Installs a tracking handler function for one or all of your application's windows. (Deprecated in Mac OS X v10.5.)

```
OSErr InstallTrackingHandler (
   DragTrackingHandlerUPP trackingHandler,
   WindowRef theWindow,
   void *handlerRefCon
);
```

Parameters

trackingHandler
A pointer to a tracking handler function. Installing a tracking handler function allows your application to track the user's movements through your application's windows during a drag.

theWindow
A reference to the window for which to track and handle dragging. When the cursor moves into this window during a drag, the Drag Manager sends tracking messages to the tracking handler function. If you pass `NULL`, the tracking handler function is installed in the default handler space for your application. Tracking handler functions installed in this way are called when the user moves the mouse over any window that belongs to your application. You may install more than one drag tracking handler on a single window.

handlerRefCon
A pointer to a reference constant that will be forwarded to your tracking handler function when it is called by the Drag Manager. Use this constant to pass any data you wish to forward to your tracking handler function.

Return Value
A result code. See "Drag Manager Result Codes" (page 986).

Discussion
The Drag Manager sequentially calls all of the tracking handler functions installed for a window when the user moves the cursor over that window during a drag.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Drag.h`

InvokeDragDrawingUPP

Calls your drag drawing callback. (Deprecated in Mac OS X v10.5.)

```
OSErr InvokeDragDrawingUPP (
   DragRegionMessage message,
   RgnHandle showRegion,
   Point showOrigin,
   RgnHandle hideRegion,
   Point hideOrigin,
   void *dragDrawingRefCon,
   DragRef theDrag,
   DragDrawingUPP userUPP
);
```

Return Value
A result code. See "Drag Manager Result Codes" (page 986).

Discussion
You should not need to use the function `InvokeDragDrawingUPP`, as the system calls your drag drawing callback for you.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`Drag.h`

InvokeDragInputUPP

Calls your drag input callback.

```
OSErr InvokeDragInputUPP (
   Point *mouse,
   SInt16 *modifiers,
   void *dragInputRefCon,
   DragRef theDrag,
   DragInputUPP userUPP
);
```

Return Value
A result code. See "Drag Manager Result Codes" (page 986).

Discussion
You should not need to use the function `InvokeDragInputUPP`, as the system calls your drag input callback for you.

Availability
Available in Mac OS X v10.0 and later.

Declared In
Drag.h

InvokeDragReceiveHandlerUPP

Calls your drag receive handler. (Deprecated in Mac OS X v10.5.)

```
OSErr InvokeDragReceiveHandlerUPP (
   WindowRef theWindow,
   void *handlerRefCon,
   DragRef theDrag,
   DragReceiveHandlerUPP userUPP
);
```

Return Value
A result code. See "Drag Manager Result Codes" (page 986).

Discussion
You should not need to use the function `InvokeDragReceiveHandlerUPP`, as the system calls your drag receive handler for you.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
Drag.h

InvokeDragSendDataUPP

Calls your drag send data callback. (Deprecated in Mac OS X v10.5.)

```
OSErr InvokeDragSendDataUPP (
   FlavorType theType,
   void *dragSendRefCon,
   DragItemRef theItemRef,
   DragRef theDrag,
   DragSendDataUPP userUPP
);
```

Return Value
A result code. See "Drag Manager Result Codes" (page 986).

Discussion
You should not need to use the function `InvokeDragSendDataUPP`, as the system calls your drag send data callback for you.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
Drag.h

InvokeDragTrackingHandlerUPP

Calls your drag tracking handler. (Deprecated in Mac OS X v10.5.)

```
OSErr InvokeDragTrackingHandlerUPP (
   DragTrackingMessage message,
   WindowRef theWindow,
   void *handlerRefCon,
   DragRef theDrag,
   DragTrackingHandlerUPP userUPP
);
```

Return Value
A result code. See "Drag Manager Result Codes" (page 986).

Discussion
You should not need to use the function `InvokeDragTrackingHandlerUPP`, as the system calls your drag tracking handler for you.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`Drag.h`

NewDrag

Creates a new drag reference for your application to use with the Drag Manager.

```
OSErr NewDrag (
   DragRef *theDrag
);
```

Parameters

theDrag

On return, a pointer to the newly created drag reference. This drag reference is required when adding drag item flavors and calling the `TrackDrag` function. Your installed drag handler functions receive this drag reference so they can call other Drag Manager functions.

Return Value
A result code. See "Drag Manager Result Codes" (page 986).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Drag.h`

NewDragDrawingUPP

Creates a new universal procedure pointer (UPP) to a drag drawing callback. (Deprecated in Mac OS X v10.5.)

```
DragDrawingUPP NewDragDrawingUPP (
   DragDrawingProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your drag drawing callback.

Return Value

On return, a UPP to the drag drawing callback. See the description of the `DragDrawingUPP` data type.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`Drag.h`

NewDragInputUPP

Creates a new universal procedure pointer (UPP) to a drag input callback.

```
DragInputUPP NewDragInputUPP (
   DragInputProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your drag input callback.

Return Value

On return, a UPP to the drag input callback. See the description of the `DragInputUPP` data type.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Drag.h`

NewDragReceiveHandlerUPP

Creates a new universal procedure pointer (UPP) to a drag receive handler. (Deprecated in Mac OS X v10.5.)

```
DragReceiveHandlerUPP NewDragReceiveHandlerUPP (
   DragReceiveHandlerProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your drag receive handler.

Return Value

On return, a UPP to the drag receive handler. See the description of the `DragReceiveHandlerUPP` data type.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`Drag.h`

NewDragSendDataUPP

Creates a new universal procedure pointer (UPP) to a drag send data callback. (Deprecated in Mac OS X v10.5.)

```
DragSendDataUPP NewDragSendDataUPP (
   DragSendDataProcPtr userRoutine
);
```

Parameters

userRoutine
A pointer to your drag send data callback.

Return Value
On return, a UPP to the drag send data callback. See the description of the `DragSendDataUPP` data type.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`Drag.h`

NewDragTrackingHandlerUPP

Creates a new universal procedure pointer (UPP) to a drag tracking handler. (Deprecated in Mac OS X v10.5.)

```
DragTrackingHandlerUPP NewDragTrackingHandlerUPP (
   DragTrackingHandlerProcPtr userRoutine
);
```

Parameters

userRoutine
A pointer to your drag tracking handler.

Return Value
On return, a UPP to the drag tracking handler. See the description of the `DragTrackingHandlerUPP` data type.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`Drag.h`

RemoveReceiveHandler

Removes a receive handler function from one or all of your application's windows. (Deprecated in Mac OS X v10.5.)

```
OSErr RemoveReceiveHandler (
   DragReceiveHandlerUPP receiveHandler,
   WindowRef theWindow
);
```

Parameters

receiveHandler

A pointer to a receive handler function.

theWindow

A reference to the window from which to remove the receive handler function. Pass `NULL` to remove the specified receive handler function from the default handler space for your application.

Return Value

A result code. See "Drag Manager Result Codes" (page 986).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Drag.h`

RemoveTrackingHandler

Removes a tracking handler function from one or all of your application's windows. (Deprecated in Mac OS X v10.5.)

```
OSErr RemoveTrackingHandler (
   DragTrackingHandlerUPP trackingHandler,
   WindowRef theWindow
);
```

Parameters

trackingHandler

A pointer to the tracking handler function to be removed.

theWindow

A reference to the window from which to remove the drag tracking handler function. Pass `NULL` to remove the specified tracking handler function from the default handler space for your application.

Return Value

A result code. See "Drag Manager Result Codes" (page 986).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`Drag.h`

SetDragAllowableActions

Sets the actions that the drag receiver may take on the data within a drag.

```
OSStatus SetDragAllowableActions (
   DragRef theDrag,
   DragActions inActions,
   Boolean isLocal
);
```

Parameters

theDrag
The drag reference.

inActions
A field specifying the allowable actions for the drag. See "Drag Actions" (page 979) for a description of the values you may use here.

isLocal
A Boolean value allowing the drag sender to specify whether the actions passed in the `inActions` parameter are allowable for a local receiver or for a remote receiver. Pass `true` in this parameter if the drag actions are for local receivers.

Return Value
A result code. See "Drag Manager Result Codes" (page 986).

Discussion
The actions set by the drag sender using the `SetDragAllowableActions` function are not requirements; they are highly recommended suggestions for operations the drag receiver may perform. The caller may select whether these drag actions apply to a remote or local process with the `inLocal` parameter.

Availability
Available in Mac OS X v10.1 and later.
Not available to 64-bit applications.

Declared In
`Drag.h`

SetDragDrawingProc

Sets the drag drawing function for the Drag Manager to use with a particular drag. (Deprecated in Mac OS X v10.5.)

Not recommended

```
OSErr SetDragDrawingProc (
   DragRef theDrag,
   DragDrawingUPP drawingProc,
   void *dragDrawingRefCon
);
```

Parameters

theDrag

The drag reference for which the drag drawing function will be set.

drawingProc

The drag drawing function to be called by the Drag Manager to draw, move, and hide the "dotted outline" drag feedback on the screen during a drag.Your drag drawing function can implement any type of drag feedback, such as dragging a bitmap of the object being dragged. Details for how to write a drag drawing function are covered in the "Drag Manager Callbacks" (page 963) section.

dragDrawingRefCon

A pointer to a reference constant that will be forwarded to your drag drawing function when it is called by the Drag Manager. Use this constant to pass any data you wish to forward to your drag drawing function.

Return Value

A result code. See "Drag Manager Result Codes" (page 986).

Carbon Porting Notes

Drag drawing functions are not supported in Mac OS X, although they continue to work in CarbonLib when running Mac OS 8 and Mac OS 9.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Drag.h`

SetDragDropAction

Sets the action performed by the receiver of the drag.

```
OSStatus SetDragDropAction (
   DragRef theDrag,
   DragActions inAction
);
```

Parameters

theDrag

The drag reference for which to set the drop action.

inAction

The drop action performed. More than one action may be performed. See "Drag Actions" (page 979) for a description of the values you may use here.

Return Value

A result code. See "Drag Manager Result Codes" (page 986).

Availability
Available in Mac OS X v10.1 and later.
Not available to 64-bit applications.

Declared In
`Drag.h`

SetDragImage

Associates an image with a drag reference. (Deprecated in Mac OS X v10.4. Use `SetDragImageWithCGImage` (page 951) instead.)

```
OSErr SetDragImage (
   DragRef inDrag,
   PixMapHandle inImagePixMap,
   RgnHandle inImageRgn,
   Point inImageOffsetPt,
   DragImageFlags inImageFlags
);
```

Parameters

theDrag

The drag reference.

imagePixMap

A handle to a `PixMap` describing the image. The Drag Manager temporarily locks the `PixMapHandle` during the drag. The Drag Manager does not copy the information in this parameter; you must ensure that the data to which this parameter refers continues to exist until `TrackDrag` completes.

imageRgn

A mask describing the portion of the `PixMap` contained in the `imagePixMap` parameter which contains the drag image. Pass `NULL` for `imageRgn` if the entire `PixMap`, including white space, should be dragged.

The Drag Manager does not copy the `imageRgn` parameter data. Until `TrackDrag` completes or `SetDragImage` is called again to update the image, you must ensure that the data to which this parameter refers continues to exist.

Don't confuse the region passed to the function `TrackDrag` and that passed to the `SetDragImage` function. The former is what's drawn to the screen during dragging, while the latter is used only for drawing the correct portion of the drag image.

imageOffsetPt

The offset required to move the `PixMap` specified in the `imagePixMap` parameter to the global coordinates where the image initially appears. If this parameter is (0,0), the `PixMap` should already be in global coordinates.

theImageFlags

Flags controlling the appearance of the drag image. See "Drag Image Flags" (page 982) for a description of the values you can use in this parameter.

Return Value
A result code. See "Drag Manager Result Codes" (page 986).

Discussion
The sending application should call `SetDragImage` prior to calling `TrackDrag`. Prior to calling `SetDragImage`, the application should draw a solid, opaque image into the `PixMap` specified in the `imagePixMap` parameter. The Drag Manager will provide translucency effects. Your application can obtain a `PixMap` by calling the QuickDraw function `GetGWorldPixMap` and supplying a `GWorld` into which your application has drawn the image.

To allow the Drag Manager to analyze the `PixMap`'s colors in order to determine if it can be rendered on the available screens, Apple recommends using an 8-bit `GWorld` for the `PixMap`.

Special Considerations

`SetDragImage` installs a custom drawing procedure to do the translucent drawing. Applications calling `SetDragImage` should not also call `SetDragDrawingProc` for the same drag.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`Drag.h`

SetDragImageWithCGImage

Associates a Core Graphics image with a drag reference.

```
OSStatus SetDragImageWithCGImage (
   DragRef inDrag,
   CGImageRef inCGImage,
   const HIPoint *inImageOffsetPt,
   DragImageFlags inImageFlags
);
```

Parameters

inDrag
The drag reference for which to display the image.

inCGImage
A reference to the image to display during the drag. The Drag Manager retains this image for the duration of the drag, so you may release the image immediately after calling `SetDragImageWithCGImage`.

inImageOffsetPt
A pointer to the offset from the mouse location to the upper left corner of the image, normally expressed in negative values. For example, an offset of (-30, -30) centers a 60 by 60 pixel image on the mouse. Note that this differs from the usage of the offset passed to the `SetDragImage` function.

inImageFlags
Flags controlling the appearance of the drag image. See "Drag Image Flags" (page 982) for a description of the values you can use in this parameter.

Return Value
A result code. See "Drag Manager Result Codes" (page 986).

Discussion
This function is called by the sender of a drag to set the image displayed to provide user feedback during the drag. You can call the `SetDragImageWithCGImage` function at any point during the drag to update the image.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`Drag.h`

SetDragInputProc

Sets the drag input function for the Drag Manager to use with a particular drag.

```
OSErr SetDragInputProc (
   DragRef theDrag,
   DragInputUPP inputProc,
   void *dragInputRefCon
);
```

Parameters

theDrag
The drag reference for which the drag input function will be set.

inputProc
The drag input function to be called by the Drag Manager whenever the Drag Manager requires the location of the mouse, the state of the mouse button, and the status of the modifier keys on the keyboard. The Drag Manager typically calls this function once per cycle through the Drag Manager's main drag tracking loop.Your drag input function may either modify the current state of the mouse and keyboard to slightly alter dragging behavior or entirely replace the input data to drive the drag completely by itself. Details for how to write a drag input function are covered in the "Drag Manager Callbacks" (page 963) section.

dragInputRefCon
A pointer to a reference constant that will be forwarded to your drag input function when it is called by the Drag Manager. Use this constant to pass any data you wish to forward to your drag input function.

Return Value
A result code. See "Drag Manager Result Codes" (page 986).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Drag.h`

SetDragItemBounds

Sets the bounding rectangle of a drag item.

```
OSErr SetDragItemBounds (
   DragRef theDrag,
   DragItemRef theItemRef,
   const Rect *itemBounds
);
```

Parameters

theDrag

A drag reference.

theItemRef

The reference number of the drag item whose bounds you wish to set.

itemBounds

A pointer to the bounding rectangle to set for the specified drag item. This rectangle is specified in global coordinates relative to the mouse down position.

Return Value

A result code. See "Drag Manager Result Codes" (page 986).

Discussion

Your application would normally call the `SetDragItemBounds` function on each drag item before starting a drag with the `TrackDrag` function.

If you do not set the bounds of an item, the rectangle returned by the `GetDragItemBounds` function is an empty rectangle centered under the pinned mouse location.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Drag.h`

SetDragItemFlavorData

Sets the data or part of the data contained within an existing flavor. (Deprecated in Mac OS X v10.5.)

```
OSErr SetDragItemFlavorData (
   DragRef theDrag,
   DragItemRef theItemRef,
   FlavorType theType,
   const void *dataPtr,
   Size dataSize,
   UInt32 dataOffset
);
```

Parameters

theDrag

The drag reference whose flavor data will be set.

theItemRef

The drag item reference of the item that contains the flavor you wish to set all or part of the data for. The data pointed to by the *dataPtr* parameter with the size specified in the *dataSize* parameter is placed into the flavor structure at the offset specified by the *dataOffset* parameter.

theType
The data type of the existing flavor for which all or part of the data will be set.

dataPtr
A pointer to the flavor data.

dataSize
The size, in bytes, of the flavor data.

dataOffset
The offset, in bytes, into the flavor structure at which to place the data specified by the *dataPtr* and the *dataSize* parameters.

Return Value
A result code. See "Drag Manager Result Codes" (page 986).

Discussion
This function is commonly used in situations where a flavor's data is not added to the flavor when the flavor is created using the `AddDragItemFlavor` function. When the sender's drag send data function is called, the `SetDragItemFlavorData` function can be used to provide the requested data to the Drag Manager. This method is useful when the data needs to be translated by the sender and it would be expensive to compute the data before it is required.

Unlike the functions that add flavors, this function may be called both before and during a drag.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Drag.h`

SetDragMouse

Sets the current pinned mouse location.

```
OSErr SetDragMouse (
   DragRef theDrag,
   Point globalPinnedMouse
);
```

Parameters

theDrag
A drag reference.

globalPinnedMouse
The coordinates to which to set the pinned mouse location, in global screen coordinates. The pinned mouse location is used to draw the drag region on the screen.

Return Value
A result code. See "Drag Manager Result Codes" (page 986).

Discussion
To constrain the mouse within one of your application's windows, call the `SetDragMouse` function from within your tracking handler when you receive the `kDragTrackingInWindow` messages. The Drag Manager updates the position of the drag region on the screen after each time your tracking handlers are called.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Drag.h`

SetDragSendProc

Sets the send data function for the Drag Manager to use with a particular drag. (Deprecated in Mac OS X v10.5.)

```
OSErr SetDragSendProc (
   DragRef theDrag,
   DragSendDataUPP sendProc,
   void *dragSendRefCon
);
```

Parameters

theDrag
The drag reference to set the send data function for.

sendProc
The send data function that will be called by the Drag Manager when the receiver of a drop requests the flavor data of a flavor that has not been cached by the Drag Manager.

dragSendRefCon
A pointer to a reference constant that will be forwarded to your send data function when it is called by the Drag Manager. Use this constant to pass any data you wish to forward to your send data function.

Return Value
A result code. See "Drag Manager Result Codes" (page 986).

Discussion
The Drag Manager caches drag item flavor data when the flavor is added to a drag by calling the `AddDragItemFlavor` function. If `NULL` is passed to the `AddDragItemFlavor` function as the data pointer, the flavor data is not cached and the Drag Manager will call your send data function when the drag item flavor data is requested.

You do not need to provide a send data function if your application never passes `NULL` to the `AddDragItemFlavor` function when adding a drag item flavor to a drag.

Details for how to write a send data function are covered in the "Drag Manager Callbacks" (page 963) section.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Drag.h`

SetDropLocation

Sets the Apple Event descriptor for the drop location of a drag. (Deprecated in Mac OS X v10.5.)

```
OSErr SetDropLocation (
   DragRef theDrag,
   const AEDesc *dropLocation
);
```

Parameters

theDrag

A drag reference.

dropLocation

A pointer to the Apple Event descriptor of the drop location to set.

Return Value

A result code. See "Drag Manager Result Codes" (page 986).

Discussion

This function is typically called by a receive handler before attempting to get any flavor data using the `GetFlavorDataSize` or `GetFlavorData` functions. When a sender application's drag send data function is called to provide flavor data to a receiver, the `GetDropLocation` function can be called to determine the drop location while providing data.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Drag.h`

SetStandardDropLocation

Used by the receiver of a drag to set the standard drop location for a drag. (Deprecated in Mac OS X v10.5.)

```
OSStatus SetStandardDropLocation (
   DragRef theDrag,
   StandardDropLocation dropLocation
);
```

Parameters

theDrag

The drag reference.

dropLocation

A value representing the location where the drag was dropped. See "Standard Drop Locations" (page 982) for a description of the values you may use here.

Return Value

A result code. See "Drag Manager Result Codes" (page 986).

Availability

Available in Mac OS X v10.2 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`Drag.h`

ShowDragHilite

Highlights an area of your window during a drag. (Deprecated in Mac OS X v10.5.)

```
OSErr ShowDragHilite (
   DragRef theDrag,
   RgnHandle hiliteFrame,
   Boolean inside
);
```

Parameters

theDrag

The drag reference of the drag currently in progress.

hiliteFrame

A QuickDraw region of the frame of the window, pane, or shape you wish to highlight, in the window's local coordinate system.

inside

Pass `true` to draw the highlighting inside the frame shape. Otherwise it will be drawn outside the frame shape. Note that in either case, the highlight will not include the boundary edge of the frame. This allows you to highlight inside a window frame or a pane, or to highlight outside of a container or object in your window.

Return Value
A result code. See "Drag Manager Result Codes" (page 986).

Discussion
The `ShowDragHilite` function creates a standard drag and drop highlight in your window. Your tracking handler function should call this if a drop is allowed at the current mouse position.

You can only have one highlight showing at a time, and if you call this function when a highlight is currently visible, the first one is removed before the newly requested highlight is shown.

The `ShowDragHilite` function uses a two pixel thick line when drawing the highlight.

The `ShowDragHilite` function assumes that the highlighting should be drawn in the current port. Your application should make sure that the correct port is set before calling the `ShowDragHilite` function. Also, highlighting drawn by the `ShowDragHilite` function is clipped to the current port. Make sure that the port's clip region is appropriately sized to draw the highlighting.

The Drag Manager maintains the currently highlighted portion of your window if you use the `HideDragHilite` and `UpdateDragHilite` functions. If you intend to scroll the window that contains the highlighting, you can use the `DragPreScroll` and `DragPostScroll` functions to properly update the drag highlighting.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Drag.h`

TrackDrag

Drags an item or collection of items from your application.

```
OSErr TrackDrag (
   DragRef theDrag,
   const EventRecord *theEvent,
   RgnHandle theRegion
);
```

Parameters

theDrag
A drag reference for performing the drag operation.

theEvent
A pointer to the `mouseDown` event record that your application received that resulted in starting a drag.

theRegion
A region that represents the item or items being dragged. Note that under normal circumstances, the drag region should only include the pixels that represent the outline of the items being dragged. The Drag Manager draws the region on the screen by calling the `PaintRgn` function (not the `FrameRgn` function).

Return Value
A result code. See "Drag Manager Result Codes" (page 986). Under some circumstances, `TrackDrag` may fail with a `procNotFound` error. See "Special Considerations" below for a description of the events that may cause this problem.

Discussion
The Drag Manager follows the cursor on the screen with the "dotted outline" drag feedback and sends tracking messages to applications that have registered drag tracking handlers. The drag item flavor information cached for the drag is available to each application that becomes active during a drag.

When the user releases the mouse button, the Drag Manager calls any drag receive handlers that have been registered on the destination window. An application's drag receive handler(s) are responsible for accepting the drag and transferring the dragged data into their application.

The `TrackDrag` function returns `noErr` in situations where the user selected a destination for the drag and the destination received data from the Drag Manager. If the user drops over a non-aware application or the receiver does not accept any data from the Drag Manager, the Drag Manager automatically provides a "zoom back" animation and returns the `userCanceledErr` flag.

Special Considerations

During the call to the `TrackDrag` function, your application's context is temporarily switched out when the Drag Manager calls a different application's tracking and receive handlers. Do not depend on your application's context to be active for the entire duration of a drag.

The following actions may cause `TrackDrag` to fail with a `procNotFound` error:

- Using a high-level debugger with the Drag Manager. Although there is no workaround for this problem, your code should work fine when run without the debugger.

- Passing `TrackDrag` an event record in which the `where` field is expressed in local coordinates. In such cases, the `where` field often points outside of the window in which the drag originated. This problem can cause a crash as well as a `procNotFound` error.
- Using the Drag Manager with Text Services Manager windows when the `gestaltDragMgrFloatingWind` bit isn't defined.

For more information, see the Q&A at:

http://developer.apple.com/dev/techsupport/develop/issue29/macqa.html.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Drag.h`

UpdateDragHilite

Updates the portion of the drag highlight that was drawn over by your application. (Deprecated in Mac OS X v10.5.)

```
OSErr UpdateDragHilite (
   DragRef theDrag,
   RgnHandle updateRgn
);
```

Parameters

`theDrag`
The drag reference.

`updateRgn`
The region that needs to be updated, typically the port's `updateRgn`.

Return Value
A result code. See "Drag Manager Result Codes" (page 986).

Discussion
Use this function if your application draws into the highlighted portion of your window during a drag. For example, dragging over a folder icon in the Finder causes the Finder to redraw the folder icon in its darkened (selected) color. The Finder calls the `UpdateDragHilite` function to redraw any portion of the drag highlight that may have intersected with the folder icon.

You must guarantee, however, that any current highlighting within the `updateRgn` has been completely erased or is clipped out. If this function is asked to highlight over an area which is still highlighted, it will be redrawn incorrectly.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Drag.h`

WaitMouseMoved

Returns `true` if a mouse movement is the beginning of a drag.

```
Boolean WaitMouseMoved (
   Point initialGlobalMouse
);
```

Parameters

`initialMouse`

The point where a `mouseDown` event occurred in global screen coordinates.

Return Value

`True` if the mouse moves away from the `initialMouse` location before the mouse button is released, otherwise `false`.

Discussion

You can use this function to determine whether you should begin to drag the object when your application receives a `mouseDown` event on a draggable object.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Drag.h`

ZoomRects

Animates a rectangle into a second rectangle. (Deprecated in Mac OS X v10.5.)

```
OSErr ZoomRects (
   const Rect *fromRect,
   const Rect *toRect,
   SInt16 zoomSteps,
   ZoomAcceleration acceleration
);
```

Parameters

`fromRect`

A pointer to the starting rectangle to animate from, in global coordinates.

`toRect`

A pointer to the ending rectangle to animate to, in global coordinates.

`zoomSteps`

Specifies the number of animation steps to be shown between the source and destination rectangles. The minimum number of steps is 4. If less than 4 steps are specified, 4 will be used. The maximum number of steps is 25. If more than 25 steps are specified, 25 will be used.

acceleration

Specifies how the intermediate animation steps will be calculated. Using the `kZoomNoAcceleration` constant makes the distance between steps from the source to the destination equal. Using the `kZoomAccelerate` constant makes each step from the source to the destination increasingly larger, making the animation appear to speed up as it approaches the destination. Using the `kZoomDecelerate` constant makes each step from the source to the destination smaller, making the animation appear to slow down as it approaches the destination.

Return Value
A result code. See "Drag Manager Result Codes" (page 986).

Discussion
The `ZoomRects` function animates a movement between two rectangles on the screen. It does this by drawing gray dithered rectangles incrementally toward the destination rectangle.

The `ZoomRects` function draws on the entire screen, outside of the current port. It does not change any pixels on the screen except during the animation. It also preserves the current port and the port's settings.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Drag.h`

ZoomRegion

Animates a region's outline from one screen location to another. (Deprecated in Mac OS X v10.5.)

```
OSErr ZoomRegion (
   RgnHandle region,
   Point zoomDistance,
   SInt16 zoomSteps,
   ZoomAcceleration acceleration
);
```

Parameters

region

The region to animate.

zoomDistance

The horizontal and vertical distance from the starting point that the region will animate to.

zoomSteps

Specifies the number of animation steps to be shown between the source and destination rectangles. The minimum number of steps is 4. If less than 4 steps are specified, 4 will be used. The maximum number of steps is 25. If more than 25 steps are specified, 25 will be used.

acceleration

Specifies how the intermediate animation steps will be calculated. Using the `kZoomNoAcceleration` constant makes the distance between steps from the source to the destination equal. Using the `kZoomAccelerate` constant makes each step from the source to the destination increasingly larger, making the animation appear to speed up as it approaches the destination. Using the `kZoomDecelerate` constant makes each step from the source to the destination smaller, making the animation appear to slow down as it approaches the destination.

Return Value
A result code. See "Drag Manager Result Codes" (page 986).

Discussion
The `ZoomRegion` function animates a region from one location to another on the screen. It does this by drawing gray dithered regions incrementally toward the destination region.

The `ZoomRegion` function draws on the entire screen, outside of the current port. It does not change any pixels on the screen except during its animation. It also preserves the current port and the port's settings.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Drag.h`

Callbacks by Task

Tracking and Receiving Drags

`DragTrackingHandlerProcPtr` (page 968)
Defines a pointer to a drag tracking handler.

`DragReceiveHandlerProcPtr` (page 966)
Defines a pointer to a drag receive handler.

Overriding Drag Manager Behavior

`DragSendDataProcPtr` (page 967)
Defines a pointer to a drag send data function, called by the Drag Manager to supply flavor data to the drag receiver.

`DragInputProcPtr` (page 964)
Defines a pointer to a drag input function that modifies keyboard and mouse input to the Drag Manager.

`DragDrawingProcPtr` (page 963)
Defines a pointer to a drag drawing function that draws the drag region.

Callbacks

DragDrawingProcPtr

Defines a pointer to a drag drawing function that draws the drag region.

Not recommended

```
typedef OSErr (*DragDrawingProcPtr) (
                DragRegionMessage message,
                RgnHandle showRegion,
                Point showOrigin,
                RgnHandle hideRegion,
                Point hideOrigin,
                void * dragDrawingRefCon,
                DragRef theDrag);
```

If you name your function `MyDragDrawingFunction`, you would declare it like this:

```
OSErr MyDragDrawingFunction (
                DragRegionMessage message,
                RgnHandle showRegion,
                Point showOrigin,
                RgnHandle hideRegion,
                Point hideOrigin,
                void * dragDrawingRefCon,
                DragRef theDrag);
```

Parameters

message

A drag region drawing message from the Drag Manager. Use this message to determine what action your drag drawing callback function should take. These messages are described further in "Drag Drawing Messages" (page 975).

showRegion

A region containing the drag region as it should be drawn or is currently visible on the screen, in global screen coordinates. The `showRegion` parameter has slightly different meanings depending on the message passed to your drag drawing callback.

showOrigin

The point corresponding to the original `mouseDown` location in the drag region within the given `showRegion`, in global screen coordinates.

hideRegion

A region containing the drag region as it should be erased from the screen, in global screen coordinates. The `hideRegion` parameter has slightly different meanings depending on the message passed to your drag drawing callback.

hideOrigin

The point corresponding to the original `mouseDown` location in the drag region within the given `hideRegion`, in global screen coordinates.

dragDrawingRefCon

A pointer to the reference constant that was provided when the `SetDragDrawingProc` function was called to install this function.

theDrag
The drag reference of the drag.

Return Value
A result code. See "Drag Manager Result Codes" (page 986).

Discussion
If your application set a custom drawing function for a drag using the `SetDragDrawingProc` function, the Drag Manager calls this drawing function to perform all drag region drawing operations.

The Drag Manager tracks the drag region as it appears on the screen and as it should follow the mouse. All drag region operations are performed on the region specified to the `TrackDrag` function. Drag region drawing is managed by sending your drag drawing callback function messages to show and hide pieces of the drag region.

The Drag Manager has its own drag region port that is used to do all drag region drawing during a drag. This port is set to the current port before calling your drag drawing function. The drag region port is for your drag drawing function's exclusive use during a drag. You may modify its fields and depend on its contents between calls to your drag drawing callback function.

Special Considerations

For Classic applications, your application's context is not available when your drag drawing callback function is called by the Drag Manager. If you need access to your application's global variables, you will need to setup and restore your application's A5 world yourself.

You cannot call the `WaitNextEvent` function or any other Event Manager functions in your drag drawing callback function. This restriction includes calling any functions that may call the Event Manager, such as the `ModalDialog` or `Alert` functions.

Carbon Porting Notes

Drag drawing functions are not supported in Mac OS X, although they continue to work in CarbonLib when running Mac OS 8 and Mac OS 9.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Drag.h`

DragInputProcPtr

Defines a pointer to a drag input function that modifies keyboard and mouse input to the Drag Manager.

```
typedef OSErr (*DragInputProcPtr) (

                Point * mouse,
                SInt16 * modifiers,
                void * dragInputRefCon,
                DragRef theDrag);
```

If you name your function `MyDragInputFunction`, you would declare it like this:

```
OSErr MyDragInputFunction (
                Point * mouse,
                SInt16 * modifiers,
```

```
                void * dragInputRefCon,
                DragRef theDrag);
```

Parameters

mouse

On entry, a pointer to the location. On return, your drag input function should provide the desired current mouse location. The mouse location is specified in global coordinates.

modifiers

On entry, a pointer to a value indicating the current state of the keyboard modifiers and mouse button. On return, your drag input function should provide a pointer to the desired state of the keyboard modifiers and mouse button. The modifiers are specified using the same format and constants as the Event Manager's `EventRecord.modifiers` field.

dragInputRefCon

A pointer to the reference constant that was provided when the `SetDragInputProc` function was called to install this function.

theDrag

The drag reference of the drag.

Return Value

A result code. See "Drag Manager Result Codes" (page 986).

Discussion

Each time the Drag Manager samples the mouse and keyboard, it calls your drag input callback (if one has been set by calling the `SetDragInputProc` function) to provide a way for your application to modify or completely change the mouse and keyboard input to the Drag Manager.

When your drag input callback function is called, the mouse and modifiers parameters contain the actual values from the physical input devices. Your drag input callback function may modify these values in any way. For example, your drag input callback function may simply inhibit the control key modifier bit from being set or it may completely replace the mouse coordinates with those generated some other way to drive the drag itself.

Note that the Drag Manager uses the `buttonState` flag in the `modifiers` parameter to determine when the mouse button has been released to finish a drag.

Special Considerations

For Classic applications, your application's context is not available when your drag input callback function is called by the Drag Manager. If you need access to your application's global variables, you will need to setup and restore your application's A5 world yourself.

You cannot call the `WaitNextEvent` function or any other Event Manager functions from your drag input callback function. This restriction includes calling any functions that may call the Event Manager, such as the `ModalDialog` or `Alert` functions.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Drag.h`

DragReceiveHandlerProcPtr

Defines a pointer to a drag receive handler.

```
typedef OSErr (*DragReceiveHandlerProcPtr)
(
                WindowRef theWindow,
                void * handlerRefCon,
                DragRef theDrag);
```

If you name your function `MyDragReceiveHandler`, you would declare it like this:

```
OSErr MyDragReceiveHandler (
                WindowRef theWindow,
                void * handlerRefCon,
                DragRef theDrag);
```

Parameters

theWindow

A reference to the window that the drop occurred in.

handlerRefCon

A pointer to the reference constant that was provided to the `InstallReceiveHandler` function when this handler was installed.

theDrag

The drag reference of the drag.

Return Value

A result code. See "Drag Manager Result Codes" (page 986).

Discussion

When the user releases a drag in a window, the Drag Manager calls any drag receive handler functions that have been installed on that window. You can get the information about the data that is being dragged, to determine if your window will accept the drop, by using the drag information functions provided by the Drag Manager. After your drag receive handler decides that it can accept the drop, use the `GetFlavorData` function to get the data from the sender of the drag.

When the Drag Manager calls your drag receive handler, the port is set to the window that the drop occurred in. If you want to provide an optional Apple Event descriptor of the drop location for the sender, use the `SetDropLocation` function to set the drop location descriptor before calling the sender with the `GetFlavorData` or `GetFlavorDataSize` functions.

If you return any result code other than `noErr` from your drag receive handler, the Drag Manager will "zoomback" the drag region to the source location and return the `userCanceledErr` result code from the `TrackDrag` function. If the drag is dropped into a location that cannot accept the drag (such as the window title bar or window scroll bars) or no acceptable data types were available, your drag receive handler should return the `dragNotAcceptedErr` result code, which will cause the Drag Manager to provide the "zoomback" animation described above.

Special Considerations

For Classic applications, the Drag Manager guarantees that your application's A5 world and low-memory environment is properly set up for your application's use. Therefore, you can allocate memory, and use your application's global variables. You can also rely on low-memory globals being valid.

Although it is possible to call `WaitNextEvent` or other functions that run the event loop from within your drag receive handler, it is not recommended as it can cause the drag to timeout and may result in a crash or in corrupt data.

Note that the Process Manager's process switching mechanism is disabled during calls to your handler. If your application is not frontmost when calling these functions, your application will not be able to switch forward. This could result in a situation where a modal dialog appears behind the front process but will not be able to come forward in order to interact with the user.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Drag.h`

DragSendDataProcPtr

Defines a pointer to a drag send data function, called by the Drag Manager to supply flavor data to the drag receiver.

```
typedef OSErr (*DragSendDataProcPtr)
(
                FlavorType theType,
                void * dragSendRefCon,
                DragItemRef theItemRef,
                DragRef theDrag);
```

If you name your function `MyDragSendDataFunction`, you would declare it like this:

```
OSErr MyDragSendDataFunction (
                FlavorType theType,
                void * dragSendRefCon,
                DragItemRef theItemRef,
                DragRef theDrag);
```

Parameters

`theType`
The flavor type being requested by a drop receiver.

`dragSendRefCon`
A pointer to the reference constant that was provided when the `SetDragSendProc` function was called to install this function.

`theItemRef`
The item reference of the item from which the flavor data is being requested.

`theDrag`
The drag reference of the drag.

Return Value
A result code. See "Drag Manager Result Codes" (page 986).

Discussion
The Drag Manager calls your drag send data function when drag item flavor data is requested by a drop receiver if the drag item flavor data is not already cached by the Drag Manager. Use the function `SetDragItemFlavorData` (page 953) to provide the requested data to the Drag Manager.

The Drag Manager caches all drag item flavor data that was specified in the data pointer when the flavor was added using the `AddDragItemFlavor` function. If the data pointer was `NULL` when the flavor was added, the Drag Manager calls the drag send data function to get the data when a receiver requests the data using the `GetFlavorData` or `GetFlavorDataSize` functions.

A second scenario where the drag send data function is called is when a drop receiver requests a flavor that is translated by the Translation Manager and the source data (which would be a different type than actually requested by the receiver) is not already cached by the Drag Manager.

You can use the `GetDropLocation` function to get the Apple event descriptor of the drop location from within your drag send data function. It is optional for the receiver to provide a drop location descriptor. If the receiver does not provide the drop location descriptor, the `typeNull` value will be returned by the `GetDropLocation` function.You do not need to provide a drag send data function if you never pass `NULL` as the data pointer when calling the `AddDragItemFlavor` function.

Special Considerations

For Classic applications, the Drag Manager guarantees that your application's A5 world and low-memory environment is properly set up for your application's use. Therefore, you can allocate memory, and use your application's global variables. You can also rely on low-memory globals being valid.

Although it is possible to call `WaitNextEvent` or other functions that run the event loop from within your drag send data callback, it is not recommended as it can cause the drag to timeout and may result in a crash or in corrupt data.

Note that the Process Manager's process switching mechanism is disabled during calls to your handler. If your application is not frontmost when calling these functions, your application will not be able to switch forward. This could result in a situation where a modal dialog appears behind the front process but will not be able to come forward in order to interact with the user.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Drag.h`

DragTrackingHandlerProcPtr

Defines a pointer to a drag tracking handler.

```
typedef OSErr (*DragTrackingHandlerProcPtr)
(
                DragTrackingMessage message,
                WindowRef theWindow,
                void * handlerRefCon,
                DragRef theDrag);
```

If you name your function `MyDragTrackingHandler`, you would declare it like this:

```
OSErr MyDragTrackingHandler (
        DragTrackingMessage message,
        WindowRef theWindow,
        void * handlerRefCon,
        DragRef theDrag);
```

Parameters

message

A tracking message from the Drag Manager indicating the action your tracking handler should take. These messages are described further in "Drag Tracking Messages" (page 976).

theWindow

A reference to the window that the mouse is currently over.

handlerRefCon

A pointer to the reference constant that was provided to the `InstallTrackingHandler` function when this handler was installed.

theDrag

The drag reference of the drag.

Return Value
A result code. See "Drag Manager Result Codes" (page 986).

Discussion
When the user drags an item or collection of items through a window, the Drag Manager calls any tracking handlers that have been installed on that window. Your tracking handler can determine the contents of the drag by calling the drag item information functions, such as `CountDragItems` (page 925), `CountDragItemFlavors` (page 925), `GetFlavorType` (page 938) and `GetFlavorFlags` (page 937), and highlighting or modifying the objects in your window accordingly.

When the Drag Manager calls your tracking handler, the port will always be set to the window that the mouse is over.

Special Considerations

For Classic applications, the Drag Manager guarantees that your application's A5 world and low-memory environment is properly set up for your application's use. Therefore, you can allocate memory, and use your application's global variables. You can also rely on low-memory globals being valid.

Although it is possible to call `WaitNextEvent` or other functions that run the event loop from within your drag tracking handler, it is not recommended as it can cause the drag to timeout and may result in a crash or in corrupt data.

Note that the Process Manager's process switching mechanism is disabled during calls to your handler. If your application is not frontmost when calling these functions, your application will not be able to switch forward. This could result in a situation where a modal dialog appears behind the front process but will not be able to come forward in order to interact with the user.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Drag.h`

Data Types

DragRef

Defines a reference to a drag object.

```
typedef struct OpaqueDragRef * DragRef;
```

Discussion
Before calling any other Drag Manager function, you must first create a new drag reference by calling the `NewDrag` function. The drag reference that is returned by the `NewDrag` function is used in all subsequent calls to the Drag Manager. Use the `DisposeDrag` function to dispose of a drag reference after you are finished using it.

The meaning of the bits in a drag reference is internal to the Drag Manager. You should not attempt to interpret the value of the drag reference.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Drag.h`

DragItemRef

Defines a reference to a drag item.

```
typedef UInt32 DragItemRef;
```

Discussion
The drag item reference is a reference number used to refer to a single item in a drag. Drag item reference numbers are created by the sender application when adding drag item flavor information to a drag. Drag item reference numbers are created by and should only be interpreted by the sender application.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Drag.h`

FlavorType

Defines a flavor type.

```
typedef OSType FlavorType;
```

Discussion
The flavor type is a four character type that describes the format of drag item flavor data. The flavor type has the same function as a scrap type; it designates the format of the associated data. Any scrap type or resource type may be used.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Drag.h`

HFSFlavor

Defines a flavor for dragging file system objects.

```
struct HFSFlavor {
    OSType fileType;
    OSType fileCreator;
    UInt16 fdFlags;
    FSSpec fileSpec;
};
typedef struct HFSFlavor HFSFlavor;
```

Fields

`fileType`
The file type of the object.

`fileCreator`
The file creator of the object.

`fdFlags`
The Finder flags of the object.

`fileSpec`
The `FSSpec` structure for the object.

Discussion

The Drag Manager defines a special flavor for dragging file system objects. The HFS drag item flavor is used when dragging document and folder icons in the Finder. The HFS drag item flavor data structure is defined by the `HFSFlavor` data type.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Drag.h`

PromiseHFSFlavor

Defines a data flavor for promising file system objects.

```
struct PromiseHFSFlavor {
    OSType fileType;
    OSType fileCreator;
    UInt16 fdFlags;
    FlavorType promisedFlavor;
};
typedef struct PromiseHFSFlavor PromiseHFSFlavor;
```

Fields

`fileType`
The file type of the object.

`fileCreator`
The file creator of the object.

`fdFlags`
The Finder flags of the object.

`promisedFlavor`

The flavor type of a separate promise flavor to contain the `FSSpec` structure for the new file. Apple recommends that you use the `kDragPromisedFlavor` type in this field.

Discussion
The promise HFS flavor type is used when you wish to create a new file when dragging to the Finder. The flavor consists of an array of `PromiseHFSFlavor` structures, with the first entry being the preferred file type you would like to create and subsequent array entries being file types in descending preference. This structure allows you to create the file in your `DragSendDataProcPtr` (page 967) callback and provide the `FSSpec` for the new file at that time.

After providing an `FSSpec`, the Finder will move the new file to the drop location. If you wish to create the file before the drag and provide the `FSSpec` data up front, create the new file in the Temporary Items folder so it does not prematurely appear in an open Finder window.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Drag.h`

DragDrawingUPP

Defines a universal procedure pointer (UPP) to a drag drawing callback.

```
typedef DragDrawingProcPtr DragDrawingUPP;
```

Discussion
For more information, see the description of the `DragDrawingProcPtr` (page 963) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Drag.h`

DragInputUPP

Defines a universal procedure pointer (UPP) to a drag input callback.

```
typedef DragInputProcPtr DragInputUPP;
```

Discussion
For more information, see the description of the `DragInputProcPtr` (page 964) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Drag.h`

DragReceiveHandlerUPP

Defines a universal procedure pointer (UPP) to a drag receive handler.

```
typedef DragReceiveHandlerProcPtr DragReceiveHandlerUPP;
```

Discussion
For more information, see the description of the `DragReceiveHandlerProcPtr` (page 966) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Drag.h`

DragSendDataUPP

Defines a universal procedure pointer (UPP) to a drag send data callback.

```
typedef DragSendDataProcPtr DragSendDataUPP;
```

Discussion
For more information, see the description of the `DragSendDataProcPtr` (page 967) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Drag.h`

DragTrackingHandlerUPP

Defines a universal procedure pointer (UPP) to a drag tracking handler.

```
typedef DragTrackingHandlerProcPtr DragTrackingHandlerUPP;
```

Discussion
For more information, see the description of the `DragTrackingHandlerProcPtr` (page 968) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Drag.h`

Constants

Drag Attributes

Provide additional information about a drag that is in progress.

```
typedef UInt32 DragAttributes;
enum {
    kDragHasLeftSenderWindow = (1L << 0),
    kDragInsideSenderApplication = (1L << 1),
    kDragInsideSenderWindow = (1L << 2) };
```

Constants

`kDragHasLeftSenderWindow`
Set if the drag has left the source window since the beginning of the drag. This flag is useful for providing window highlighting after the user has moved the mouse outside of the source window.
Available in Mac OS X v10.0 and later.
Declared in `Drag.h`.

`kDragInsideSenderApplication`
Set if the drag is currently in any window that belongs to the application that started the drag.
Available in Mac OS X v10.0 and later.
Declared in `Drag.h`.

`kDragInsideSenderWindow`
Set if the drag is currently in the same window that the drag started from.
Available in Mac OS X v10.0 and later.
Declared in `Drag.h`.

Discussion
The attribute flags defined by the `DragAttributes` type provide information about the window and application that the drag is currently occurring in. During a drag, the current drag attributes can be obtained by calling the function `GetDragAttributes` (page 930).

Drag Behaviors

Specify the current zoomback behavior of a drag.

```
typedef UInt32 DragBehaviors;
enum {
    kDragBehaviorNone = 0,
    kDragBehaviorZoomBackAnimation = (1L << 0) };
```

Constants

`kDragBehaviorNone`
The Drag Manager performs no animation for a failed drag.
Available in Mac OS X v10.0 and later.
Declared in `Drag.h`.

`kDragBehaviorZoomBackAnimation`
The Drag Manager performs zoomback animation for a failed drag. This behavior is normally enabled.
Available in Mac OS X v10.0 and later.
Declared in `Drag.h`.

Discussion
To change the behavior associated with a drag reference, use the `ChangeDragBehaviors` (page 924) function.

Drag Drawing Messages

Define messages that may be sent to your drag drawing callback.

```
typedef SInt16 DragRegionMessage;
enum {
    kDragRegionBegin = 1,
    kDragRegionDraw = 2,
    kDragRegionHide = 3,
    kDragRegionIdle = 4,
    kDragRegionEnd = 5
};
```

Constants

`kDragRegionBegin`

Your drag drawing callback function receives this message when a drag is being started and it is time to initialize your drawing function. You should not draw anything to the screen when you receive this message.The `showRegion` and `showOrigin` parameters to your drag drawing callback function contain the drag region and the `mouseDown` location, respectively, that were specified to the `TrackDrag` function. The `mouseDown` location is the origin of the drag region.The `hideRegion` parameter is `NULL` when your drag drawing callback function receives this message.

Available in Mac OS X v10.0 and later.

Declared in `Drag.h`.

`kDragRegionDraw`

Your drag drawing callback receives this message when you should move your drag region from the area of the screen defined by the `hideRegion` parameter to the area of the screen defined by the `showRegion` parameter.The `showRegion` parameter contains the drag region that was passed to the `TrackDrag` function, offset to the current pinned mouse location. This region represents the area of the screen that must be drawn into.The `hideRegion` parameter contains the drag region as it is currently visible on the screen from the last call with a `dragRegionDraw` message. This region represents the area of the screen that must be restored. Any part of the drag region that was previously obscured by a call with the `dragRegionHide` message is not included in this `hideRegion` parameter.

Available in Mac OS X v10.0 and later.

Declared in `Drag.h`.

`kDragRegionHide`

Your drag drawing callback receives this message when you should remove part of the drag region from the screen. You receive this message when the drag has ended or when part of the region must be obscured for drawing operations to occur underneath the drag region.The `showRegion` parameter is `NULL` when your drag drawing callback function receives this message.The `hideRegion` parameter contains the part of the currently visible drag region that must be removed from the screen.

Available in Mac OS X v10.0 and later.

Declared in `Drag.h`.

`kDragRegionIdle`

Your drag drawing callback receives this message when the drag region has not moved on the screen and no drawing is necessary. You can use this message if animation of the drag region is necessary.The `showRegion` parameter contains the drag region as it is currently visible on the screen.The `hideRegion` parameter is `NULL` when your drag drawing callback receives this message.

Available in Mac OS X v10.0 and later.

Declared in `Drag.h`.

`kDragRegionEnd`

Your drag drawing callback receives this message when the drag has completed and it is time to deallocate any allocations made from within your drag drawing callback. Your drag drawing callback will have already received a `dragRegionHide` message to hide the entire drag region before receiving this message. After you receive this message, your drag drawing callback will not be called again for the duration of the drag.Both the `showRegion` and `hideRegion` parameters are `NULL` when your drag drawing callback function receives this message.

Available in Mac OS X v10.0 and later.

Declared in `Drag.h`.

Discussion
See `DragDrawingProcPtr` (page 963) for more information on drag drawing callback functions.

Drag Tracking Messages

Define messages that may be sent to your drag tracking handler.

```
typedef SInt16 DragTrackingMessage;
enum {
    kDragTrackingEnterHandler = 1,
    kDragTrackingEnterWindow = 2,
    kDragTrackingInWindow = 3,
    kDragTrackingLeaveWindow = 4,
    kDragTrackingLeaveHandler = 5
};
```

Constants
`kDragTrackingEnterHandler`

Your tracking handler receives this message when the focus of a drag enters a window that is handled by your tracking handler. If the user moves the drag directly to another window that is handled by the same tracking handler, a second `kDragTrackingEnterHandler` message is not received. Your tracking handler only receives this message when the drag enters the domain of your function after leaving another.

Available in Mac OS X v10.0 and later.

Declared in `Drag.h`.

`kDragTrackingEnterWindow`

Your tracking handler receives this message when a drag enters any window that is handled by your tracking handler. This message is sent to your tracking handler for each window that the drag may enter. Your tracking handler will always receive this message within a pair of `kDragTrackingEnterHandler` and `kDragTrackingLeaveHandler` messages.

Available in Mac OS X v10.0 and later.

Declared in `Drag.h`.

`kDragTrackingInWindow`

Your tracking handler receives this message as the user is dragging within a window handled by your tracking handler. You can use this message to track the dragging process through your window. Your tracking handler will always receive this message within a pair of `kDragTrackingEnterWindow` and `kDragTrackingLeaveWindow` messages.Your tracking handler would typically draw the majority of your window highlighting and track objects in your window when you receive this message from the Drag Manager.

Available in Mac OS X v10.0 and later.

Declared in `Drag.h`.

`kDragTrackingLeaveWindow`

Your tracking handler receives this message when a drag leaves any window that is handled by your tracking handler. You are guaranteed to receive this message after receiving a corresponding `kDragTrackingEnterWindow` message. Your tracking handler will always receive this message within a pair of `kDragTrackingEnterHandler` and `kDragTrackingLeaveHandler` messages.

Available in Mac OS X v10.0 and later.

Declared in `Drag.h`.

`kDragTrackingLeaveHandler`

Your tracking handler receives this message when the focus of a drag enters a window that is not handled by your tracking handler. Your tracking handler is guaranteed to receive this message after receiving a corresponding `kDragTrackingEnterHandler` message.

Available in Mac OS X v10.0 and later.

Declared in `Drag.h`.

Discussion

See `DragTrackingHandlerProcPtr` (page 968) for more information on drag tracking handlers.

Flavor Flags

Provide additional information about drag item flavors.

```
typedef UInt32 FlavorFlags;
enum {
    flavorSenderOnly = (1 << 0),
    flavorSenderTranslated = (1 << 1),
    flavorNotSaved = (1 << 2),
    flavorSystemTranslated = (1 << 8),
    flavorDataPromised = (1 << 9) };
```

Constants

`flavorSenderOnly`

Set by the sender if the flavor should only be available to the sender of a drag. If this flag is set when adding the flavor to a drag, no Drag Manager clients other than the sender can receive this flavor.

Available in Mac OS X v10.0 and later.

Declared in `Drag.h`.

`flavorSenderTranslated`
: Set by the sender if the flavor data is translated by the sender. This flag is useful to a receiver if the receiver needs to determine if the sender is performing its own translation to generate this data type. Typically, receivers that store dragged data without interpreting each data type do not store translated types. Flavor types marked with this flag are not stored by the Finder in clipping files.

  Available in Mac OS X v10.0 and later.

  Declared in `Drag.h`.

`flavorNotSaved`
: Set by the sender if the flavor data should not be stored by the receiver. This flag is useful for marking flavor data that will become stale after the drag has completed. Receivers that store dragged data should not store flavors that are marked with this flag. Flavor types marked with this flag are not stored by the Finder in clipping files.

  Available in Mac OS X v10.0 and later.

  Declared in `Drag.h`.

`flavorSystemTranslated`
: Set if the flavor data is provided by the Translation Manager. If this flavor is requested, the Drag Manager will obtain any required data types from the sender and then it will use the Translation Manager to provide the data that the receiver requested. Typically, receivers that store dragged data without interpreting each data type do not store translated types. Flavor types marked with this flag are not stored by the Finder in clipping files.

  Available in Mac OS X v10.0 and later.

  Declared in `Drag.h`.

`flavorDataPromised`
: Set by the sender if the flavor data is promised at a later time.

  Available in Mac OS X v10.1 and later.

  Declared in `Drag.h`.

Discussion
These constants are used when calling the `AddDragItemFlavor` (page 923) function and can be obtained by calling the `GetFlavorFlags` (page 937) function.

flavorTypeDirectory

Represents a special flavor type for AOCE directory specifications.

```
enum {
    flavorTypeDirectory = 'diry'
};
```

Constants
`flavorTypeDirectory`
: The flavor type for a AOCE directory specification. Refer to the AOCE documentation for a definition of the `DSSpec` data structure.

  Available in Mac OS X v10.0 through Mac OS X v10.4.

  Declared in `Drag.h`.

Drag Actions

Specify the actions that should be or have been performed on the data in a drag.

```
enum {
    kDragActionNothing = 0,
    kDragActionCopy = 1L,
    kDragActionAlias = (1L << 1),
    kDragActionGeneric = (1L << 2),
    kDragActionPrivate = (1L << 3),
    kDragActionMove = (1L << 4),
    kDragActionDelete = (1L << 5),
    kDragActionAll = 0xFFFFFFFF };
typedef UInt32 DragActions;
```

Constants

`kDragActionNothing`
Nothing should be or has been done with the data in the drag. When set as an allowable action for remote drags, the drag is not sent to applications other than the drag sender.

Available in Mac OS X v10.1 and later.

Declared in `Drag.h`.

`kDragActionCopy`
The data contained in the drag can be or has been copied.

Available in Mac OS X v10.1 and later.

Declared in `Drag.h`.

`kDragActionAlias`
The data contained in the drag can be or has been shared.

Available in Mac OS X v10.1 and later.

Declared in `Drag.h`.

`kDragActionGeneric`
When set by the drag sender, suggests that the drag receiver can determine the drag action. When returned by the drag receiver, indicates that the receiver did not define a drag action.

Available in Mac OS X v10.1 and later.

Declared in `Drag.h`.

`kDragActionPrivate`
Suggests that the drag action should be negotiated privately between the drag source and destination.

Available in Mac OS X v10.1 and later.

Declared in `Drag.h`.

`kDragActionMove`
The data contained in the drag can be or has been moved.

Available in Mac OS X v10.1 and later.

Declared in `Drag.h`.

`kDragActionDelete`
The data contained in the drag can be or has been deleted.

Available in Mac OS X v10.1 and later.

Declared in `Drag.h`.

`kDragActionAll`
Indicates that all of the above drag actions are allowed.

Available in Mac OS X v10.1 and later.

Declared in `Drag.h`.

Discussion
The drag sender can use these constants to indicate what actions are allowable on the data contained within a drag. The drag receiver can use these constants to indicate what, if any, action was performed on the drag.

Some of the drag actions defined here enforce a mode of operation, while others are suggestions. The `DragActions` constants are used in conjunction with the `GetDragAllowableActions` (page 929), `SetDragAllowableActions` (page 948), `GetDragDropAction` (page 931), and `SetDragDropAction` (page 949) functions. Using drag actions increases compatibility with the Cocoa drag operation model.

HFS Flavor Types

Identify flavor types for file system objects.

```
enum {
    kDragFlavorTypeHFS = 'hfs ',
    kDragFlavorTypePromiseHFS = 'phfs',
    flavorTypeHFS = kDragFlavorTypeHFS,
    flavorTypePromiseHFS = kDragFlavorTypePromiseHFS
};
```

Constants
`kDragFlavorTypeHFS`
The flavor type for an HFS file system object. The Finder uses HFS flavors when dragging existing file system objects. The HFS flavor data is defined by the data type `HFSFlavor` (page 971).

Available in Mac OS X v10.0 and later.

Declared in `Drag.h`.

`kDragFlavorTypePromiseHFS`
The flavor type for promising an HFS file system object to the receiver of the drag. This flavor type can be used when a file could be created if the destination of the drag can accept file system objects. The data type `PromiseHFSFlavor` (page 971) is used to access the information in this flavor type.

Available in Mac OS X v10.0 and later.

Declared in `Drag.h`.

`flavorTypeHFS`
Use `kDragFlavorTypeHFS` instead.

Available in Mac OS X v10.0 and later.

Declared in `Drag.h`.

`flavorTypePromiseHFS`
Use `kDragFlavorTypePromiseHFS` instead.

Available in Mac OS X v10.0 and later.

Declared in `Drag.h`.

Promised Flavor Types

Identify flavor types for the `PromiseHFSFlavor` structure.

```
enum {
    kDragPromisedFlavorFindFile = 'rWm1',
    kDragPromisedFlavor = 'fssP'
};
```

Constants

`kDragPromisedFlavorFindFile`
The value of the `promisedFlavor` field of the `PromiseHFSFlavor` structure for Find File.

Available in Mac OS X v10.0 and later.

Declared in `Drag.h`.

`kDragPromisedFlavor`
The value of the `promisedFlavor` field of the `PromiseHFSFlavor` structure for all other file system objects.

Available in Mac OS X v10.0 and later.

Declared in `Drag.h`.

Type and Creator Constants for Volumes and Directories

Define a creator code and file types for flavor data referring to a volume or directory.

```
enum {
    kDragPseudoCreatorVolumeOrDirectory = 'MACS',
    kDragPseudoFileTypeVolume = 'disk',
    kDragPseudoFileTypeDirectory = 'fold'
};
```

Constants

`kDragPseudoCreatorVolumeOrDirectory`
The "creator type" for volumes and directories. If the data in a drag containing `kDragFlavorTypeHFS` data refers to a folder or volume, the `fileCreator` field of the `HFSFlavor` structure should be set to this value.

Available in Mac OS X v10.0 and later.

Declared in `Drag.h`.

`kDragPseudoFileTypeVolume`
The value of the `fileType` field of the `HFSFlavor` structure for a volume.

Available in Mac OS X v10.0 and later.

Declared in `Drag.h`.

`kDragPseudoFileTypeDirectory`
The value of the `fileType` field of the `HFSFlavor` structure for a directory.

Available in Mac OS X v10.0 and later.

Declared in `Drag.h`.

Standard Drop Locations

Define common drop locations.

```
enum {
    kDragStandardDropLocationTrash = 'trsh',
    kDragStandardDropLocationUnknown = 'unkn'
};
typedef OSType StandardDropLocation;
```

Constants

`kDragStandardDropLocationTrash`

Set when a drag is dropped on the trash icon. Setting this standard drop location automatically sets the traditional drop location to an alias to the trash folder.

Available in Mac OS X v10.2 and later.

Declared in `Drag.h`.

`kDragStandardDropLocationUnknown`

The receiver did not specify a drop location. This is the default.

Available in Mac OS X v10.2 and later.

Declared in `Drag.h`.

Discussion

These values are used in conjunction with the `GetStandardDropLocation` (page 939) and `SetStandardDropLocation` (page 956) functions.

Drag Image Flags

Specify the appearance of a translucent drag.

```
typedef UInt32 DragImageFlags;
enum {
    kDragRegionAndImage = (1L << 4)
    kDragStandardTranslucency = 0,
    kDragDarkTranslucency = 1,
    kDragDarkerTranslucency = 2,
    kDragOpaqueTranslucency = 3
};
```

Constants

`kDragRegionAndImage`

Add this constant to the transparency levels represented by the following constants to specify that the outline region passed to `TrackDrag` should be drawn on screen, in addition to the translucent drag image.

Available in Mac OS X v10.0 and later.

Declared in `Drag.h`.

`kDragStandardTranslucency`

Use the standard translucency level for the drag image. Currently, this is 65%.

Available in Mac OS X v10.0 and later.

Declared in `Drag.h`.

`kDragDarkTranslucency`
Use 50% translucency for the drag image.

Available in Mac OS X v10.0 and later.

Declared in `Drag.h`.

`kDragDarkerTranslucency`
Use 25% transparency for the drag image.

Available in Mac OS X v10.0 and later.

Declared in `Drag.h`.

`kDragOpaqueTranslucency`
Use an opaque drag image (0% translucency).

Available in Mac OS X v10.0 and later.

Declared in `Drag.h`.

Discussion
These constants are used in conjunction with the `SetDragImageWithCGImage` (page 951)and `SetDragImage` (page 950) functions to specify the appearance of the drag image.

Finder Flavor Types

Identify flavor types for the Finder.

```
enum {
    kFlavorTypeClippingName = 'clnm',
    kFlavorTypeClippingFilename = 'clfn',
    kFlavorTypeUnicodeClippingName = 'ucln',
    kFlavorTypeUnicodeClippingFilename = 'uclf',
    kFlavorTypeDragToTrashOnly = 'fdtt',
    kFlavorTypeFinderNoTrackingBehavior = 'fntb'
};
```

Constants
`kFlavorTypeClippingName`
The flavor of a name hint for a clipping file. This flavor type is preferred over the `kFlavorTypeClippingFilename` type.

Available in Mac OS X v10.0 and later.

Declared in `Drag.h`.

`kFlavorTypeClippingFilename`
The flavor of the name of a clipping file.

Available in Mac OS X v10.0 and later.

Declared in `Drag.h`.

`kFlavorTypeUnicodeClippingName`
The flavor of a hint for the unicode name of a clipping file. This flavor type is preferred over the `kFlavorTypeUnicodeClippingFilename` type.

Available in Mac OS X v10.2 and later.

Declared in `Drag.h`.

`kFlavorTypeUnicodeClippingFilename`
The flavor of the unicode name of a clipping file.

Available in Mac OS X v10.2 and later.

Declared in `Drag.h`.

`kFlavorTypeDragToTrashOnly`
Specify this flavor to allow dragging private data to the trash.

Available in Mac OS X v10.0 and later.

Declared in `Drag.h`.

`kFlavorTypeFinderNoTrackingBehavior`
A flavor type indicating that the Finder should ignore the drag.

Available in Mac OS X v10.0 and later.

Declared in `Drag.h`.

Zoom Acceleration Constants

Specify acceleration constants for the `ZoomRects` and `ZoomRegion` functions.

```
typedef SInt16 ZoomAcceleration;
enum {
    kZoomNoAcceleration = 0,
    kZoomAccelerate = 1,
    kZoomDecelerate = 2
};
```

Constants

`kZoomNoAcceleration`
Use linear interpolation for each frame of animation between the source and destination.

Available in Mac OS X v10.0 and later.

Declared in `Drag.h`.

`kZoomAccelerate`
Increment the step size for each frame of animation between the source and destination. This option produces the visual appearance of the animation speeding up as it approaches the destination.

Available in Mac OS X v10.0 and later.

Declared in `Drag.h`.

`kZoomDecelerate`
Decrement the step size for each frame of animation between the source and destination. This option produces the visual appearance of the animation slowing down as it approaches the destination.

Available in Mac OS X v10.0 and later.

Declared in `Drag.h`.

zoomNoAcceleration

Obsolete. Use "Zoom Acceleration Constants" instead.

```
enum {
    zoomNoAcceleration = kZoomNoAcceleration,
    zoomAccelerate = kZoomAccelerate,
    zoomDecelerate = kZoomDecelerate
};
```

kDragStandardImage

Obsolete. Use "Drag Image Flags" instead.

```
enum {
    kDragStandardImage = kDragStandardTranslucency,
    kDragDarkImage = kDragDarkTranslucency,
    kDragDarkerImage = kDragDarkerTranslucency,
    kDragOpaqueImage = kDragOpaqueTranslucency
};
```

dragTrackingEnterHandler

Obsolete. Use "Drag Tracking Messages" instead.

```
enum {
    dragTrackingEnterHandler = kDragTrackingEnterHandler,
    dragTrackingEnterWindow = kDragTrackingEnterWindow,
    dragTrackingInWindow = kDragTrackingInWindow,
    dragTrackingLeaveWindow = kDragTrackingLeaveWindow,
    dragTrackingLeaveHandler = kDragTrackingLeaveHandler
};
```

dragRegionBegin

Obsolete. Use "Drag Drawing Messages" instead.

```
enum {
    dragRegionBegin = kDragRegionBegin,
    dragRegionDraw = kDragRegionDraw,
    dragRegionHide = kDragRegionHide,
    dragRegionIdle = kDragRegionIdle,
    dragRegionEnd = kDragRegionEnd
};
```

dragHasLeftSenderWindow

Obsolete. Use "Drag Attributes" instead.

```
enum {
    dragHasLeftSenderWindow = kDragHasLeftSenderWindow,
    dragInsideSenderApplication = kDragInsideSenderApplication,
    dragInsideSenderWindow = kDragInsideSenderWindow
};
```

Result Codes

The table below lists the most common result codes returned by the Drag Manager.

Result Code	Value	Description
`badDragRefErr`	-1850	Unknown drag reference Available in Mac OS X v10.0 and later.
`badDragItemErr`	-1851	Unknown drag item reference Available in Mac OS X v10.0 and later.
`badDragFlavorErr`	-1852	Unknown flavor type Available in Mac OS X v10.0 and later.
`duplicateFlavorErr`	-1853	Flavor type already exists Available in Mac OS X v10.0 and later.
`cantGetFlavorErr`	-1854	Error while trying to get flavor data Available in Mac OS X v10.0 and later.
`duplicateHandlerErr`	-1855	Handler already exists Available in Mac OS X v10.0 and later.
`handlerNotFoundErr`	-1856	Handler not found Available in Mac OS X v10.0 and later.
`dragNotAcceptedErr`	-1857	Drag was not accepted by receiver Available in Mac OS X v10.0 and later.
`unsupportedForPlatformErr`	-1858	Call is for PowerPC only Available in Mac OS X v10.0 and later.
`noSuitableDisplaysErr`	-1859	No displays support translucency Available in Mac OS X v10.0 and later.
`badImageRgnErr`	-1860	Bad translucent image region Available in Mac OS X v10.0 and later.
`badImageErr`	-1861	Bad translucent image PixMap Available in Mac OS X v10.0 and later.

Result Code	Value	Description
`nonDragOriginatorErr`	-1862	Illegal attempt to access originator only data Available in Mac OS X v10.0 and later.

Gestalt Constants

You can check for version and feature availability information by using the Drag Manager selectors defined in the Gestalt Manager. For more information see *Inside Mac OS X: Gestalt Manager Reference*

Event Manager Reference (Not Recommended)

Framework: Carbon/Carbon.h

Declared in Events.h

Overview

Important: The Event Manager is a legacy System 7 technology. You should use the Mac OS X Carbon Event Manager instead. See *Carbon Event Manager Programming Guide*.

The Event Manager is a legacy System 7 technology that was created to support the cooperative, multitasking environment available on Macintosh computers at the time. This environment allowed users to switch between many open applications and allows other applications to receive background processing time.

The Carbon Event Manager, introduced in Mac OS X, offers a simple yet flexible approach to event handling that greatly reduces the amount of code needed to write a basic application. Moreover, the Carbon Event Manager's streamlined event handling enhances system performance on Mac OS X through more efficient allocation of processing time. Applications that use the Carbon Event Manager not only run better on Mac OS X, they help improve overall performance and responsiveness.

Since the introduction of the Macintosh computer, Mac applications have used the Event Manager to receive information about actions performed by the user, to receive notices of changes in their processing, and to communicate with other applications. For example, an application can retrieve information from the Event Manager about whether the user has pressed a key or the mouse button, whether one of the application's windows needs updating, or whether some other hardware-related or software-related action requires a response from the application.

Applications also used the Event Manager to support the cooperative, multitasking environment available on versions of the Mac OS that preceded Mac OS X. This environment allows users to switch between many open applications and allows other applications to receive background processing time. By using Event Manager routines, an application allowed the system software to coordinate the scheduling of processing time between it and other applications.

Carbon supports the majority of the Event Manager.

High-level events APIs (as contained in `EPPC.h`) are not supported. You should use Apple events instead.

Carbon does not support the `diskEvt` event. Support for volume mount and unmount events will be available in the Carbon Event Manager.

Carbon does not set the `convertClipboardFlag` in the `EventRecord` to indicate that the scrap has changed while the application was suspended. You should call the Scrap Manager function `GetCurrentScrap` instead.

Low-level event queue functions, such as `GetEvQHdr` and `PPostEvent`, are no longer supported.

Application-defined function-key procedures are not supported in Carbon.

Functions by Task

Accessors for Low-Memory Globals

`LMGetKeyRepThresh` (page 1000)
Returns the low-memory auto-key rate.

`LMGetKeyThresh` (page 1000)
Returns the low-memory auto-key threshold.

`LMSetKeyRepThresh` (page 1002)
Sets the low-memory auto-key rate.

`LMSetKeyThresh` (page 1002)
Sets the low-memory auto-key threshold.

`LMGetKbdType` (page 1000)
Returns a value that specifies the physical keyboard type.

`LMGetKbdLast` (page 999)
Returns a value that specifies the last physical keyboard type used.

`LMSetKbdLast` (page 1001)
Sets a value that specifies the last physical keyboard type used.

`LMSetKbdType` (page 1001)
Sets the keyboard type.

Getting Timing Information

`GetCaretTime` (page 993)
Obtains the suggested difference in ticks that should exist between blinks of the caret (usually a vertical bar marking the insertion point) in editable text.

`GetDblTime` (page 994)
Determines whether a sequence of mouse events constitutes a double click.

Making Keyboard Settings

`KeyScript` (page 997) Deprecated in Mac OS X v10.5
Changes the keyboard script (the script system used for keyboard input), changes the keyboard layout (the mapping of keys to characters) or input method within the current keyboard script (a facility for entering 2-byte characters), or makes a setting related to text input, using the supplied value.

Reading the Keyboard

`GetKeys` (page 995)
Obtains the current state of the keyboard.

`KeyTranslate` (page 998)
Converts a virtual key code to a character code based on a `'KCHR'` resource.

`GetCurrentKeyModifiers` (page 994)
Returns the current state of the keyboard modifier keys.

`IsCmdChar` (page 997)
Tests whether the Command key is pressed in conjunction with another key (or keys) that could generate the specified test character.

Receiving Events

`EventAvail` (page 992)
Retrieves the next available event from the Event Manager without removing the returned event from your application's event stream.

`FlushEvents` (page 993)
Removes low-level events from the Operating System event queue.

`GetNextEvent` (page 996)
Retrieves events one at a time from the Event Manager.

`SetEventMask` (page 1003)
Sets the system event mask of your application to the specified mask.

`WaitNextEvent` (page 1004)
Retrieves events one at a time from the Event Manager.

Sending Events

`PostEvent` (page 1002)
Posts events into the Operating System event queue.

Miscellaneous

`CheckEventQueueForUserCancel` (page 991)
Checks the event queue for an user-cancel event.

`GetGlobalMouse` (page 995)
Obtains the position of the mouse, in global coordinates.

Functions

CheckEventQueueForUserCancel

Checks the event queue for an user-cancel event.

```
Boolean CheckEventQueueForUserCancel (
   void
);
```

Return Value
Returns `true` if a user-cancel event is in the queue, `false` otherwise.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CarbonEvents.h`

EventAvail

Retrieves the next available event from the Event Manager without removing the returned event from your application's event stream.

```
Boolean EventAvail (
   EventMask eventMask,
   EventRecord *theEvent
);
```

Parameters

eventMask

A value that indicates which kinds of events are to be returned; this parameter is interpreted as a sum of event mask constants. You specify the event mask using one or more of the values defined by the "Event Mask Constants" (page 1017). If no event of any of the designated types is available, `EventAvail` returns a null event.

theEvent

A pointer to an event structure for the next available event of the specified type or types. The `EventAvail` function does not remove the returned event from the event stream, but does return the information about the event in an event structure. The event structure includes the type of event received and other information.

Return Value
`EventAvail` returns `false` as its function result if the event being returned is a null event; otherwise, `EventAvail` returns `true`.

Special Considerations

If `EventAvail` returns a low-level event from the Operating System event queue, the event will not be accessible later if, in the meantime, the event queue becomes full and the event is discarded from it; however, this is not a common occurrence.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
Simple DrawSprocket

Declared In
`Events.h`

FlushEvents

Removes low-level events from the Operating System event queue.

```
void FlushEvents (
   EventMask whichMask,
   EventMask stopMask
);
```

Parameters

whichMask

A value that indicates which kinds of low-level events are to be removed from the Operating System event queue; this parameter is interpreted as a sum of event mask constants. You specify the event mask using one or more of the values defined in "Event Mask Constants" (page 1017). The `whichMask` and `stopMask` parameters together specify which events to remove.

stopMask

A value that limits which low-level events are to be removed from the Operating System event queue; this parameter is interpreted as a sum of event mask constants. You specify the event mask using one or more of the values defined in "Event Mask Constants" (page 1017). `FlushEvents` does not remove any low-level events that are specified by the `stopMask` parameter. To remove all events specified by the `whichMask` parameter, specify 0 as the `stopMask` parameter.

Discussion

`FlushEvents` removes only low-level events stored in the Operating System event queue; it does not remove activate, update, operating-system, or high-level events. `FlushEvents` does not remove any types of events not stored in the Operating System event queue.

You can choose to use the `FlushEvents` function when your application first starts to empty the Operating System event queue of any keystrokes or mouse events generated by the user while the Finder loaded your application. In general, however, your application should not empty the queue at any other time as this loses user actions and makes your application and the computer appear unresponsive to the user.

You specify which low-level events to remove using the `whichMask` and `stopMask` parameters. `FlushEvents` removes the low-level events specified by the `whichMask` parameter, up to but not including the first event of any type specified by the `stopMask` parameter.

If the event queue doesn't contain any of the events specified by the `whichMask` parameter, `FlushEvents` does not remove any events from the queue.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

HideMenuBar

ictbSample

Declared In

`Events.h`

GetCaretTime

Obtains the suggested difference in ticks that should exist between blinks of the caret (usually a vertical bar marking the insertion point) in editable text.

```
UInt32 GetCaretTime (
   void
);
```

Return Value
The blink delay, in ticks.

Discussion
If your application supports editable text, your application should use the value returned by `GetCaretTime` to determine how often to blink the caret. If your application uses only TextEdit, you can use TextEdit functions to automatically blink the caret at the time interval that the user specifies in the General Controls panel.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Events.h`

GetCurrentKeyModifiers

Returns the current state of the keyboard modifier keys.

```
UInt32 GetCurrentKeyModifiers (
   void
);
```

Parameters

Return Value
A bit mask indicating which keyboard modifier keys are pressed. See "Event Modifier Constants" (page 1010) for a list of possible values.

Discussion
`GetCurrentKeyModifiers` provides a more convenient way to get the modifier key state than calling `GetNextEvent`. It returns a value whose individual bits indicate which keyboard modifier keys are currently being pressed. You can test for the Caps Lock, Shift, Control, Option, and Command keys.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`CarbonEventsCore.h`

GetDblTime

Determines whether a sequence of mouse events constitutes a double click.

```
UInt32 GetDblTime (
   void
);
```

Return Value
The suggested maximum elapsed time, in ticks, between a mouse-up event and a mouse-down event.

Discussion
The `GetDblTime` function returns the suggested maximum elapsed time, in ticks, between a mouse-up event and a mouse-down event. The user can adjust this value using the Mouse control panel.

If your application distinguishes a double click of the mouse from a single click, your application should use the value returned by `GetDblTime` to make this distinction. If your application uses TextEdit, the TextEdit functions automatically recognize and handle double clicks of text within a TextEdit edit structure by appropriately highlighting or unhighlighting the selection.

The ratio of ticks to value in the `DoubleTime` global variable is 1:1. However, the Finder multiplies `DoubleTime` by 2 to determine double click time because it needs to account for user problems that typically occur during icon arrangement. Therefore, the Finder uses `DoubleTime*2` whereas the rest of the system uses `DoubleTime`.

Incidentally, the Finder does not limit the `DoubleTime` to 64 ticks. In most places, it treats it like a byte although in some others it treats it like a longword. The best method would be to provide a one-second double-byte (two seconds in the Finder).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Events.h`

GetGlobalMouse

Obtains the position of the mouse, in global coordinates.

```
void GetGlobalMouse (
   Point *globalMouse
);
```

Parameters

`globalMouse`
The position of the mouse, as a global point.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`CarbonEventsCore.h`

GetKeys

Obtains the current state of the keyboard.

```
void GetKeys (
   KeyMap theKeys
);
```

Parameters

theKeys

On output, the current state of the keyboard, including the keypad, if any.

Discussion

You can use the `GetKeys` function to determine the current state of the keyboard at any time. For example, you can determine whether one of the modifier keys is down by itself or in combination with another key using the `GetKeys` function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Events.h`

GetNextEvent

Retrieves events one at a time from the Event Manager.

```
Boolean GetNextEvent (
   EventMask eventMask,
   EventRecord *theEvent
);
```

Parameters

eventMask

A value that indicates which kinds of events are to be returned; this parameter is interpreted as a sum of event mask constants. You specify the event mask using one or more of the values defined in "Event Mask Constants" (page 1017). If no event of any of the designated types is available, `GetNextEvent` returns a null event.

theEvent

A pointer to an event structure for the next available event of the specified type or types. The `GetNextEvent` function removes the returned event from the event stream and returns the information about the event in an event structure. The event structure includes the type of event received and other information.

Return Value

Discussion

`GetNextEvent` returns `false` as its function result if the event being returned is a null event or if `GetNextEvent` has intercepted the event; otherwise, `GetNextEvent` returns `true`. The `GetNextEvent` function calls the Operating System Manager function `SystemEvent` to determine whether the event should be handled by the application or the Operating System.

The `GetNextEvent` function also intercepts Command–Shift–number key sequences and calls the corresponding `'FKEY'` resource to perform the associated action. The Event Manager's processing of Command–Shift–number key sequences with numbers 3 through 9 can be disabled by setting the `ScrDmpEnable` global variable (a byte) to 0.

Special Considerations

For greater support of the multitasking environment, your application should use `WaitNextEvent` instead of `GetNextEvent` whenever possible.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Events.h`

IsCmdChar

Tests whether the Command key is pressed in conjunction with another key (or keys) that could generate the specified test character.

```
Boolean IsCmdChar (
   const EventRecord *event,
   short test
);
```

Parameters

event

The event record for a key-down or auto-key event with the Command key down.

test

The character you want to test.

Return Value

The function returns `TRUE` if the test character is produced with the current modifier keys, or if it would be produced by changing the current modifier key bits in either or both of the following ways: (1) turning the Command bit off or (2) toggling the Shift bit.

Discussion

This function tests whether the Command key is pressed in conjunction with another key (or keys) that could generate the `test` character for some combination of Command up or down and Shift up or down. This accommodates European keyboards that may have the `test` character as a shifted character, and non-Roman keyboards that will only generate the `test` character if the Command key is pressed. It's most useful for testing for Command-period, but it can test for command-AnyCharacter.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Events.h`

KeyScript

Changes the keyboard script (the script system used for keyboard input), changes the keyboard layout (the mapping of keys to characters) or input method within the current keyboard script (a facility for entering 2-byte characters), or makes a setting related to text input, using the supplied value. (Deprecated in Mac OS X v10.5.)

```
void KeyScript (
   short code
);
```

Parameters

code

If 0 or positive, directly specifies a script system (that is, it is read as a script code). Negative values have special meanings.

The `code` parameter is a selector that can explicitly specify a keyboard script by script code. See the Script Manager for a list of script codes. If the selector specifies a script, then the current default keyboard layout (`'KCHR'` resource) for that script, as specified in the script's international bundle resource, becomes the current keyboard layout.

The selector can also implicitly specify a keyboard script (for example, the next script), a keyboard layout (for example, the previously used keyboard layout in the current script), or an input method (for example, inline input versus window-based input). It can also specify settings that enable or disable keyboard layouts and keyboard scripts, and toggle among input options or line direction.

Discussion

For the purposes of `KeyScript`, keyboard layout means a keyboard-layout (`'KCHR'`) resource, plus optionally a key-remap (`'itlk'`) resource. To change keyboard layouts means to change the current keyboard-layout resource.

If the Keyboard menu is displayed, `KeyScript` also updates the Keyboard menu.

If you call `KeyScript` and explicitly specify a script system that is not available, `KeyScript` does nothing. The current keyboard script remains unchanged.

Note that `Keyscript` also does nothing when passed a positive or zero script code if the user has Font and Keyboard Synchronization turned off. You can find the Font and Synchronization checkbox under Options in the Input Menu tab of the International System Preference in Mac OS X.

Special Considerations

`KeyScript` operates only on those keyboard-layout and key-remap resources that are present in the System file.

Your application's keyboard-menu setting is not maintained by the Process Manager if the state of the keyboard menu is changed while you are switched out, the Process Manager does not restore your setting when you are switched back in. However, the Process Manager does maintain the keyboard disable state (Script Manager variable `smKeyDisableState`) for your application. See the Script Manager for a description of the `smKeyDisableState` variable. `KeyScript` may move memory; your application should not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Events.h`

KeyTranslate

Converts a virtual key code to a character code based on a `'KCHR'` resource.

```
UInt32 KeyTranslate (
   const void *transData,
   UInt16 keycode,
   UInt32 *state
);
```

Parameters

transData

A pointer to the 'KCHR' resource that you want the KeyTranslate function to use when converting the key code to a character code.

keycode

A 16-bit value that your application should set so that bits 0–6 contain the virtual key code and bit 7 contains either 1 to indicate an up stroke or 0 to indicate a down stroke of the key. Bits 8–15 have the same interpretation as the high byte of the modifiers field of the event structure and should be set according to the needs of your application.

state

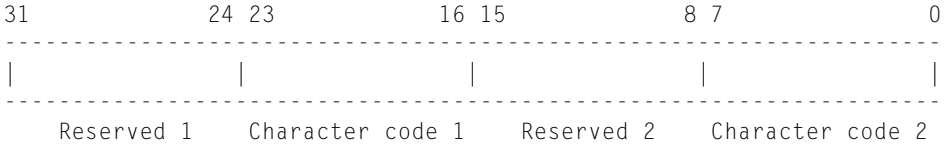
A pointer to a value that your application should set to 0 the first time it calls KeyTranslate or any time your application calls KeyTranslate with a different 'KCHR' resource. Thereafter, your application should pass the same value in the state parameter as KeyTranslate returned in the previous call.

Return Value

A 32-bit value that gives the character code for the virtual key code specified by the keycode parameter.

Discussion

The KeyTranslate function returns the values that correspond to one or possibly two characters that are generated by the specified virtual key code. The following diagram shows the structure of the 32-bit number that KeyTranslate returns.



For example, a given virtual key code might correspond to an alphabetic character with a separate accent character. When the user presses Option-E followed by E, you can map this through the KeyTranslate function using the U.S. 'KCHR' resource to produce é, which KeyTranslate returns as two characters in the bytes labeled Character code 1 and Character code 2.

If KeyTranslate returns only one character code, it is always in the byte labeled Character code 2. However, your application should always check both bytes labeled Character code 1 and Character code 2 for possible values that map to the virtual key code.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

Events.h

LMGetKbdLast

Returns a value that specifies the last physical keyboard type used.

```
UInt8 LMGetKbdLast (
   void
);
```

Return Value
The last physical keyboard type used.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Events.h`

LMGetKbdType

Returns a value that specifies the physical keyboard type.

```
UInt8 LMGetKbdType (
   void
);
```

Return Value
The physical keyboard type.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Events.h`

LMGetKeyRepThresh

Returns the low-memory auto-key rate.

```
SInt16 LMGetKeyRepThresh (
   void
);
```

Return Value
The auto-key rate, that is, the amount of time, in ticks, that must elapse before the Event Manager generates a subsequent auto-key event.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Events.h`

LMGetKeyThresh

Returns the low-memory auto-key threshold.

```
SInt16 LMGetKeyThresh (
   void
);
```

Return Value
Returns the auto-key threshold, that is, the amount of time, in ticks, that must elapse before the Event Manager generates an auto-key event.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Events.h`

LMSetKbdLast

Sets a value that specifies the last physical keyboard type used.

```
void LMSetKbdLast (
   UInt8 value
);
```

Parameters

value
The physical keyboard type you want to set.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Events.h`

LMSetKbdType

Sets the keyboard type.

```
void LMSetKbdType (
   UInt8 value
);
```

Parameters

value
The physical keyboard type you want to set.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Events.h`

LMSetKeyRepThresh

Sets the low-memory auto-key rate.

```
void LMSetKeyRepThresh (
   SInt16 value
);
```

Parameters

value

The low-memory auto-key rate you want to set.

Discussion

`LMSetKeyRepThresh` sets the low-memory auto-key rate, that is, the amount of time, in ticks, that must elapse before the Event Manager generates a subsequent auto-key event.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Events.h`

LMSetKeyThresh

Sets the low-memory auto-key threshold.

```
void LMSetKeyThresh (
   SInt16 value
);
```

Parameters

value

The low-memory auto-key threshold you want to set.

Discussion

`LMSetKeyThresh` sets the low-memory auto-key threshold, that is, the amount of time, in ticks, that must elapse before the Event Manager generates an auto-key event.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Events.h`

PostEvent

Posts events into the Operating System event queue.

```
OSErr PostEvent (
   EventKind eventNum,
   UInt32 eventMsg
);
```

Parameters

eventNum

A value that indicates the type of event to post into the Operating System event queue. You specify the event kind using one or more of these values defined in "Event Mask Constants" (page 1017): `mouseDown`, `mouseUp`, `keyDown`, `keyUp`, `autoKey`, and `diskEvt`. Do not attempt to post any other type of event in the Operating System event queue.

eventMsg

An unsigned integer that contains the contents of the `message` field for the event that `PostEvent` should post in the queue.

Return Value

A result code. See "Event Manager Result Codes" (page 1023).

Discussion

In the `eventNum` and `eventMsg` parameters, you specify the value for the `what` and `message` fields of the event's event structure. The `PostEvent` function fills out the `when`, `where`, and `modifiers` fields of the event structure with the current time, current mouse location, and current state of the modifier keys and mouse button.

The `PostEvent` function posts only events that are enabled by the system event mask. If the event queue is full, `PostEvent` removes the oldest event in the queue and posts the new event.

Note that if you use `PostEvent` to repost an event, the `PostEvent` function fills out the `when`, `where`, and `modifier` fields of the event structure, giving these fields of the reposted event different values from the values contained in the original event.

Do not post any events other than mouse-down, mouse-up, key-down, key-up, auto-key, and disk-inserted events in the Operating System event queue. Attempting to post other events into the Operating System event queue interferes with the internal operation of the Event Manager.

In most cases, your application should not call the `PostEvent` function.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Events.h`

SetEventMask

Sets the system event mask of your application to the specified mask.

```
void SetEventMask (
   EventMask value
);
```

Parameters

value

A value that specifies which events should be posted in the Operating System event queue. You specify the event mask using one or more of the values defined in "Event Mask Constants" (page 1017).

Discussion

Your application should not call the `SetEventMask` function to disable any event types from being posted. Use `SetEventMask` only to enable key-up events if your application needs to respond to key-up events.

The `SetEventMask` function sets the system event mask of your application according to the parameter `value`. The Operating System Event Manager posts only low-level events (other than update or activate events) corresponding to bits in the system event mask of the current process when posting events in the Operating System event queue. The system event mask of an application is initially set to post mouse-up, mouse-down, key-down, auto-key, and disk-inserted events into the Operating System event queue.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Events.h`

WaitNextEvent

Retrieves events one at a time from the Event Manager.

```
Boolean WaitNextEvent (
   EventMask eventMask,
   EventRecord *theEvent,
   UInt32 sleep,
   RgnHandle mouseRgn
);
```

Parameters

eventMask

A value that indicates which kinds of events are to be returned. This parameter is interpreted as a sum of event mask constants. You specify the event mask using values defined in "Event Mask Constants" (page 1017). To accept all events, you can specify the `everyEvent` constant as the event mask.

If no event of any of the designated types is available, `WaitNextEvent` returns a null event. `WaitNextEvent` determines the next available event to return based on the `eventMask` parameter and the priority of the event.

Events not designated by the event mask remain in the event stream until retrieved by an application. Low-level events in the Operating System event queue are kept in the queue until they are retrieved by your application or another application or until the queue becomes full. Once the queue becomes full, the Operating System Event Manager begins discarding the oldest events in the queue.

theEvent

A pointer to an event structure for the next available event of the specified type or types. The `WaitNextEvent` function removes the returned event from the event stream and returns the information about the event in an event structure. The event structure includes the type of event received and other information.

In addition to the event structure, high-level events can contain additional data; use the Apple Event Manager `AEProcessAppleEvent` function to get additional data associated with these events.

sleep

The number of ticks (a tick is approximately 1/60 of a second) indicating the amount of time your application is willing to relinquish the processor if no events are pending for your application. If you specify a value greater than 0 for the `sleep` parameter, your application relinquishes the processor for the specified time or until an event occurs.

You should not set the `sleep` parameter to a value greater than the number of ticks returned by `GetCaretTime` if your application provides text-editing capabilities. When the specified time expires, and if there are no pending events for your application, `WaitNextEvent` returns a `NULL` event in the parameter `theEvent`.

When running on Mac OS X, a Carbon application will block for the entire duration of the `sleep` parameter if there are no events to be delivered. This is slightly different behavior than on Mac OS 9, where the application will often receive `NULL` events before the sleep duration has elapsed.

mouseRgn

A handle to a region that specifies a region inside of which mouse movement does not cause mouse-moved events. In other words, your application receives mouse-moved events only when the cursor is outside the specified region. You should specify the region in global coordinates. If you pass an empty region or a `null` region handle, the Event Manager does not report mouse-moved events to your application. Note that your application should recalculate the `mouseRgn` parameter when it receives a mouse-moved event, or it will continue to receive mouse-moved events as long as the cursor position is outside the original `mouseRgn`.

Return Value

The `WaitNextEvent` function returns `false` as its function result if the event being returned is a null event or if `WaitNextEvent` has intercepted the event; otherwise, `WaitNextEvent` returns `true`.

Discussion

The `WaitNextEvent` function calls the Operating System Event Manager function `SystemEvent` to determine whether the event should be handled by the application or the Operating System.

If no events are pending for your application, `WaitNextEvent` waits for a specified amount of time for an event. (During this time, processing time may be allocated to background processes.) If an event occurs, it is returned through the parameter `theEvent`, and `WaitNextEvent` returns a function result of `true`. If the specified time expires and there are no pending events for your application, `WaitNextEvent` returns a null event in `theEvent` and a function result of `false`.

Before returning an event to your application, `WaitNextEvent` performs other processing and may intercept the event.

The `WaitNextEvent` function intercepts Command–Shift–number key sequences and calls the corresponding `'FKEY '` resource to perform the associated action. The Event Manager's processing of Command–Shift–number key sequences with numbers 3 through 9 can be disabled by setting the `ScrDmpEnable` global variable (a byte) to 0.

If the returned event is a high-level event and your application supports Apple events, use the Apple Event Manager function `AEProcessAppleEvent` to respond to the Apple event and to get additional information associated with the Apple event.

To retrieve an event without removing it from the event stream, use `EventAvail` (page 992).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
HideMenuBar
Simple DrawSprocket

Declared In
`Events.h`

Data Types

EventRecord

Contains information associated with an event.

```
struct EventRecord {
    EventKind what;
    UInt32 message;
    UInt32 when;
    Point where;
    EventModifiers modifiers;
};
typedef struct EventRecord EventRecord;
```

Fields

`what`

The kind of event received. The Event Manager specifies the kind of event with one of the values defined by the `EventKind` enumeration.

`message`

Additional information associated with the event. The interpretation of this information depends on the event type. The contents of the `message` field for each event type are summarized here:

- For a null, mouse-up, or mouse-down event, the event message is:Undefined.
- For a key-up, key-down, or auto-key event, the event message is:The low-order word contains the character code and virtual key code, which you can access with the constants `charCodeMask` and `keyCodeMask`, respectively. For Apple Desktop Bus (ADB) keyboards, the low byte of the high-order word contains the ADB address of the keyboard where the keyboard event occurred. The high byte of the high-order word is reserved.
- For an update or activate event, the event message is:A pointer to the window to update, activate, or deactivate.
- For a disk-inserted event, the event message is:The drive number in the low-order word, the File Manager result code in the high-order word.
- For a resume event, the event message is:The `suspendResumeMessage` enumerator in bits 24–31 and a 1 (the `resumeFlag` enumerator) in bit 0 indicate the event is a resume event. Bit 1 contains a 1 (the `convertClipBoardFlag` enumerator) if Clipboard conversion is required, and bits 2–23 are reserved.
- For a suspend event, the event message is:The `suspendResumeMessage` enumerator in bits 24–31 and a 0 in bit 0 to indicate the event is a suspend event. Bit 1 is undefined, and bits 2–23 are reserved.
- For a mouse-moved event, the event message is:The `mouseMovedMessage` enumerator in bits 24–31. Bits 2–23 are reserved, and bit 0 and bit 1 are undefined.
- For a high-level event, the event message is:

  The class of events to which the high-level event belongs. The `message` and `where` fields of a high-level event define the specific type of high-level event received.

`when`

The `when` field indicates the time when the event was posted (in ticks since system startup).

`where`

For low-level events and operating-system events, the `where` field contains the location of the cursor at the time the event was posted (in global coordinates).

For high-level events, the `where` field contains a second event specifier, the event ID. The event ID defines the particular type of event within the class of events defined by the `message` field of the high-level event. For high-level events, you should interpret the `where` field as having the data type `OSType`, not `Point`.

`modifiers`

The `modifiers` field contains information about the state of the modifier keys and the mouse button at the time the event was posted. For activate events, this field also indicates whether the window should be activated or deactivated. In System 7 it also indicates whether the mouse-down event caused your application to switch to the foreground.

Each of the modifier keys is represented by a specific bit in the `modifiers` field of the event structure. The modifier keys include the Option, Command, Caps Lock, Control, and Shift keys. If your application attaches special meaning to any of these keys in combination with other keys or when the mouse button is down, you can test the state of the `modifiers` field to determine the action your application should take. For example, you can use this information to determine whether the user pressed the Command key and another key to make a menu choice.

Discussion
When your application uses an Event Manager function to retrieve an event, the Event Manager returns information about the retrieved event in an event structure, which is a structure of type `EventRecord`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Events.h`

EvQEl

Defines an event queue entry.

```
struct EvQEl {
    QElemPtr qLink;
    SInt16 qType;
    EventKind evtQWhat;
    UInt32 evtQMessage;
    UInt32 evtQWhen;
    Point evtQWhere;
    EventModifiers evtQModifiers;
};
typedef struct EvQEl EvQEl;
typedef EvQEl * EvQElPtr;
```

Fields

`qLink`

Next queue entry.

`qType`

Queue type (`evType`).

`evtQWhat`

Event code.

`evtQMessage`

Event message.

`evtQWhen`

Ticks since startup.

`evtQWhere`

Mouse location.

`evtQModifiers`
Modifier flags.

Discussion
A structure of type `EvQEl` defines an entry in the Operating System event queue. Each entry in the event queue begins with 4 bytes of flags followed by a pointer to the next queue entry. The flags are maintained by and internal to the Operating System Event Manager. The queue entries are linked by pointers, and the first field of the `EvQEl` data type, which represents the structure of a queue entry, begins with a pointer to the next queue entry. Thus, you cannot directly access the flags using the `EvQEl` data type.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Events.h`

KeyMap

Contains information about the current state of the keyboard.

```
typedef BigEndianLong KeyMap[4];
```

Discussion
The type `KeyMap` is used in `GetKeys` (page 995) to return the current state of the keyboard, including the keypad, if any. The `KeyMap` type is interpreted as an array of 128 elements, each having a Boolean value. Each key on the keyboard or keypad corresponds to an element in the `KeyMap` array. A `KeyMap` element is `true` if the corresponding key is down and `false` if it isn't. The maximum number of keys that can be down simultaneously is two character keys plus any combination of the five modifier keys.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Events.h`

KeyMapByteArray

Contains information about the current state of the keyboard.

```
typedef UInt8 KeyMapByteArray[16];
```

Discussion
The type `KeyMapByteArray` is an alternate version of the type `KeyMap` (page 1009) for use on little endian platforms.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Events.h`

Constants

Event Modifier Constants

Define modifiers for event types.

```
enum {
    activeFlagBit = 0,
    btnStateBit = 7,
    cmdKeyBit = 8,
    shiftKeyBit = 9,
    alphaLockBit = 10,
    optionKeyBit = 11,
    controlKeyBit = 12,
    rightShiftKeyBit = 13,
    rightOptionKeyBit = 14,
    rightControlKeyBit = 15
};
typedef UInt16 EventModifiers;
```

Constants

`activeFlagBit`
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`btnStateBit`
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`cmdKeyBit`
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`shiftKeyBit`
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`alphaLockBit`
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`optionKeyBit`
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`controlKeyBit`
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`rightShiftKeyBit`
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`rightOptionKeyBit`
Available in Mac OS X v10.0 and later.

Declared in `Events.h`.

`rightControlKeyBit`
Available in Mac OS X v10.0 and later.

Declared in `Events.h`.

charCodeMask

```
enum {
    charCodeMask = 0x000000FF,
    keyCodeMask = 0x0000FF00,
    adbAddrMask = 0x00FF0000,
    osEvtMessageMask = 0xFF000000
};
```

Constants

`charCodeMask`
The enumerator indicating you want your application to receive a character-code keyboard event.

Available in Mac OS X v10.0 and later.

Declared in `Events.h`.

`keyCodeMask`
The enumerator indicating you want your application to receive a key-code keyboard event.

Available in Mac OS X v10.0 and later.

Declared in `Events.h`.

`adbAddrMask`
The enumerator indicating you want your application to receive an ADB address if there is an ADB keyboard.

Available in Mac OS X v10.0 and later.

Declared in `Events.h`.

`osEvtMessageMask`
The enumerator indicating you want your application to receive a keyboard event that can be used to extract a message code.

Available in Mac OS X v10.0 and later.

Declared in `Events.h`.

convertClipboardFlag

Obsolete in Carbon.

```
enum {
    convertClipboardFlag = 2
};
```

Constants

`convertClipboardFlag`

Available in Mac OS X v10.0 and later.

Declared in `Events.h`.

Discussion

Obsolete in Carbon. To determine if the clipboard has changed while your application was suspended, use the Scrap Manager function `GetCurrentScrap`.

Carbon Porting Notes

Unsupported. To determine if the clipboard has changed while your application was suspended, use the Scrap Manager function `GetCurrentScrap`.

Event Modifier Bits

Modifer bits for events.

```
enum {
    activeFlag = 1 << activeFlagBit,
    btnState = 1 << btnStateBit,
    cmdKey = 1 << cmdKeyBit,
    shiftKey = 1 << shiftKeyBit,
    alphaLock = 1 << alphaLockBit,
    optionKey = 1 << optionKeyBit,
    controlKey = 1 << controlKeyBit,
    rightShiftKey = 1 << rightShiftKeyBit,
    rightOptionKey = 1 << rightOptionKeyBit,
    rightControlKey = 1 << rightControlKeyBit
};
```

Constants

`activeFlag`

The enumerator that indicates a window is being activated or that a mouse-down event caused a foreground switch.

Available in Mac OS X v10.0 and later.

Declared in `Events.h`.

`btnState`

The enumerator indicating that the mouse button has been released.

Available in Mac OS X v10.0 and later.

Declared in `Events.h`.

`cmdKey`

The enumerator indicating that the Command key is being pressed.

Available in Mac OS X v10.0 and later.

Declared in `Events.h`.

`shiftKey`
The enumerator indicating that the Shift key is being pressed.
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`alphaLock`
The enumerator indicating that the Caps Lock key is being pressed.
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`optionKey`
The enumerator indicating that the Option key is being pressed.
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`controlKey`
The enumerator indicating that the Control key is being pressed.
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`rightShiftKey`
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`rightOptionKey`
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`rightControlKey`
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

HighLevelEventMsgClass

```
enum {
    HighLevelEventMsgClass = 'jaym',
    rtrnReceiptMsgID = 'rtrn'
};
```

Constants

`HighLevelEventMsgClass`
The enumerator indicating a high-level event message class for return receipt.

`rtrnReceiptMsgID`
The posting enumerator indicating the return receipt message ID.

Character Codes

Define character codes for events.

```
enum {
    kNullCharCode = 0,
    kHomeCharCode = 1,
    kEnterCharCode = 3,
    kEndCharCode = 4,
    kHelpCharCode = 5,
    kBellCharCode = 7,
    kBackspaceCharCode = 8,
    kTabCharCode = 9,
    kLineFeedCharCode = 10,
    kVerticalTabCharCode = 11,
    kPageUpCharCode = 11,
    kFormFeedCharCode = 12,
    kPageDownCharCode = 12,
    kReturnCharCode = 13,
    kFunctionKeyCharCode = 16,
    kCommandCharCode = 17,
    kCheckCharCode = 18,
    kDiamondCharCode = 19,
    kAppleLogoCharCode = 20,
    kEscapeCharCode = 27,
    kClearCharCode = 27,
    kLeftArrowCharCode = 28,
    kRightArrowCharCode = 29,
    kUpArrowCharCode = 30,
    kDownArrowCharCode = 31,
    kSpaceCharCode = 32,
    kDeleteCharCode = 127,
    kBulletCharCode = 165,
    kNonBreakingSpaceCharCode = 202
};
```

Constants

`kNullCharCode`
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`kHomeCharCode`
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`kEnterCharCode`
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`kEndCharCode`
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`kHelpCharCode`
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`kBellCharCode`
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`kBackspaceCharCode`
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`kTabCharCode`
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`kLineFeedCharCode`
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`kVerticalTabCharCode`
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`kPageUpCharCode`
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`kFormFeedCharCode`
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`kPageDownCharCode`
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`kReturnCharCode`
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`kFunctionKeyCharCode`
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`kCommandCharCode`
Available in Mac OS X v10.1 and later.
Declared in `Events.h`.

`kCheckCharCode`
Available in Mac OS X v10.1 and later.
Declared in `Events.h`.

`kDiamondCharCode`
Available in Mac OS X v10.1 and later.
Declared in `Events.h`.

`kAppleLogoCharCode`
Available in Mac OS X v10.1 and later.
Declared in `Events.h`.

`kEscapeCharCode`
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`kClearCharCode`
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`kLeftArrowCharCode`
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`kRightArrowCharCode`
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`kUpArrowCharCode`
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`kDownArrowCharCode`
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`kSpaceCharCode`
Available in Mac OS X v10.1 and later.
Declared in `Events.h`.

`kDeleteCharCode`
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`kBulletCharCode`
Available in Mac OS X v10.1 and later.
Declared in `Events.h`.

`kNonBreakingSpaceCharCode`
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

kShiftUnicode

```
enum {
    kShiftUnicode = 0x21E7,
    kControlUnicode = 0x2303,
    kOptionUnicode = 0x2325,
    kCommandUnicode = 0x2318,
    kPencilUnicode = 0x270E,
    kCheckUnicode = 0x2713,
    kDiamondUnicode = 0x25C6,
    kBulletUnicode = 0x2022,
    kAppleLogoUnicode = 0xF8FF
};
```

Constants
`kShiftUnicode`
Available in Mac OS X v10.1 and later.
Declared in `Events.h`.

`kControlUnicode`
Available in Mac OS X v10.1 and later.
Declared in `Events.h`.

`kOptionUnicode`
Available in Mac OS X v10.1 and later.
Declared in `Events.h`.

`kCommandUnicode`
Available in Mac OS X v10.1 and later.
Declared in `Events.h`.

`kPencilUnicode`
Available in Mac OS X v10.1 and later.
Declared in `Events.h`.

`kCheckUnicode`
Available in Mac OS X v10.1 and later.
Declared in `Events.h`.

`kDiamondUnicode`
Available in Mac OS X v10.1 and later.
Declared in `Events.h`.

`kBulletUnicode`
Available in Mac OS X v10.1 and later.
Declared in `Events.h`.

`kAppleLogoUnicode`
Available in Mac OS X v10.1 and later.
Declared in `Events.h`.

Event Mask Constants

Define constants you can use in the event mask.

```
enum {
    mDownMask = 1 << mouseDown,
    mUpMask = 1 << mouseUp,
    keyDownMask = 1 << keyDown,
    keyUpMask = 1 << keyUp,
    autoKeyMask = 1 << autoKey,
    updateMask = 1 << updateEvt,
    diskMask = 1 << diskEvt,
    activMask = 1 << activateEvt,
    highLevelEventMask = 0x0400,
    osMask = 1 << osEvt,
    everyEvent = 0xFFFF
};
typedef UInt16 EventMask;
```

Constants

`mDownMask`
The enumerator indicating you want your application to receive a mouse-down event.
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`mUpMask`
The enumerator indicating you want your application to receive a mouse-up event.
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`keyDownMask`
The enumerator indicating you want your application to receive a key-down event.
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`keyUpMask`
The enumerator indicating you want your application to receive a key-up event.
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`autoKeyMask`
The enumerator indicating you want your application to receive an auto-key event.
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`updateMask`
The enumerator indicating you want your application to receive an update event.
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`diskMask`
The enumerator indicating you want your application to receive a disk-inserted event.
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`activMask`
The enumerator indicating you want your application to receive an activate event.

Available in Mac OS X v10.0 and later.

Declared in `Events.h`.

`highLevelEventMask`
The enumerator indicating you want your application to receive a high-level event.

Available in Mac OS X v10.0 and later.

Declared in `Events.h`.

`osMask`
The enumerator indicating you want your application to receive an operating-system event

Available in Mac OS X v10.0 and later.

Declared in `Events.h`.

`everyEvent`
The enumerator indicating you want your application to receive every event.

Available in Mac OS X v10.0 and later.

Declared in `Events.h`.

mouseMovedMessage

```
enum {
    mouseMovedMessage = 0x00FA,
    suspendResumeMessage = 0x0001
};
```

Constants

`mouseMovedMessage`
The message code indicating the mouse-moved operating-system event.

Available in Mac OS X v10.0 and later.

Declared in `Events.h`.

`suspendResumeMessage`
The message code indicating a suspend or resume operating-system event.

Available in Mac OS X v10.0 and later.

Declared in `Events.h`.

msgWasPartiallyAccepted

```
enum {
    msgWasPartiallyAccepted = 2,
    msgWasFullyAccepted = 1,
    msgWasNotAccepted = 0
};
```

Constants

`msgWasPartiallyAccepted`
The posting enumerator value in the return receipt that indicates the message was partially accepted.

`msgWasFullyAccepted`
The posting enumerator value in the return receipt that indicates the message was fully accepted.

`msgWasNotAccepted`
The posting enumerator value in the return receipt that indicates the message was fully accepted.

networkEvt

```
enum {
    networkEvt = 10,
    driverEvt = 11,
    app1Evt = 12,
    app2Evt = 13,
    app3Evt = 14,
    app4Evt = 15,
    networkMask = 0x0400,
    driverMask = 0x0800,
    app1Mask = 0x1000,
    app2Mask = 0x2000,
    app3Mask = 0x4000,
    app4Mask = 0x8000
};
```

Constants

`networkEvt`
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`driverEvt`
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`app1Evt`
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`app2Evt`
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`app3Evt`
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`app4Evt`
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`networkMask`
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`driverMask`
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`app1Mask`
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`app2Mask`
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`app3Mask`
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`app4Mask`
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

Event Kind Constants

Define event kinds.

```
enum {
    nullEvent = 0,
    mouseDown = 1,
    mouseUp = 2,
    keyDown = 3,
    keyUp = 4,
    autoKey = 5,
    updateEvt = 6,
    diskEvt = 7,
    activateEvt = 8,
    osEvt = 15,
    kHighLevelEvent = 23
};
typedef UInt16 EventKind;
```

Constants

`nullEvent`
The event code indicating that there are no other pending events.
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`mouseDown`
The event code indicating that the mouse button has been pressed.
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`mouseUp`
The event code indicating that the mouse button has been released.
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`keyDown`
The event code indicating that a key has been pressed.
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`keyUp`
The event code indicating that a key has been released.
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`autoKey`
The event code indicating that a key has been repeatedly held down.
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`updateEvt`
The event code indicating that a window needs updating.
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`diskEvt`
The event code indicating that a disk has been inserted.
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`activateEvt`
The event code indicating that a window has been activated or deactivated.
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`osEvt`
The event code indicating a suspend, resume, or mouse-moved operating-system event.
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

`kHighLevelEvent`
A high-level event.
Available in Mac OS X v10.0 and later.
Declared in `Events.h`.

resumeFlag

Indicates a resume event.

```
enum {
    resumeFlag = 1
};
```

Constants

`resumeFlag`

Flag for a resume event.

Available in Mac OS X v10.0 and later.

Declared in `Events.h`.

Result Codes

Result codes defined for the Event Manager are listed below.

Result Code	Value	Description
`evtNotEnb`	1	Event not enabled for `PostEvent`. Available in Mac OS X v10.0 and later.

Ink Services Reference

Framework: Carbon/Carbon.h

Declared in Ink.h

Overview

Ink is a technology that allows users to enter text by writing with a stylus on a graphics tablet without requiring any modifications to the application that receives the text. As text is written on a tablet, it is automatically recognized and entered as a stream of key-down events into a document or text field.

The Ink Services application programming interface provides a set of functions that enables you to customize Ink input for your application. Using the Ink Services API, you can:

- Programmatically turn handwriting recognition on or off for your application
- Access Ink data at multiple levels (as points and recognized text)
- Support gestures that allow the user to manipulate text directly
- Set up deferred recognition or use on-demand recognition
- Access alternate text interpretations
- Manage options for drawing Ink
- Create and terminate an Ink phrase

Ink Services provides Ink input data (text interpretations, gestures, and so forth) through the Carbon Event Manager. Your application must set up one or more handlers to receive Ink-related events and to extract the relevant parameters from the events of interest to your application.

Functions by Task

Customizing Ink Services

`InkSetApplicationWritingMode` (page 1029)
Controls where the user is allowed to write in the current application.

`InkSetApplicationRecognitionMode` (page 1028)
Specifies whether Ink input should be interpreted as text, gestures, both, or neither.

`InkSetPhraseTerminationMode` (page 1030)
Sets the conditions that define a phrase termination.

`InkSetDrawingMode` (page 1030)
Controls what is drawn when the user writes.

Obtaining Information About Ink Services

`InkUserWritingMode` (page 1042)
Returns the Ink writing mode set by the user in the Ink preferences pane.

`InkIsPhraseInProgress` (page 1028)
Returns whether Ink Services has initiated and is currently maintaining an Ink phrase whose source is user input.

Handling Ink Phrases

`InkAddStrokeToCurrentPhrase` (page 1027)
Adds a stroke to the current Ink phrase.

`InkTerminateCurrentPhrase` (page 1033)
Terminates the current phrase.

Working With Alternate Text Interpretations

`InkTextAlternatesCount` (page 1034)
Returns the number of alternate text interpretations available for an Ink phrase.

`InkTextCreateCFString` (page 1035)
Obtains the string associated with a text interpretation of an Ink phrase.

`InkTextInsertAlternatesInMenu` (page 1040)
Inserts a list of alternate text interpretations into a menu.

Working With Ink Text Objects

`InkTextKeyModifiers` (page 1042)
Returns a value that specifies the key modifiers applied to an Ink phrase.

`InkTextCopy` (page 1035)
Copies an existing Ink text object.

`InkTextBounds` (page 1034)
Returns the bounds of an Ink text object.

`InkTextDraw` (page 1036)
Rescales and draws Ink text into the specified bounds.

`InkTextGetTypeID` (page 1040)
Returns the `CFTypeID` of an `InkTextRef` object.

Flattening and Unflattening Ink Text Objects

`InkTextFlatten` (page 1037)
Flattens an Ink text object for archiving.

`InkTextCreateFromCFData` (page 1036)
Creates an Ink text object from a previously-flattened Ink text object.

Working with Ink Stroke Objects

`InkTextGetStroke` (page 1038)
Returns a reference to the specified stroke in an `InkTextRef`.

`InkTextGetStrokeCount` (page 1039)
Returns the number of strokes in the specified `InkTextRef`.

`InkStrokeGetPointCount` (page 1031)
Returns the number of points in the specified `InkStrokeRef`.

`InkStrokeGetPoints` (page 1032)
Fills an array with the points belonging to the specified `InkStrokeRef`.

`InkStrokeGetTypeID` (page 1033)
Returns the `CFTypeID` of an `InkStrokeRef` object.

Functions

InkAddStrokeToCurrentPhrase

Adds a stroke to the current Ink phrase.

```
void InkAddStrokeToCurrentPhrase (
   unsigned long iPointCount,
   InkPoint *iPointArray
);
```

Parameters

iPointCount
The number of elements in the `iPointArray` array.

iPointArray
A pointer to an array of `InkPoint` structures that specify the path of the stylus, starting with the point that defines the first stylus-down location and ending with the point that defines the last stylus-down location.

Discussion
This function operates on the Ink source from the application, and not on that from direct user input. So there is no need to specify the Ink source as `kInkSourceApplication`. See "Ink Source Types" (page 1052) for more information on sources.

You do not need to call this function unless you have raw data to process or you have turned off automatic recognition (by calling the function `InkSetApplicationWritingMode` (page 1029)) and have set up your application to handle Ink input events itself. For example, you might need to handle Ink input if your application needs to acquire pen data in a device-specific manner.

If your application handles Ink input events, it can still take advantage of the recognition service provided by Ink Services. To do so, your application should call the function `InkAddStrokeToCurrentPhase` to add one stroke at a time to the current phrase. You then terminate the phrase at the appropriate time by calling the function `InkTerminateCurrentPhrase` (page 1033). Note that calling `InkAddStrokeToCurrentPhase` adds a stroke to the current phrase, but does not draw the stroke. See *Using Ink Services in Your Application* for details on writing code that uses the function `InkAddStrokeToCurrentPhase` to implement deferred recognition.

Availability
Not available in CarbonLib 1.x.
Available in Mac OS X v10.3 and later.

Declared In
`Ink.h`

InkIsPhraseInProgress

Returns whether Ink Services has initiated and is currently maintaining an Ink phrase whose source is user input.

```
Boolean InkIsPhraseInProgress (
   void
);
```

Return Value
Returns `TRUE` if the user is currently engaged in Ink input; `FALSE` otherwise.

Discussion
If your application manages its own phrase termination, you should use this function to make sure there is a phrase that can be terminated before you call the function `InkTerminateCurrentPhrase`. Don't call this function if the Ink data stream originates from your application rather than directly from user input. The application data stream is completely independent of the user data stream. If your application builds its own Ink phrases by calling the function `InkAddStrokeToCurrentPhrase`, it should be able track whether such a phrase is in-progress or not.

Availability
Not available in CarbonLib 1.x.
Available in Mac OS X v10.3 and later.

Declared In
`Ink.h`

InkSetApplicationRecognitionMode

Specifies whether Ink input should be interpreted as text, gestures, both, or neither.

```
void InkSetApplicationRecognitionMode (
   InkRecognitionType iRecognitionType
);
```

Parameters

iRecognitionType

The recognition mode you want Ink Services to use. Pass `kInkRecognitionGesture` to specify gesture recognition, `kInkRecognitionText` to specify text recognition, `kInkRecognitionNone` to turn off recognition, or `kInkRecognitionDefault` (which is `kInkRecognitionGesture | kInkRecognitionText`) to specify both gesture and text recognition. See "Recognition Modes" (page 1048) for more information on the constants you can supply.

Discussion

This function only affects recognition of Ink that originates from the user. It does not affect recognition of Ink that originates from your application, and is recognized using the function `InkAddStrokeToCurrentPhrase`. Note that only text recognition (not gesture recognition) is performed on an Ink data stream that originates from your application.

Availability

Not available in CarbonLib 1.x.

Available in Mac OS X v10.3 and later.

Declared In

`Ink.h`

InkSetApplicationWritingMode

Controls where the user is allowed to write in the current application.

```
void InkSetApplicationWritingMode (
   InkApplicationWritingModeType iWriteWhere
);
```

Parameters

iWriteWhere

An "Application Modes" (page 1046) constant that specifies the Ink writing mode to use for your application. Pass `kInkWriteAnywhereInApp` if you want your application to allow Ink input and recognition and to receive Ink events when the user writing mode is set to `kInkWriteInkAwareOnly`. When you call this function with the `iWriteWhere` parameter set to `kInkWriteAnywhereInApp`, your application can receive Ink events whose screen locations lie outside the application windows. Pass `kInkWriteNowhereInApp` to disable Ink input temporarily, such as when the user is using a paint tool.

Discussion

You can call the function `InkSetApplicationWritingMode` to control when Ink input and recognition are allowed in your application. Using this function, you can turn Ink Services on or off for your application. Note that Ink input is available for your application only when your application is frontmost and when the user has turned on recognition in the Ink preferences pane.

If your application calls the function `InkSetApplicationWritingMode` with the parameter `kInkWriteNowhereInApp` to disable Ink Services management of pen events because you want to accumulate Ink data yourself, be aware that you may need to manage mouse event coalescing yourself. You can use the Carbon Event Manger function `SetMouseCoalescingEnabled` for this purpose. See *Using Ink Services in Your Application* for a discussion of mouse coalescing.

Availability
Not available in CarbonLib 1.x.
Available in Mac OS X v10.3 and later.

Declared In
`Ink.h`

InkSetDrawingMode

Controls what is drawn when the user writes.

```
void InkSetDrawingMode (
   InkDrawingModeType iDrawingMode
);
```

Parameters

`iDrawingMode`

A "Drawing Modes" (page 1046) constant that specifies the drawing mode to use for your application. The default (`kInkDrawInkAndWritingGuides`) is for Ink Services to draw both the Ink writing guides and the Ink. Pass `kInkDrawInkOnly` if you want Ink Services to draw only the Ink. Pass `kInkDrawNothing` to turn off drawing of both the Ink writing guides and the Ink.

Discussion
Normally Ink Services draws writing guides, similar in look to the alternating solid and broken lines used on many paper writing tablets. The Ink itself is drawn anti-aliased and grayscale. Your application can call the function `InkSetDrawingMode` to request that Ink Services not draw the writing guides or not draw either Ink or the writing guides. If Ink drawing is disabled, your application must receive the points (by installing a handler for `kEventInkPoint` events) and draw the Ink.

You do not need to call the function `InkSetDrawingMode` to inhibit drawing if you called the function `InkSetApplicationWritingMode`, passing the value `kInkWriteNowhereInApp`. Also, Ink Services will not draw any point for which a `kEventInkPoint` Carbon event handler returns `noErr`.

Availability
Not available in CarbonLib 1.x.
Available in Mac OS X v10.3 and later.

Declared In
`Ink.h`

InkSetPhraseTerminationMode

Sets the conditions that define a phrase termination.

```
void InkSetPhraseTerminationMode (
   InkSourceType iSource,
   InkTerminationType iAllowedTerminationTypes
);
```

Parameters

iSource

An "Ink Source Types" (page 1052) constant that specifies the source of the Ink data stream. You can use one of these constants to get independent control over termination of data originating with the user versus data that is passed from your application to Ink Services. To manage phrase termination for user input, pass the constant `kInkSourceUser`. To manage phrase termination for application input (that is recognized using the function `InkAddStrokeToCurrentPhrase`, pass the constant `kInkSourceApplication`.

iAllowedTerminationTypes

A constant that specifies the conditions which define a phrase termination. To turn off automatic phrase termination, pass `kInkTerminationNone`. You can restore the default phrase termination behavior by passing the constant `kInkTerminationDefault`. See "Phrase Termination Modes" (page 1047) for more information on the constants you can supply.

Discussion

The default behavior is for Ink Services to terminate a phrase when one of the following events occur:

- The user removes the stylus from the proximity of the tablet
- A specified period of time elapses in which the stylus is not pressed to the tablet (The user can control the period of time in the Ink preferences pane.)
- The user writes sufficiently far away from the previous Ink—either horizontally, or on a new line

You can use the function `InkSetPhraseTerminationMode` if your application does not want the default behavior or wants complete control over when Ink phrases are terminated. If you turn off automatic phrase termination, you must make sure you manage phrase termination appropriately for your application.

For example, if you want to force Ink drawn in a specific input window to be treated as a single phrase until the user presses a "finished-writing" button, you would call `InkSetPhraseTerminationMode` with the parameter `kInkTerminationNone` to turn off automatic phrase termination. Then you would need to install a Carbon event handler for the event `kEventInkPoint`. Your handler would examine the `kEventInkPoint` events, notice when a pen-down event occurs on the "finished-writing" button, and then terminate the phrase by calling the function `InkTerminateCurrentPhrase`. See *Using Ink Services in Your Application* for details on writing code to handle phrase termination.

Availability

Not available in CarbonLib 1.x.

Available in Mac OS X v10.3 and later.

Declared In

`Ink.h`

InkStrokeGetPointCount

Returns the number of points in the specified `InkStrokeRef`.

```
CFIndex InkStrokeGetPointCount (
   InkStrokeRef iStrokeRef
);
```

Parameters

iStrokeRef

The `InkStrokeRef` to get the point count from.

Return Value

A `CFIndex` indicating the number of points contained in the specified `InkStrokeRef`.

Discussion

Given an `InkStrokeRef`, this function returns the number of points that stroke contains. Use this function to calculate the appropriate size of the buffer passed to `InkStrokeGetPoints` (page 1032).

Availability

Not available in CarbonLib 1.x.

Available in Mac OS X v10.4 and later.

Declared In

`Ink.h`

InkStrokeGetPoints

Fills an array with the points belonging to the specified `InkStrokeRef`.

```
InkPoint * InkStrokeGetPoints (
   InkStrokeRef iStrokeRef,
   InkPoint *oPointBuffer
);
```

Parameters

iStrokeRef

The `InkStrokeRef` to get the points from.

oPointBuffer

The buffer into which the point data is to be copied.

Return Value

A pointer to the copied array of point data from the specified `InkStrokeRef`; this value is the same as the `oPointBuffer` address provided by the application.

Discussion

Given an `InkStrokeRef` and a point buffer, this function fills that buffer with the points belonging to that stroke.

The size of the point buffer must be at least the size of `InkStrokeGetPointCount( iStrokeRef ) * sizeof( InkPoint )`. For details, see `InkStrokeGetPointCount` (page 1031). The pointer to the block of memory containing the ink points is returned as the result.

Availability

Not available in CarbonLib 1.x.

Available in Mac OS X v10.4 and later.

Declared In

`Ink.h`

InkStrokeGetTypeID

Returns the `CFTypeID` of an `InkStrokeRef` object.

```
CFTypeID InkStrokeGetTypeID (
   void
);
```

Return Value
The `CFTypeID` of an `InkStrokeRef` object.

Discussion
Given an `InkStrokeRef`, this function returns its `CFTypeID`.

Availability
Not available in CarbonLib 1.x.
Available in Mac OS X v10.4 and later.

Declared In
`Ink.h`

InkTerminateCurrentPhrase

Terminates the current phrase.

```
void InkTerminateCurrentPhrase (
   InkSourceType iSource
);
```

Parameters

iSource

An "Ink Source Types" (page 1052) constant that specifies the source of the Ink data stream.To terminate a phrase that originates from application input (that is recognized using the function `InkAddStrokeToCurrentPhrase)`, pass the constant `kInkSourceApplication`.

If you are managing phrase termination that originates from direct user input, you can pass the constant `kInkSourceUser`. Note that this function is normally not used in this fashion, as most applications can let Ink Services terminate such phrases automatically.

Discussion
You do not need to call this function unless you have turned off automatic phrase termination (by calling the function `InkSetPhraseTerminationMode` (page 1030)) and have set up your application to manage phrase termination. When you call the function `InkTerminateCurrentPhrase`, any Ink drawn by Ink Services is erased. If your application handles phrase termination, it can still take advantage of the recognition service provided by Ink Services.

Availability
Not available in CarbonLib 1.x.
Available in Mac OS X v10.3 and later.

Declared In
`Ink.h`

InkTextAlternatesCount

Returns the number of alternate text interpretations available for an Ink phrase.

```
CFIndex InkTextAlternatesCount (
   InkTextRef iTextRef
);
```

Parameters

iTextRef

On input, a reference to the Ink text object that specifies the Ink word whose alternate count you want to obtain. You must obtain an Ink text object reference (`InkTextRef`) through your application's Ink event handler. Your handler must take care of the Carbon event class `kEventClassInk` and the event kind `kEventInkText`. The event parameter `kEventParamInkTextRef` that you obtain from this event kind is a reference to an Ink text object.

Return Value

Returns the number of interpretations available for the specified Ink phrase.

Discussion

You can obtain the string associated with a text interpretation by calling the function `InkTextCreateCFString` (page 1035). If you want to display a list of the alternate text interpretations to the user, call the function `InkTextInsertAlternatesInMenu` (page 1040).

Availability

Not available in CarbonLib 1.x.

Available in Mac OS X v10.3 and later.

Declared In

`Ink.h`

InkTextBounds

Returns the bounds of an Ink text object.

```
HIRect InkTextBounds (
   InkTextRef iTextRef
);
```

Parameters

iTextRef

On input, a reference to the Ink text object whose bounds you want to obtain. You must obtain an Ink text object reference (`InkTextRef`) through your application's Ink event handler. Your handler must take care of the Carbon event class `kEventClassInk` and the event kind `kEventInkText`. The event parameter `kEventParameterInkTextRef` that you obtain from this event kind is a reference to an Ink text object.

Return Value

An `HIRect` data structure that defines the bounds of the specified Ink text object.

Discussion

The bounds are initially global coordinates, and may extend beyond your application's windows.

Availability

Not available in CarbonLib 1.x.

Available in Mac OS X v10.3 and later.

Declared In
`Ink.h`

InkTextCopy

Copies an existing Ink text object.

```
InkTextRef InkTextCopy (
   InkTextRef iTextRef
);
```

Parameters

iTextRef

On input, a reference to the Ink text object you want to copy. You must obtain an Ink text object reference (`InkTextRef`) through your application's Ink event handler. Your handler must take care of the Carbon event class `kEventClassInk` and the event kind `kEventInkText`. The event parameter `kEventParameterInkTextRef` that you obtain from this event kind is a reference to an Ink text object.

Return Value
A reference to the newly-created Ink text object.

Discussion
You can use this function to implement copy-and-paste functions in a deferred-recognition application, or in a text application to retain Ink when text is copied and pasted. The retention count of the new `InkTextRef` is 1.

Availability
Not available in CarbonLib 1.x.
Available in Mac OS X v10.3 and later.

Declared In
`Ink.h`

InkTextCreateCFString

Obtains the string associated with a text interpretation of an Ink phrase.

```
CFStringRef InkTextCreateCFString (
   InkTextRef iTextRef,
   CFIndex iAlternateIndex
);
```

Parameters

iTextRef

On input, a reference to the Ink text object that specifies the Ink word for which you want to create a string. You must obtain an Ink text object reference (`InkTextRef`) through your application's Ink event handler. Your handler must take care of the Carbon event class `kEventClassInk` and the event kind `kEventInkText`. The event parameter `kEventParamInkTextRef` that you obtain from this event kind is a reference to an Ink text object.

iIndex

The index that specfies the text interpretation for which you want to obtain a `CFString`. Text interpretations are stored in an array in ranked order, with the most-likely interpretation at index zero.

Return Value
A `CFStringRef` that specifies an interpretation for the given Ink text phrase. Returns `NULL` if the index you provide is invalid. Your application is responsible for releasing the returned `CFStringRef`.

Availability
Not available in CarbonLib 1.x.
Available in Mac OS X v10.3 and later.

Declared In
`Ink.h`

InkTextCreateFromCFData

Creates an Ink text object from a previously-flattened Ink text object.

```
InkTextRef InkTextCreateFromCFData (
   CFDataRef iFlattenedInkText,
   CFIndex iIndex
);
```

Parameters

`iFlattenedInkText`
On input, a reference to a `CFData` data structure that contains data from a previously-flattened Ink text object.

`iIndex`
The index at which to start reading the data.

Return Value
Returns a reference to the newly-created Ink text object. The retention count of the newly-created Ink text object is 1.

Discussion
You can unflatten an Ink text object that was previously flattened using the function `InkTextFlatten`. If you flattened more than one Ink text object to the `CFMutableData` data type, then you must call the function `InkTextCreateFromCFData` for each Ink text object you want to unflatten, specifying the index that defines the start of the Ink text data you want to unflatten.

Availability
Not available in CarbonLib 1.x.
Available in Mac OS X v10.3 and later.

Declared In
`Ink.h`

InkTextDraw

Rescales and draws Ink text into the specified bounds.

```
void InkTextDraw (
   InkTextRef iTextRef,
   CGContextRef iContext,
   const CGRect *iBounds,
   InkTextDrawFlagsType iFlags
);
```

Parameters

iTextRef

On input, a reference to the Ink text object whose text you want to rescale and draw. You must obtain an Ink text object reference (`InkTextRef`) through your application's Ink event handler. Your handler must take care of the Carbon event class `kEventClassInk` and the event kind `kEventInkText`. The event parameter `kEventParameterInkTextRef` that you obtain from this event kind is a reference to an Ink text object.

iContext

The `CGContext` into which you want to draw. Drawing is relative to the specified `CGContextRef`, and subject to the usual window and clipping constraints. Pass `NULL` if you want to draw to the canonical context of the current port.

iBounds

On input, a `CGRect` data structure that specifies the bounds into which you want the Ink text object to be drawn.

iFlags

A "Text Drawing Flags" (page 1052) constant that specifies drawing settings. Pass `kInkTextDrawDefault` to use the default system settings when drawing, `kInkTextDrawIgnorePressure` if you do not want to use pressure sensitive gradients, and `kInkTextDrawHonorContext` t o use the current Quartz context settings.

Discussion

The function `InkTextDraw` is useful to applications that implement deferred recognition or searchable Ink. The original points and bounds of the Ink text object are scaled and offset to fit the specified bounds, so subsequent calls to the function `InkTextBounds` return the rescaled bounds.

Availability

Not available in CarbonLib 1.x.

Available in Mac OS X v10.3 and later.

Declared In

`Ink.h`

InkTextFlatten

Flattens an Ink text object for archiving.

```
CFIndex InkTextFlatten (
   InkTextRef iTextRef,
   CFMutableDataRef ioDataRef,
   CFIndex iIndex
);
```

Parameters

iTextRef

On input, a reference to the Ink text object you want to flatten. You must obtain an Ink text object reference (`InkTextRef`) through your application's Ink event handler. Your handler must take care of the Carbon event class `kEventClassInk` and the event kind `kEventInkText`. The event parameter `kEventParameterInkTextRef` that you obtain from this event kind is a reference to an Ink text object.

ioDataRef

On input, a reference to a `CFMutableData` data type. On output, refers to the flattened data. Your application is responsible for creating and releasing the `CFMutableDataRef`.

iIndex

The index at which you want the data to be written.

Return Value

Returns the number of bytes added to the `ioDataRef`. Returns `0` if the operation is unsuccessful or `iTextRef` is `NULL` or empty.

Discussion

`CFMutableData` objects are extensible, which means you can flatten more than one Ink text object into a `CFMutableData` object. You store data in a `CFMutableData` object by specifying the index at which data is to be stored. The function `InkTextFlatten` accepts the starting index as an input parameter, writes all the Ink text object data to the specified `CFMutableData` object, and returns the byte count for the amount of data that is actually stored.

To flatten an additional Ink text object into the same `CFMutableData` object, you must supply a value for the `iIndex` parameter that specifies the byte location at which to start writing the data for the additional Ink text object. You can calculate this value by summing the byte count returned by the previous call with the value of the `iIndex` parameter you provided in the previous call.

Availability

Not available in CarbonLib 1.x.

Available in Mac OS X v10.3 and later.

Declared In

`Ink.h`

InkTextGetStroke

Returns a reference to the specified stroke in an `InkTextRef`.

```
InkStrokeRef InkTextGetStroke (
   InkTextRef iTextRef,
   CFIndex iStrokeIndex
);
```

Parameters

iTextRef

The `InkTextRef` to get the stroke from.

iStrokeIndex

The index of the stroke for which you want to get an `InkStrokeRef`.

Return Value

An `InkStrokeRef` for the specified stroke of the specified `InkTextRef`.

Discussion

Given an `InkTextRef` and a stroke index (between 0 and `InkTextGetStrokeCount( iTextRef ) - 1`), this function returns the `InkStrokeRef` corresponding to the specified stroke index. For details, see `InkTextGetStrokeCount` (page 1039).

The returned `InkStrokeRef` is guaranteed to persist only for the life of the `InkTextRef` from which it was obtained. If you want to use the `InkStrokeRef` after the `InkTextRef` has been released, you must call the function `CFRetain` and pass the `InkStrokeRef` to it.

When any Ink object reference is obtained from a Carbon event, it is guaranteed to persist only for the life of the event handler. If you want to use the object at some later time, you must call the function `CFRetain` and pass the object to it.

Availability

Not available in CarbonLib 1.x.

Available in Mac OS X v10.4 and later.

Declared In

`Ink.h`

InkTextGetStrokeCount

Returns the number of strokes in the specified `InkTextRef`.

```
CFIndex InkTextGetStrokeCount (
   InkTextRef iTextRef
);
```

Parameters

iTextRef

The `InkTextRef` to get the stroke count from.

Return Value

The number of stokes in the specified `InkTextRef`.

Discussion

Given an `InkTextRef`, this function returns the number of strokes the `InkTextRef` contains.

Availability

Not available in CarbonLib 1.x.

Available in Mac OS X v10.4 and later.

Declared In
`Ink.h`

InkTextGetTypeID

Returns the `CFTypeID` of an `InkTextRef` object.

```
CFTypeID InkTextGetTypeID (
   void
);
```

Return Value
The `CFTypeID` of an `InkStrokeRef` object.

Discussion
Given an `InkTextRef`, this function returns its `CFTypeID`.

Availability
Not available in CarbonLib 1.x.
Available in Mac OS X v10.4 and later.

Declared In
`Ink.h`

InkTextInsertAlternatesInMenu

Inserts a list of alternate text interpretations into a menu.

```
ItemCount InkTextInsertAlternatesInMenu (
   InkTextRef iTextRef,
   MenuRef iMenuRef,
   MenuItemIndex iAfterItem
);
```

Parameters

`iTextRef`

On input, a reference to an Ink text object that specifies the Ink word for which you want to provide a list of alternate text interpretations. You must obtain an Ink text object reference (`InkTextRef`) through your application's Ink event handler. Your handler must take care of the Carbon event class `kEventClassInk` and the event kind `kEventInkText`. The event parameter `kEventParamInkTextRef` that you obtain from this event kind is a reference to an Ink text object.

`iMenuRef`

A reference to the menu into which you want to insert the list of alternate text interpretations. Ink Services attaches menu event handlers to this menu, so you should use this `MenuRef` directly, rather than copy items from the menu reference to another menu.

`iAfterItem`

A value that specifies the menu item after which you want to insert the list of alternate text interpretations. If the specified menu item is `0`, the text alternates are inserted at the head of the menu. If the specified menu item is greater than or equal to the existing number of menu items, the text alternates are appended to the end of the menu. Remember that the first item in a menu item array is numbered `1`, not `0`.

Return Value
Returns the number of menu items added to the menu. Returns `0` if the operation is not successful.

Discussion
The function `InkTextInsertAlternatesInMenu` allows your application to insert a list of text interpretations for a given Ink text phrase into an existing contextual menu. You should handle a list of alternate text interpretations as a standard contextual menu using the Menu Manager function `ContextualMenuSelect`.

When a user selects an item from the list of alternates, the list of alternates maintained by Ink Services are reordered automatically. This means that if you call the function `InkTextCreateCFString` with the parameter `iIndex` set to `0`, you obtain the newly selected item.

Thus the user's choice persists in internal system data structures without requiring your application to call additional functions. However your application must update its own internal data structures appropriately.

You must rebuild the menu to reflect the user's choice. After the user makes a choice and then reopens the menu, you must make sure the newly-selected item shows up as the first item in the menu. The items in the menu should mirror the list of alternates maintained by Ink Services.

Upon return from the function `ContextualMenuSelect`, your application can determine if the user has made a selection by checking the value of the parameter `outUserSelectionType`. The value indicates the item that the user selected from the contextual menu. If there is a selection, your application can examine the `outMenuID` and `outMenuItem` parameters of the function `ContextualMenuSelect`, and use these values to obtain the alternate text interpretation by calling the Menu Manager function `CopyMenuItemTextAsCFString`.

Menu items for a set of alternates whose first letter is an alphabetical character always include an alternate whose first letter is the opposite lettercase. Menu items for a set of alternates whose first letter is a non-alphabetical character do not include a lettercase alternate.

When the menu items are reordered automatically, the text that was first in the list moves to the second or the third position, depending upon whether the first letter is alphabetical or non-alphabetical. For example, the following list of menu items:

crash, Crash, crush, crust, wrash

If the user chooses crush, the menu items are reordered as follows:

crush, Crush, crash, crust, wrash

Notice that the list of alternates is kept to a length of five. A lettercase alternate for crush is added to the menu while the uppercase alternate Crash is dropped.

For a non-alphabetic first character, however, such as a number, the original moves to the second position. So for the following menu:

1239, 1234, 1289, 1284

If the user chooses 1234, the menu becomes:

1234, 1239, 1289, 1284

If it is important for your application to maintain the original order of alternates, then it must use its own internal data structures to keep track of the original list.

See *Using Ink Services in Your Application* for details on writing code that uses the function `InkTextInsertAlternatesInMenu` to implement a correction model.

Availability
Not available in CarbonLib 1.x.
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In
`Ink.h`

InkTextKeyModifiers

Returns a value that specifies the key modifiers applied to an Ink phrase.

```
UInt32 InkTextKeyModifiers (
   InkTextRef iTextRef
);
```

Parameters

`iTextRef`

On input, a reference to the Ink text object whose key modifiers you want to obtain. You must obtain an Ink text object reference (`InkTextRef`) through your application's Ink event handler. Your handler must take care of the Carbon event class `kEventClassInk` and the event kind `kEventInkText`. The event parameter `kEventParameterInkTextRef` that you obtain from this event kind is a reference to an Ink text object.

Return Value
Returns a value that indicates which modifier keys were down during input of the Ink phrase. This value is in the same form as that used by the Carbon Event Manager for the event parameter `kEventParamKeyModifiers`.

Discussion
Ink Services assigns keyboard modifier keys to a stroke if those keys are held down for more than 50% of the stroke's points. Ink Services assigns the modifier keys associated with a phrase's first stroke to the entire phrase. Ink Services assigns the modifier keys associated with a phrase to all of the text interpretations that are derived from an Ink phrase.

Availability
Not available in CarbonLib 1.x.
Available in Mac OS X v10.3 and later.

Declared In
`Ink.h`

InkUserWritingMode

Returns the Ink writing mode set by the user in the Ink preferences pane.

```
InkUserWritingModeType InkUserWritingMode (
   void
);
```

Return Value
A value that specifies the current user preferences settings for Ink: `kInkWriteInInkAwareAppsOnly`, `kInkWriteAnywhere`, or `kInkWriteNowhere`. In general, Ink services are not available if `kInkWriteNowhere` is returned (indicating the user has turned Ink off entirely). See "User Writing Modes" (page 1045) for more information on each of these constants.

Discussion
User preferences for Ink are set by the user in the Ink pane of System Preferences. Your application can only read these values, not set them.

Availability
Not available in CarbonLib 1.x.
Available in Mac OS X v10.3 and later.

Declared In
`Ink.h`

Data Types

InkTextRef

Defines a data type for a reference to an opaque Ink text object.

```
typedef     struct OpaqueInkTextRef * InkTextRef;
```

Discussion
You must use the Core Foundation functions `CFRetain` and `CFRelease` to manage the retention and release of Ink text objects. When an Ink text reference is obtained from a Carbon event, it is guaranteed to persist only for the life of the event handler. If your application needs to use the Ink text object at some later time, you must call the function `CFRetain`, passing the object you want to retain as a parameter.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`Ink.h`

InkStrokeRef

Defines a data type for a reference to an opaque Ink stroke object.

```
typedef     struct OpaqueInkStrokeRef * InkStrokeRef;
```

Discussion
You must use the Core Foundation functions `CFRetain` and `CFRelease` to manage the retention and release of Ink stroke objects. When an Ink stroke reference is obtained from a Carbon event, it is guaranteed to persist only for the life of the event handler. If your application needs to use the Ink stroke object at some later time, you must call the function `CFRetain`, passing the object you want to retain as a parameter.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`Ink.h`

InkAlternateCount

Defines a data type that specifies the number of alternate text interpretations of an Ink phrase.

```
typedef unsigned long     InkAlternateCount;
```

Discussion
Values of `type InkAlternateCount` are returned by the function `InkTextAlternatesCount` (page 1034) and passed as a parameter to the function `InkTextCreateCFString` (page 1035).

Availability
Available in Mac OS X v10.3 and later.

Declared In
`Ink.h`

InkPoint

Contains data that describes an Ink point.

```
struct InkPoint {
    HIPoint             point;
    TabletPointRec      tabletPointData;
    UInt32              keyModifiers;
};
typedef struct InkPoint      InkPoint;
typedef InkPoint *      InkPointPtr;
```

Fields

`point`

Defines a point in floating-point coordinates. These values are generally in global coordinates, with full sub-pixel accuracy. This coordinate is what you obtain for a mouse event from the Carbon event parameter `kEventParamMouseLocation`, which also contains a `typeHIPoint` value.

`tabletPointData`

A tablet point structure that contains pressure, tilt, rotation, and coordinate (in tablet space) data for a pen. The pressure value is a measure of how hard the pen is being pressed, ranging from 0 to 65535. Some tablet manufacturers allow users to adjust pen sensitivity. In these cases, the zero value always corresponds to the threshold set by the user, and the pressure value is relative to that threshold. See *Carbon Event Manager Reference* for information on the `TabletPointRec` data type.

`keyModifiers`
A value that specifies the keyboard modifier key that is pressed when the point is sampled. This value is in the same form as that used by the Carbon Event Manager for the event parameter `kEventParamKeyModifiers`.

Discussion
An `InkPoint` data structure contains an essentially complete set of per-point data. Ink Services currently only requires the point's (x,y) coordinates and pressure to perform recognition and to draw the ink, but future recognition services may require other information from the `TabletPointRec`.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`Ink.h`

Constants

User Writing Modes

Specify the Ink writing mode set by the user in the Ink pane of System Preferences.

```
enum {
    kInkWriteNowhere     = 'nowh',
    kInkWriteAnywhere = 'anyw',
    kInkWriteInInkAwareAppsOnly     = 'iapp'
};
typedef FourCharCode InkUserWritingModeType;
```

Constants
`kInkWriteNowhere`
Specifies the user has disabled Ink or that Ink Services are not available (for example, a tablet is not attached).

Available in Mac OS X v10.3 and later.

Declared in `Ink.h`.

`kInkWriteAnywhere`
Specifies the user has enabled Ink to allow writing anywhere on the screen. Ink Services flows ink points and recognition results to the frontmost application. This is the default situation when the user enables Ink.

Available in Mac OS X v10.3 and later.

Declared in `Ink.h`.

`kInkWriteInInkAwareAppsOnly`
Specifies the user has enabled Ink only to allow writing in an application that has enabled Ink Services by calling the function `InkSetApplicationWritingMode` with the `kInkWriteAnywhereInApp` parameter.

Available in Mac OS X v10.3 and later.

Declared in `Ink.h`.

Discussion
These constants are returned by the function `InkUserWritingMode` (page 1042).

Application Modes

Specify an Ink input mode to use for an application.

```
enum {
    kInkWriteNowhereInApp     = 'nowa',
    kInkWriteAnywhereInApp     = 'anya'
};
typedef FourCharCode     InkApplicationModeType;
```

Constants

`kInkWriteNowhereInApp`

Specifies not to allow Ink input in your application.

Available in Mac OS X v10.3 and later.

Declared in `Ink.h`.

`kInkWriteAnywhereInApp`

Specifies to allow Ink input anywhere onscreen for your application.

Available in Mac OS X v10.3 and later.

Declared in `Ink.h`.

Discussion

You can supply these constants as parameters to the function `InkSetApplicationWritingMode` (page 1029). If the user has not enabled Ink or if there is not an Ink input device available, then calling `InkSetApplicationWritingMode` (page 1029) with the parameter `kInkWriteAnywhereInApp` has no effect.

Drawing Modes

Specify what Ink Services should draw.

```
enum {
    kInkDrawNothing = 0,
    kInkDrawInkOnly = 1,
    kInkDrawInkAndWritingGuides= 2
};
typedef unsigned long InkDrawingModeType;
```

Constants

`kInkDrawNothing`

Specifies not to draw Ink or the writing guides.

Available in Mac OS X v10.3 and later.

Declared in `Ink.h`.

`kInkDrawInkOnly`

Specifies to draw Ink but not the writing guides.

Available in Mac OS X v10.3 and later.

Declared in `Ink.h`.

`kInkDrawInkAndWritingGuides`

Specifies to draw both the Ink and the writing guides. This is the default.

Available in Mac OS X v10.3 and later.

Declared in `Ink.h`.

Discussion
You can pass these constants as parameters to the function `InkSetDrawingMode` (page 1030).

Phrase Termination Modes

Defines the conditions under which an Ink phrase should be terminated.

```
enum InkTerminationType{
    kInkTerminationNone = 0,
    kInkTerminationTimeOut = 1,
    kInkTerminationOutOfProximity = 1 << 1,
    kInkTerminationRecognizerHorizontalBreak = 1 << 2,
    kInkTerminationRecognizerVerticalBreak = 1 << 3,
    kInkTerminationStroke = 1 << 4,
    kInkTerminationAll = (unsigned long) 0xFFFFFFFF,
    kInkTerminationDefault = 0x0F
};
typedef unsigned long         InkTerminationType;
```

Constants

`kInkTerminationNone`
Specifies to inhibit automatic phrase termination by Ink Services.

Available in Mac OS X v10.3 and later.

Declared in `Ink.h`.

`kInkTerminationTimeOut`
Specifies to terminate a phrase when all of the following are true:

- The user stops writing and lifts the stylus
- The user keeps the stylus within the proximity range of the tablet
- The user does not resume writing within the period of time defined by the user in the Ink pane of System Preferences

Available in Mac OS X v10.3 and later.

Declared in `Ink.h`.

`kInkTerminationOutOfProximity`
Specifies to terminate a phrase when the user stops writing and lifts the stylus entirely out of the proximity range of the tablet. This is on by default. However, users can turn off proximity termination in the Ink pane of System Preferences if they find it interferes with their writing style.

If the user turns off proximity termination, your application can't turn it on even if you call the function `InkSetPhraseTerminationMode` (page 1030) with the parameter `kInkTerminationOutOfProximity`.

Available in Mac OS X v10.3 and later.

Declared in `Ink.h`.

`kInkTerminationRecognizerHorizontalBreak`
Specifies to terminate a phrase when the user leaves a large horizontal space between words (approximately two character widths or more).

Available in Mac OS X v10.3 and later.

Declared in `Ink.h`.

`kInkTerminationRecognizerVerticalBreak`
Specifies to terminate a phrase when the user finishes one line and begins writing on the next.

Available in Mac OS X v10.3 and later.

Declared in `Ink.h`.

`kInkTerminationStroke`
Causes phrases to be terminated at the end of every stroke (whenever the pen is lifted from the tablet while writing). Only useful for single-stroke gesture input, not for text.

Available in Mac OS X v10.4 and later.

Declared in `Ink.h`.

`kInkTerminationAll`
Specifies to restore automatic phrase termination by Ink Services. In this case, Ink Services uses all of the termination modes (except `kInkTerminationNone`) described previously. Deprecated in Mac OS X v10.4. As of Mac OS X v10.4, this value is overridden to behave like `kInkTerminationDefault`.

Available in Mac OS X v10.3 and later.

Declared in `Ink.h`.

`kInkTerminationDefault`
Restores default phrase termination matching the current user settings (`kInkTerminationTimeOut | kInkTerminationOutOfProximity | kInkTerminationRecognizerHorizontalBreak | kInkTerminationRecognizerVerticalBreak`). See also `kInkTerminationOutOfProximity`.

Declared in `Ink.h`.

Available in Mac OS X v10.4 and later.

Discussion
An Ink phrase (represented as an `InkTextRef` in your application) is typically a word in a Roman language. Ink Services uses phrases to determine when to erase onscreen Ink and initiate recognition. You can pass Ink phrase termination constants as arguments to the function `InkSetPhraseTerminationMode` (page 1030). You can combine two or more constants to obtain precise control over phrase termination.

Recognition Modes

Specify how to interpret Ink input for an application.

```
enum InkRecognitionType{
    kInkRecognitionNone     = 0,
    kInkRecognitionText     = 1,
    kInkRecognitionGesture     = 1 << 1,
    kInkRecognitionDefault     = 3
};
typedef unsigned long     InkRecognitionType;
```

Constants
`kInkRecognitionNone`
Specifies to turn off Ink recognition.

Available in Mac OS X v10.3 and later.

Declared in `Ink.h`.

`kInkRecognitionText`
Specifies to allow interpretation of Ink input as text.

Available in Mac OS X v10.3 and later.

Declared in `Ink.h`.

`kInkRecognitionGesture`
Specifies to allow interpretation of Ink input as gestures.

Available in Mac OS X v10.3 and later.

Declared in `Ink.h`.

`kInkRecognitionDefault`
Specifies the default setting, which is to interpret Ink input as text or gestures.

Available in Mac OS X v10.3 and later.

Declared in `Ink.h`.

Discussion
The recognition type constants are used as arguments for the function `InkSetApplicationRecognitionMode` (page 1028). You can use these constants to specify that Ink Services interprets input as both text and gestures or as either type individually.

Editing Gestures

Define editing actions.

```
enum InkGestureKind {
    kInkGestureUndo     = 'undo',
    kInkGestureCut     = 'cut ',
    kInkGestureCopy     = 'copy',
    kInkGesturePaste     = 'past',
    kInkGestureClear     = 'cler',
    kInkGestureSelectAll     = 'sall',
    kInkGestureLeftSpace     = 'lspc',
    kInkGestureRightSpace     = 'rspc',
    kInkGestureTab     = 'tab ',
    kInkGestureLeftReturn     = 'lrtn',
    kInkGestureRightReturn     = 'rrtn',
    kInkGestureDelete     = 'del ',
    kInkGestureEscape     = 'esc ',
    kInkGestureJoin     = 'join'
    };
typedef FourCharCode     InkGestureKind;
```

Constants
`kInkGestureUndo`
Specifies to undo the last action.

Available in Mac OS X v10.3 and later.

Declared in `Ink.h`.

`kInkGestureCut`
Specifies to cut.

Available in Mac OS X v10.3 and later.

Declared in `Ink.h`.

`kInkGestureCopy`
Specifies to copy.

Available in Mac OS X v10.3 and later.

Declared in `Ink.h`.

`kInkGesturePaste`
Specifies to paste.

Available in Mac OS X v10.3 and later.

Declared in `Ink.h`.

`kInkGestureClear`
Specifies to clear.

Available in Mac OS X v10.3 and later.

Declared in `Ink.h`.

`kInkGestureSelectAll`
Specifies to select all items in the area that has user focus.

Available in Mac OS X v10.3 and later.

Declared in `Ink.h`.

`kInkGestureLeftSpace`
Specifies to insert a single space character. The “left” distinction indicates that the gesture is drawn with the long, horizontal tail is on the left side.

Available in Mac OS X v10.3 and later.

Declared in `Ink.h`.

`kInkGestureRightSpace`
Specifies to insert a single space character. The “right” distinction indicates that the gesture is drawn with the long, horizontal tail is on the right side.

Available in Mac OS X v10.3 and later.

Declared in `Ink.h`.

`kInkGestureTab`
Specifies to insert a tab character.

Available in Mac OS X v10.3 and later.

Declared in `Ink.h`.

`kInkGestureLeftReturn`
Specifies to insert a return (new line) character. The “left” distinction indicates that the gesture is drawn with the small angle-bracket pointing to the left side.

Available in Mac OS X v10.3 and later.

Declared in `Ink.h`.

`kInkGestureRightReturn`
Specifies to insert a return (new line) character. The “right” distinction indicates that the gesture is drawn with the small angle-bracket pointing to the right side.

Available in Mac OS X v10.3 and later.

Declared in `Ink.h`.

`kInkGestureDelete`
Specifies to delete. This corresponds to pressing the Delete key.

Available in Mac OS X v10.3 and later.

Declared in `Ink.h`.

`kInkGestureEscape`
This corresponds to pressing the Escape key.

Available in Mac OS X v10.3 and later.

Declared in `Ink.h`.

`kInkGestureJoin`
Specifies to join two words into a single word, eliding the space between them, and may be applied to editable objects other than text. The gesture is similar in shape to the letter "v". The joined words are the ones closest to the top-most points of the gesture. This is a tentative, always targeted, gesture, meaning that the system treats the associated Ink tentatively as a gesture until your application either confirms the Ink is indeed a gesture or returns `eventNotHandledErr`, informing the system the Ink is not a gesture.

Available in Mac OS X v10.3 and later.

Declared in `Ink.h`.

Discussion
These constants are returned in the Carbon event parameter `kEventParamInkGestureKind`. The Carbon event class for this parameter is `kEventClassInk` and the event kind is `kEventInkGesture`. The constants define the complete set of gestures recognized by Ink Services. When a gesture event is received by your application, your application should determine the gesture kind and then take appropriate action. For more details, see *Using Ink Services in Your Application*.

Alternates Menu Command IDs

Specify the menu command IDs assigned to items inserted in the alternates menu.

```
enum {
    kInkAlternateCommand = 'inka',
    kInkSeparatorCommand         = 'inks',
    kInkDrawingCommand             = 'inkd'
};
```

Constants
`kInkAlternateCommand`
Specifies the menu command ID assigned to menu items inserted by the function `InkTextInsertAlternatesInMenu` (page 1040). You can use this constant to determine which menu items in a menu are supplied by Ink Services.

Available in Mac OS X v10.3 and later.

Declared in `Ink.h`.

`kInkSeparatorCommand`
Specifies the menu command ID assigned to the separator item between the alternates and the Ink drawing.

Available in Mac OS X v10.3 and later.

Declared in `Ink.h`.

`kInkDrawingCommand`
Specifies the menu command ID assigned to the menu item containing the ink drawing.

Available in Mac OS X v10.3 and later.

Declared in `Ink.h`.

Text Drawing Flags

Specify settings to use when drawing Ink text.

```
enum unsigned long InkTextDrawFlagsType{
kInkTextDrawDefault         = 0,
kInkTextDrawIgnorePressure     = 1,
kInkTextDrawHonorContext     = 1 << 1
};
```

Constants

`kInkTextDrawDefault`
Specifies to use the default system settings when drawing. By default, Ink is drawn with pressure sensitive gradients, and the Quartz context settings are overridden for line color and width.

Available in Mac OS X v10.3 and later.

Declared in `Ink.h`.

`kInkTextDrawIgnorePressure`
Specifies not to use pressure sensitive gradients when drawing.

Available in Mac OS X v10.3 and later.

Declared in `Ink.h`.

`kInkTextDrawHonorContext`
Specifies to use the current Quartz context settings for line color and width.

Available in Mac OS X v10.3 and later.

Declared in `Ink.h`.

Ink Source Types

Specify sources for an Ink data stream.

```
enum unsigned long InkSourceType{
    kInkSourceUser                    = 1,
    kInkSourceApplication                        = 2
};
```

Constants

`kInkSourceUser`
Specifies the Ink source from direct user input.

Available in Mac OS X v10.3 and later.

Declared in `Ink.h`.

`kInkSourceApplication`
Specifies the Ink source from the application.

Available in Mac OS X v10.3 and later.

Declared in `Ink.h`.

Discussion
You can use these constants to specify which data stream is currently being controlled by calls to the functions `InkTerminateCurrentPhrase` and `InkSetPhraseTerminationMode`. You can control phrase termination for both the user-input data stream (`kInkSourceUser`) and an application-input data stream (`kInkSourceApplication`) independently.

Ink Pen Constants

Specify ink pen constants.

```
enum {
    kInkPenTipButtonMask = NX_TABLET_BUTTON_PENTIPMASK + 0,
    kInkPenLowerSideButtonMask = NX_TABLET_BUTTON_PENLOWERSIDEMASK
+ 0,
    kInkPenUpperSideButtonMask = NX_TABLET_BUTTON_PENUPPERSIDEMASK
+ 0
};
```

Constants
`kInkPenTipButtonMask`
The writing or eraser tip.

Available in Mac OS X v10.4 and later.

Declared in `Ink.h`.

`kInkPenLowerSideButtonMask`
The lower pen barrel button.

Available in Mac OS X v10.4 and later.

Declared in `Ink.h`.

`kInkPenUpperSideButtonMask`
The upper pen barrel button.

Available in Mac OS X v10.4 and later.

Declared in `Ink.h`.

Discussion
Pens used with modern graphics tablets often have multiple barrel buttons that can be assigned special meaning by the tablet driver or by an application. In addition, the writing or eraser tip may be engaged at any given moment. By performing an AND operation of these contsants with that of a `buttons` member of a `TabletPointRec` in Carbon or the value returned by the `buttonMask` message sent to tablet events (or mouse events containing tablet data) in Cocoa, you can determine which (if any) pen tip or barrel buttons are currently being held down. These buttons and `buttonMask` data are only available for tablet-point events (not tablet-proximity events).

To ensure consistency between the values used by driver writers and the values used by applications, these constants are defined in terms of `NX_` constants from `IOKit/hidsystem/IOLLEvent.h`.

Availability
Available in Mac OS X v10.4 and later.

Ink Tablet Constants

Specify ink tablet constants.

```
enum {
    kInkTabletPointerUnknown = NX_TABLET_POINTER_UNKNOWN + 0,
    kInkTabletPointerPen = NX_TABLET_POINTER_PEN + 0,
    kInkTabletPointerCursor = NX_TABLET_POINTER_CURSOR + 0,
    kInkTabletPointerEraser = NX_TABLET_POINTER_ERASER + 0
};
```

Constants

`kInkTabletPointerUnknown`

The type of tablet pointer is unknown; having an unknown type of tablet pointer should not happen.

Available in Mac OS X v10.4 and later.

Declared in `Ink.h`.

`kInkTabletPointerPen`

The writing end of a stylus-like device.

Available in Mac OS X v10.4 and later.

Declared in `Ink.h`.

`kInkTabletPointerCursor`

Any puck-like device.

Available in Mac OS X v10.4 and later.

Declared in `Ink.h`.

`kInktabletPointerEraser`

The eraser end of a stylus-like device.

Discussion

Pens used with modern graphics tablets often have a writing tip and an eraser tip. Some tablets also support pucks in addition to, or instead of, stylus-like devices. By comparing these constants to the contents of the `pointerType` element of a `TabletProximityRec` in Carbon or to the value returned by the `pointerType` message to tablet events (or mouse events with tablet data in them) in Cocoa, you can determine what kind of pointer device and which tip of a stylus-like device is being used with a graphics tablet. These `pointerType` data are only available in tablet-proximity events (not tablet-point events).

To ensure consistency between the values used by driver writers and the values used by applications, these constants are defined in terms of `NX_` constants from `IOKit/hidsystem/IOLLEvent.h`.

Availability

Available in Mac OS X v10.4 and later.

Result Codes

There are no results codes specific to Ink Services. Rather than returning `OSStatus` values, functions return `NULL` or specific, predetermined invalid responses when you pass invalid parameters to them. Designing the API in this way allows you to chain function calls and write code that is more compact.

Interface Builder Services Reference

Framework: Carbon/Carbon.h
Declared in IBCarbonRuntime.h

Overview

This reference describes the functions you use to unarchive interface objects you create using Interface Builder.

Carbon supports the Interface Builder APIs.

Functions by Task

Creating and Disposing of Nib References

`CreateNibReference` (page 1057)
Creates a reference to a nib file in the current bundle.

`CreateNibReferenceWithCFBundle` (page 1058)
Creates a reference to a nib file in the specified bundle.

`DisposeNibReference` (page 1060)
Disposes of a nib reference.

Unarchiving Menu Bars and Menus

`CreateMenuBarFromNib` (page 1056)
Unarchives a menu bar from a nib file.

`SetMenuBarFromNib` (page 1060)
Unarchives a menu bar from a nib file, then makes the menu bar available in your application.

`CreateMenuFromNib` (page 1056)
Unarchives a menu from a nib file.

Unarchiving Windows

`CreateWindowFromNib` (page 1059)
Unarchives a window from a nib file.

Functions

CreateMenuBarFromNib

Unarchives a menu bar from a nib file.

```
OSStatus CreateMenuBarFromNib (
   IBNibRef inNibRef,
   CFStringRef inName,
   Handle *outMenuBar
);
```

Parameters

inNibRef

A reference to the nib file that contains the menu bar you want to unarchive. You obtain this reference by calling the function `CreateNibReference` (page 1057) or `CreateNibReferenceWithCFBundle` (page 1058).

inName

A `CFStringRef` that denotes the menu bar you want to unarchive. This is the name you supplied to the menu bar in the Instances pane of Interface Builder. See the Base Services documentation for a description of the `CFStringRef` data type. You can use the Core Foundation function `CFSTR` to convert a string to a `CFString`.

outMenuBar

On output, points to a handle to the menu bar.

Return Value

A result code. See Runtime Errors (page 1061).

Discussion

You need to call the Menu Manager function `SetMenuBarFromNib` to make the unarchived menu bar available in your application.

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X 10.0 and later.

Not available to 64-bit applications.

Related Sample Code

BSDLLCTest

Declared In

`IBCarbonRuntime.h`

CreateMenuFromNib

Unarchives a menu from a nib file.

```
OSStatus CreateMenuFromNib (
   IBNibRef inNibRef,
   CFStringRef inName,
   MenuRef *outMenuRef
);
```

Parameters

inNibRef

A reference to the nib file that contains the menu you want to unarchive. You obtain this reference by calling the function `CreateNibReference` (page 1057) or `CreateNibReferenceWithCFBundle` (page 1058).

inName

A `CFStringRef` that denotes the menu you want to unarchive. This is the name you supplied to the menu in the Instance pane of Interface Builder. See the Base Services documentation for a description of the `CFStringRef` data type. You can use the Core Foundation function `CFSTR` to convert a string to a `CFString`.

outMenuRef

On output, points to a menu reference.

Return Value

A result code. See Runtime Errors (page 1061).

Discussion

After you call the function `CreateMenuFromNib` you need to call the Menu Manager function `InsertMenu` to make the unarchived menu available in your application.

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X 10.0 and later.

Not available to 64-bit applications.

Related Sample Code

QTCarbonShell

Declared In

`IBCarbonRuntime.h`

CreateNibReference

Creates a reference to a nib file in the current bundle.

```
OSStatus CreateNibReference (
   CFStringRef inNibName,
   IBNibRef *outNibRef
);
```

Parameters

inNibName

A `CFStringRef` that represents the name of a nib file you created for your application, but without the nib extension. See the Base Services documentation for a description of the `CFStringRef` data type. You can use the Core Foundation function `CFSTR` to convert a string to a `CFString`.

outNibRef

On output, points to a nib reference. You pass the nib reference to Interface Builder Services functions that unarchive objects from a nib file.

Return Value
A result code. See Runtime Errors (page 1061).

Discussion
Use this function if the nib file is located in the current bundle. Use the function `CreateNibReferenceWithCFBundle` (page 1058) if the nib file is located in a framework or other bundle that is not the current bundle.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Related Sample Code
BSDLLCTest
HID Config Save
HID Explorer
QTCarbonShell
QTMetaData

Declared In
`IBCarbonRuntime.h`

CreateNibReferenceWithCFBundle

Creates a reference to a nib file in the specified bundle.

```
OSStatus CreateNibReferenceWithCFBundle (
   CFBundleRef inBundle,
   CFStringRef inNibName,
   IBNibRef *outNibRef
);
```

Parameters

inBundle

A `CFBundleRef` to your application's bundle. See the Bundle Services documentation for a description of the `CFBundleRef` data type. You can get this reference by calling the appropriate Core Foundation Bundle Services functions.

inNibName

A `CFStringRef` that represents the name of a nib file you created for your application, but without the nib extension. See the Base Services documentation for a description of the `CFStringRef` data type. You can use the Core Foundation function `CFSTR` to convert a string to a `CFString`.

outNibRef

On output, points to a nib reference. You pass the nib reference to Interface Builder Services functions that unarchive objects from a nib file.

Return Value
A result code. See Runtime Errors (page 1061).

Discussion
Use this function if the nib file is located in a framework or other bundle that is not the current bundle. Use the function `CreateNibReference` (page 1057) if the nib file is located in the current bundle.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`IBCarbonRuntime.h`

CreateWindowFromNib

Unarchives a window from a nib file.

```
OSStatus CreateWindowFromNib (
   IBNibRef inNibRef,
   CFStringRef inName,
   WindowRef *outWindow
);
```

Parameters

inNibRef
A reference to the nib file that contains the window you want to unarchive. You obtain this reference by calling the function `CreateNibReference` (page 1057) or `CreateNibReferenceWithCFBundle` (page 1058).

inName
A `CFStringRef` that denotes the window you want to unarchive. This is the name you supplied to the window in the Instances pane of Interface Builder. See the Base Services documentation for a description of the `CFStringRef` data type. You can use the Core Foundation function `CFSTR` to convert a string to a `CFString`.

outWindow
On output, points to the window unarchived from the nib file.

Return Value
A result code. See Runtime Errors (page 1061).

Discussion
You need to call the Window Manager function `ShowWindow` to make the unarchived window visible.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Related Sample Code
BSDLLCTest
CarbonSketch
HID Config Save
HID Explorer
QTCarbonShell

Declared In
IBCarbonRuntime.h

DisposeNibReference

Disposes of a nib reference.

```
void DisposeNibReference (
   IBNibRef inNibRef
);
```

Parameters

inNibRef

A nib reference you created by calling the function `CreateNibReference` (page 1057) or `CreateNibReferenceWithCFBundle` (page 1058).

Return Value

Discussion
You should call the function `DisposeNibReference` immediately after you have finished unarchiving objects (windows, menus, menu bar, and so forth) from the nib file associated with the nib reference.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Related Sample Code
BSDLLCTest
CarbonSketch
HID Config Save
HID Explorer
QTMetaData

Declared In
IBCarbonRuntime.h

SetMenuBarFromNib

Unarchives a menu bar from a nib file, then makes the menu bar available in your application.

```
OSStatus SetMenuBarFromNib (
   IBNibRef inNibRef,
   CFStringRef inName
);
```

Parameters

inNibRef

A reference to the nib file that contains the menu bar you want to unarchive. You obtain this reference by calling the function `CreateNibReference` (page 1057) or `CreateNibReferenceWithCFBundle` (page 1058).

inName

A `CFStringRef` that denotes the menu bar you want to unarchive. This is the name you supplied to the menu bar in the Instances pane of Interface Builder. (The default name is "`MainMenu`".) See the Base Services documentation for a description of the `CFStringRef` data type. You can use the Core Foundation function `CFSTR` to convert a string to a `CFString`.

Return Value
A result code. See Runtime Errors (page 1061).

Discussion
The function `SetMenuBarFromNib` makes the menu bar visible and selectable by the user when your application opens. If you don't want the menu bar to be visible and selectable when it is unarchived, use the function `CreateMenuBarFromNib` (page 1056).

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Related Sample Code
CarbonCocoa_PictureCursor
HID Config Save
HID Explorer
QTCarbonShell
QTMetaData

Declared In
`IBCarbonRuntime.h`

Data Types

IBNibRef

A reference to a nib file.

```
typedef struct OpaqueIBNibRef * IBNibRef;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`IBCarbonRuntime.h`

Constants

Runtime Errors

Specify a problem that occurs when Interface Builder Services tries to load nib files or objects.

```
enum {
    kIBCarbonRuntimeCantFindNibFile = -10960,
    kIBCarbonRuntimeObjectNotOfRequestedType = -10961,
    kIBCarbonRuntimeCantFindObject = -10962
};
```

Constants

`kIBCarbonRuntimeCantFindNibFile`

Indicates the `CFStringRef` supplied for a nib file does not match the name of a nib file.

Available in Mac OS X v10.0 and later.

Declared in `IBCarbonRuntime.h`.

`kIBCarbonRuntimeObjectNotOfRequestedType`

Indicates the window, menu, or menu bar you want to unarchive isn't what you specified.

Available in Mac OS X v10.0 and later.

Declared in `IBCarbonRuntime.h`.

`kIBCarbonRuntimeCantFindObject`

Indicates the window, menu, or menu bar wasn't found.

Available in Mac OS X v10.0 and later.

Declared in `IBCarbonRuntime.h`.

Keyboard Layout Services Reference

Framework: Carbon/Carbon.h

Declared in Keyboards.h

Overview

Keyboard Layout Services is the programming interface that lets you obtain information about available keyboard layouts, set a keyboard layout, and access a keyboard layout property list. Keyboard Layout Services fully supports Unicode (`'uchr'`) as well as old style (`'KCHR'`) keyboard layouts.

Functions by Task

Working With Keyboard Layouts

`KBGetLayoutType` (page 1064)
Obtains the type of keyboard attached to the computer.

`KLGetCurrentKeyboardLayout` (page 1064) Deprecated in Mac OS X v10.5
Obtains the keyboard layout for the layout currently selected in the Keyboards menu.

`KLGetKeyboardLayoutAtIndex` (page 1065) Deprecated in Mac OS X v10.5
Obtains the keyboard layout reference associated with the specified index.

`KLGetKeyboardLayoutCount` (page 1066) Deprecated in Mac OS X v10.5
Returns the number of keyboard layouts.

`KLGetKeyboardLayoutProperty` (page 1066) Deprecated in Mac OS X v10.5
Obtains the value associated with the specified property tag for a keyboard layout.

`KLGetKeyboardLayoutWithIdentifier` (page 1067) Deprecated in Mac OS X v10.5
Obtains the keyboard layout reference associated with the specified identifier.

`KLGetKeyboardLayoutWithName` (page 1068) Deprecated in Mac OS X v10.5
Obtains the keyboard layout associated with the specified name.

`KLSetCurrentKeyboardLayout` (page 1068) Deprecated in Mac OS X v10.5
Sets the keyboard layout.

Unsupported Functions

`KLGetIndexedKeyboardLayout` (page 1065)
Obtains the keyboard layout reference associated with the specified index.

Functions

KBGetLayoutType

Obtains the type of keyboard attached to the computer.

```
PhysicalKeyboardLayoutType KBGetLayoutType (
   SInt16 iKeyboardType
);
```

Parameters

keyboardType

The keyboard type ID of the keyboard whose type you want to obtain. You can obtain the keyboard type ID by calling the Event Manager function `LMGetKbdType`.

Return Value

The type of the keyboard attached to the computer. See "Physical Keyboard Layout Types" (page 1072) for the values that can be returned.

Discussion

You can call the function `KBGetLayoutType` to determine whether the keyboard attached to the computer is ANSI, ISO, or JIS. You should call this function in Mac OS 9 only if the Gestalt selector `gestaltKeyboardLib` is present.

Availability

Not available in CarbonLib.

Available in Mac OS X version 10.0 and later.

Declared In

`Keyboards.h`

KLGetCurrentKeyboardLayout

Obtains the keyboard layout for the layout currently selected in the Keyboards menu. (Deprecated in Mac OS X v10.5.)

```
OSStatus KLGetCurrentKeyboardLayout (
   KeyboardLayoutRef *oKeyboardLayout
);
```

Parameters

oKeyboardLayout

On output, a pointer to the keyboard layout reference for keyboard layout currently selected in the Keyboard menu.

Return Value

A result code.

Availability

Not available in CarbonLib.

Available in Mac OS X version 10.2 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
Keyboards.h

KLGetIndexedKeyboardLayout

Obtains the keyboard layout reference associated with the specified index.

Not supported.

```
OSStatus KLGetIndexedKeyboardLayout (
    CFIndex iIndex,
    KeyboardLayoutRef * oKeyboardLayout
);
```

Availability
Unsupported. Not available in CarbonLib. Not available in Mac OS X version 10.2 and later.

Carbon Porting Notes

You should use the function KLGetKeyboardLayoutAtIndex (page 1065) instead.

Declared In
Keyboards.h

KLGetKeyboardLayoutAtIndex

Obtains the keyboard layout reference associated with the specified index. (Deprecated in Mac OS X v10.5.)

```
OSStatus KLGetKeyboardLayoutAtIndex (
   CFIndex iIndex,
   KeyboardLayoutRef *oKeyboardLayout
);
```

Parameters

iIndex

The index of the keyboard layout whose keyboard layout reference you want to retrieve. The index must be a value from 0 through N-1, where N is the value returned in the oCount parameter by the function KLGetKeyboardLayoutCount (page 1066).

oKeyboardLayout

On output, a pointer to the keyboard layout reference associated with the index specified by the iIndex parameter.

Return Value
A result code.

Discussion
You can call this function from within an iteration to access all of the available keyboard layouts.

Availability
Not available in CarbonLib.
Available in Mac OS X version 10.2 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
Keyboards.h

KLGetKeyboardLayoutCount

Returns the number of keyboard layouts. (Deprecated in Mac OS X v10.5.)

```
OSStatus KLGetKeyboardLayoutCount (
   CFIndex *oCount
);
```

Parameters

oCount

On output, a pointer to a variable that specifies the number of keyboard layouts.

Return Value
A result code.

Discussion
You can call the function `KLGetKeyboardLayoutCount` when you want to obtain the number of keyboard layouts available to the user. Once you know the number of keyboard layouts, you can iterate through them by setting up a loop and calling the function `KLGetKeyboardLayoutAtIndex` (page 1065).

Availability
Not available in CarbonLib.
Available in Mac OS X version 10.2 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
Keyboards.h

KLGetKeyboardLayoutProperty

Obtains the value associated with the specified property tag for a keyboard layout. (Deprecated in Mac OS X v10.5.)

```
OSStatus KLGetKeyboardLayoutProperty (
   KeyboardLayoutRef iKeyboardLayout,
   KeyboardLayoutPropertyTag iPropertyTag,
   const void **oValue
);
```

Parameters

iKeyboardLayout

A keyboard layout reference for the keyboard layout whose property value you want to retrieve. You must pass a valid keyboard layout reference. You can obtain a keyboard layout reference by calling the functions `KLGetKeyboardLayoutAtIndex` (page 1065), `KLGetKeyboardLayoutWithIdentifier` (page 1067), `KLGetKeyboardLayoutWithName` (page 1068),or `KLGetCurrentKeyboardLayout` (page 1064).

iPropertyTag

A keyboard layout property tag that specifies the property whose value you want to retrieve. See "Keyboard Layout Property Tag" (page 1070) for a list of the property tag constants you can supply.

oValue

On output, a pointer to the variable associated with the tag specified by the `iPropertyTag` parameter.

Return Value
A result code.

Discussion
You can use this function to obtain keyboard layout data, the keyboard identifier, the icon used for the keyboard menu, the keyboard layout name (including its localized name), the group identifier, and the keyboard layout kind. See "Keyboard Layout Property Tag" (page 1070) for a complete list of the available property tags.

Availability
Not available in CarbonLib.
Available in Mac OS X version 10.2 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Keyboards.h`

KLGetKeyboardLayoutWithIdentifier

Obtains the keyboard layout reference associated with the specified identifier. (Deprecated in Mac OS X v10.5.)

```
OSStatus KLGetKeyboardLayoutWithIdentifier (
   KeyboardLayoutIdentifier iIdentifier,
   KeyboardLayoutRef *oKeyboardLayout
);
```

Parameters

iIdentifier

The keyboard layout identifier for which you want to obtain a keyboard layout reference. See "Keyboard Layout Identifier" (page 1069) for a list of the constants you can supply.

oKeyboardLayout

On output, a pointer to the keyboard layout reference associated with the identifier specified by the `iIdentifer` parameter.

Return Value
A result code.

Availability
Not available in CarbonLib.
Available in Mac OS X version 10.2 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Keyboards.h`

KLGetKeyboardLayoutWithName

Obtains the keyboard layout associated with the specified name. (Deprecated in Mac OS X v10.5.)

```
OSStatus KLGetKeyboardLayoutWithName (
   CFStringRef iName,
   KeyboardLayoutRef *oKeyboardLayout
);
```

Parameters

iName

A `CFStringRef` that specifies the name of the keyboard layout whose keyboard layout reference you want to obtain. This name is the nonlocalized keyboard layout name.

oKeyboardLayout

On output, a pointer to the keyboard layout reference associated with the name specified by the `iName` parameter.

Return Value

A result code.

Discussion

You can call this function when you want to obtain the keyboard layout reference for a keyboard layout other than the active keyboard layout. If you want to obtain the keyboard layout reference for the active keyboard layout, call the function `KLGetCurrentKeyboardLayout` (page 1064).

Availability

Not available in CarbonLib.

Available in Mac OS X version 10.2 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Keyboards.h`

KLSetCurrentKeyboardLayout

Sets the keyboard layout. (Deprecated in Mac OS X v10.5.)

```
OSStatus KLSetCurrentKeyboardLayout (
   KeyboardLayoutRef iKeyboardLayout
);
```

Parameters

iKeyboardLayout

A keyboard layout reference for the keyboard layout you want to set. You must pass a valid keyboard layout reference. You can obtain a keyboard layout reference by calling the functions `KLGetKeyboardLayoutAtIndex` (page 1065), `KLGetKeyboardLayoutWithIdentifier` (page 1067), `KLGetKeyboardLayoutWithName` (page 1068),or `KLGetCurrentKeyboardLayout` (page 1064).

Return Value

A result code.

Availability

Not available in CarbonLib.

Available in Mac OS X version 10.2 and later.

Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Keyboards.h`

Data Types

KeyboardLayoutRef

Specifies a reference to an opaque keyboard layout data structure.

```
typedef struct OpaqueKeyboardLayoutRef * KeyboardLayoutRef;
```

Discussion
You can use the Keyboard Layout Services functions `KLGetKeyboardLayoutAtIndex` (page 1065), `KLGetKeyboardLayoutWithIdentifier` (page 1067), `KLGetKeyboardLayoutWithName` (page 1068),or `KLGetCurrentKeyboardLayout` (page 1064) to obtain a keyboard layout reference. If you want to obtain keyboard layout data, use the function `KLGetKeyboardLayoutProperty` (page 1066).

Availability
Available in Mac OS X v10.2 and later.

Declared In
`Keyboards.h`

Constants

Keyboard Layout Constants

Keyboard Layout Identifier

Specifies a layout identifier for a keyboard.

```
typedef SInt32 KeyboardLayoutIdentifier;
enum {
    kKLUSKeyboard = 0
};
```

Constants
`kKLUSKeyboard`
Specifies a US keyboard layout. All systems support this keyboard layout.
Available in Mac OS X v10.2 and later.
Declared in `Keyboards.h`.

Discussion
You can supply this constant as the `iIdentifier` parameter to the function `KLGetKeyboardLayoutWithIdentifier` (page 1067). This constant is the property value associated with the keyboard layout property tag `kKLIdentifier`.

Keyboard Layout Formats

Specify the layout format used for a keyboard.

```
typedef SInt32 KeyboardLayoutKind;
enum {
    kKLKCHRuchrKind = 0,
    kKLKCHRKind = 1,
    kKLuchrKind = 2
};
```

Constants

`kKLKCHRuchrKind`
Specifies that both `'KCHR'` and `'uchr'` formats are available.
Available in Mac OS X v10.2 and later.
Declared in `Keyboards.h`.

`kKLKCHRKind`
Specifies that only the `'KCHR'` format is available.
Available in Mac OS X v10.2 and later.
Declared in `Keyboards.h`.

`kKLuchrKind`
Specifies that only the `'uchr'` format is available.
Available in Mac OS X v10.2 and later.
Declared in `Keyboards.h`.

Discussion
These values are associated with the keyboard layout property tag `kKLKind`.

Keyboard Layout Property Tag

Specify the property tag for a keyboard layout.

```
typedef UInt32 KeyboardLayoutPropertyTag;
enum {
    kKLKCHRData = 0,
    kKLuchrData = 1,
    kKLIdentifier = 2,
    kKLIcon = 3,
    kKLLocalizedName = 4,
    kKLName = 5,
    kKLGroupIdentifier = 6,
    kKLKind = 7
};
```

Constants

`kKLKCHRData`
Specifies `'KCHR'` layout format. The data associated with this tag is a pointer (`const void *`) to keyboard layout data formatted as `'KCHR'` data. You can use `'KCHR'` data with the Event Manager function `KeyTranslate`.

Available in Mac OS X v10.2 and later.

Declared in `Keyboards.h`.

`kKLuchrData`
Specifies `'uchr'` layout format. The data associated with this tag is a pointer (`const void *`) to keyboard layout data formatted as `'uchr'` data. You can use `'uchr'` data with the Unicode Utilities function `UCKeyTranslate`.

Available in Mac OS X v10.2 and later.

Declared in `Keyboards.h`.

`kKLIdentifier`
Specifies a keyboard layout identifier. The data associated with this tag a "Keyboard Layout Identifier" (page 1069) constant.

Available in Mac OS X v10.2 and later.

Declared in `Keyboards.h`.

`kKLIcon`
Specifies a keyboard layout icon. The data associated with this tag is an icon reference (`IconRef`) that specifies the icon that appears in the Keyboard menu.

Available in Mac OS X v10.2 and later.

Declared in `Keyboards.h`.

`kKLLocalizedName`
Specifies the localized keyboard layout name. The data associated with this tag is a `CFStringRef` that specifies the localized name for the keyboard layout.

Available in Mac OS X v10.2 and later.

Declared in `Keyboards.h`.

`kKLName`
Specifies the keyboard layout name. The data associated with this tag is a `CFStringRef` that specifies the name for the keyboard layout.

Available in Mac OS X v10.2 and later.

Declared in `Keyboards.h`.

`kKLGroupIdentifier`
Specifies the keyboard layout group identifier. The data associated with this tag is an `SInt32` value that specifies the grouping of a keyboard layout in the Keyboard menu.

Available in Mac OS X v10.2 and later.

Declared in `Keyboards.h`.

`kKLKind`
Specifies the keyboard layout format. The data associated with this tag is a "Keyboard Layout Formats" (page 1070) constant.

Available in Mac OS X v10.2 and later.

Declared in `Keyboards.h`.

Discussion
You can supply any of these constants as the `iPropertyTag` parameter to the function `KLGetKeyboardLayoutProperty` (page 1066) to retrieve the data associated with the property tag.

Physical Keyboard Layout Types

Specify the layout type associated with a physical keyboard.

```
typedef UInt32 PhysicalKeyboardLayoutType;
enum {
    kKeyboardJIS = 'JIS ',
    kKeyboardANSI = 'ANSI',
    kKeyboardISO = 'ISO ',
    kKeyboardUnknown = kUnknownType
};
```

Constants
`kKeyboardJIS`
Specifies a JIS keyboard.

Available in Mac OS X v10.0 and later.

Declared in `Keyboards.h`.

`kKeyboardANSI`
Specifies an ANSI keyboard layout.

Available in Mac OS X v10.0 and later.

Declared in `Keyboards.h`.

`kKeyboardISO`
Specifies an ISO keyboard layout.

Available in Mac OS X v10.0 and later.

Declared in `Keyboards.h`.

`kKeyboardUnknown`
Specifies the keyboard layout type is unknown.

Available in Mac OS X v10.0 and later.

Declared in `Keyboards.h`.

Discussion
These constants are returned by the function `KBGetLayoutType` (page 1064).

Unsupported Constants

Trap Value

Specifies a trap value for Keyboard Layout Services.

```
enum {
    _KeyboardDispatch = 0xAA7A
};
```

Discussion
This constant is not needed in Mac OS X.

PS2 Error Codes

Specify an error returned by unsupported PS2 keyboard functions.

```
enum {
    errKBPS2KeyboardNotAvailable = -30850,
    errKBIlligalParameters = -30851,
    errKBFailSettingID = -30852,
    errKBFailTranslationTable = -30853
    errKBFailWritePreference = -30854
};
```

Discussion
These constants are returned as error codes by the unsupported PS2 keyboard functions. They are not relevant to the functions whose prefix is 'KL'.

Keychain Manager Reference

Framework:	CoreServices/CoreServices.h, Carbon/Carbon.h
Declared in	KeychainCore.h KeychainHI.h

Overview

The Keychain Manager is an API that provides a uniform way for your application to handle passwords for multiple users, multiple databases, or any situation in which a user must enter single or multiple passwords. You can use the Keychain Manager to provide secure storage for a user's passwords, cryptographic keys, and digital certificates.

This document, which describes KeychainLib 2.0, is relevant to you if your application needs to create and manage passwords and other secure data.

Important: Keychain Manager is being phased out and replaced by Keychain Services. Any new development should use Keychain Services. See Keychain Services Reference.

Carbon fully supports the Keychain Manager.

Functions by Task

Getting Information About the Keychain Manager

`KCGetKeychainManagerVersion` (page 1114)
Determines the version of the Keychain Manager installed on the user's system.

Creating and Disposing of Keychain References

`KCMakeKCRefFromAlias` (page 1119)
Creates a keychain reference from a keychain alias.

`KCMakeAliasFromKCRef` (page 1118)
Creates an alias to a keychain reference.

`KCReleaseKeychain` (page 1122)
Disposes of the memory associated with a keychain reference.

KCMakeKCRefFromFSSpec (page 1119) Deprecated in Mac OS X v10.5
Creates a keychain reference from a file specification record.

Managing Keychains

KCCreateKeychain (page 1094)
Creates an empty keychain.

kccreatekeychain (page 1095)

KCSetDefaultKeychain (page 1126)
Sets the default keychain.

KCGetDefaultKeychain (page 1112)
Obtains the default keychain.

KCGetStatus (page 1116)
Determines the permissions that are set in a keychain.

KCGetKeychainName (page 1115)
Determines the name of a keychain.

kcgetkeychainname (page 1115)

KCCountKeychains (page 1093)
Determines the number of available keychains.

KCGetIndKeychain (page 1112)
Obtains the reference to an indexed keychain.

Storing and Retrieving Passwords

KCAddAppleSharePassword (page 1080)
Adds a new AppleShare server password to the default keychain.

kcaddapplesharepassword (page 1082)

KCFindAppleSharePassword (page 1096)
Finds the first AppleShare password in the default keychain that matches the specified parameters.

kcfindapplesharepassword (page 1098)

KCAddInternetPassword (page 1086)
Adds a new Internet server password to the default keychain.

kcaddinternetpassword (page 1087)

KCAddInternetPasswordWithPath (page 1088)
Adds a new Internet server password with a specified path to the default keychain.

kcaddinternetpasswordwithpath (page 1089)

`KCFindInternetPassword` (page 1102)
Finds the first Internet password in the default keychain that matches the specified parameters.

`kcfindinternetpassword` (page 1104)

`KCFindInternetPasswordWithPath` (page 1105)
Finds the first Internet password in the default keychain that matches the specified parameters, including path information.

`kcfindinternetpasswordwithpath` (page 1107)

`KCAddGenericPassword` (page 1084)
Adds a new generic password to the default keychain.

`kcaddgenericpassword` (page 1085)

`KCFindGenericPassword` (page 1100)
Finds the first generic password in the default keychain matching the specified parameters.

`kcfindgenericpassword` (page 1102)

Creating and Disposing of Keychain Item References

`KCNewItem` (page 1120)
Creates a reference to a keychain item.

`KCReleaseItem` (page 1121)
Disposes of the memory occupied by a keychain item reference.

Manipulating Keychain Items

`KCAddItem` (page 1090)
Adds a password or other keychain item to the default keychain.

`KCDeleteItem` (page 1096)
Deletes a password or other keychain item from the default keychain.

`KCUpdateItem` (page 1129)
Updates a password or other keychain item.

`KCCopyItem` (page 1092)
Copies a password or other keychain item from one keychain to another.

`KCGetKeychain` (page 1113)
Determines the location of a password or other keychain item.

Setting and Obtaining Keychain Item Data

`KCSetAttribute` (page 1124)
Sets or edits keychain item data using a keychain item attribute structure.

`KCGetAttribute` (page 1109)
Determines keychain item data using a keychain item attribute structure.

`KCSetData` (page 1125)
Sets or edits keychain item data.

`KCGetData` (page 1111)
Determines keychain item data.

Searching for Keychain Items

`KCFindFirstItem` (page 1099)
Finds the first keychain item in a specified keychain that matches specified attributes.

`KCFindNextItem` (page 1108)
Finds the next keychain item matching the previously specified search criteria.

`KCReleaseSearch` (page 1122)
Disposes of the memory occupied by a search criteria reference.

Managing User Interaction

`KCLock` (page 1117)
Locks a keychain.

`KCUnlock` (page 1127)
Displays a dialog box that prompts the user for a password before unlocking a keychain.

`kcunlock` (page 1129)

`KCChangeSettings` (page 1091)
Displays a dialog box enabling the user to change the name, password, or settings of a keychain.

`KCSetInteractionAllowed` (page 1127)
Enables or disables Keychain Manager functions that display a user interface.

`KCIsInteractionAllowed` (page 1117)
Indicates whether Keychain Manager functions that display a user interaction will do so.

Registering Your Keychain Event Callback Function

`KCAddCallback` (page 1083)
Registers your keychain event callback function.

`KCRemoveCallback` (page 1123)
Unregisters your keychain event callback function.

Working With Your Keychain Manager Callback Function

`NewKCCallbackUPP` (page 1130)
Creates a UPP to your keychain event callback.

`InvokeKCCallbackUPP` (page 1079)
Invokes your keychain event callback.

`DisposeKCCallbackUPP` (page 1079)
Disposes of a UPP to your keychain event callback.

Unsupported Functions

`KCChooseCertificate` (page 1092)
Displays a list of certificates that the user can choose from.

`KCFindX509Certificates` (page 1109)
Finds the certificates in a keychain that match specified search criteria.

Functions

DisposeKCCallbackUPP

Disposes of a UPP to your keychain event callback.

Not recommended

```
void DisposeKCCallbackUPP (
   KCCallbackUPP userUPP
);
```

Parameters

userUPP
A Universal Procedure Pointer (UPP) to your keychain event callback function.

Return Value
A result code. See "Keychain Manager Result Codes" (page 1154).

Discussion
When you are finished with a UPP to your keychain event callback function, you should dispose of it by calling the `DisposeKCCallbackUPP` function.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

There is no replacement function available.

Declared In
`KeychainCore.h`

InvokeKCCallbackUPP

Invokes your keychain event callback.

Not recommended

```
OSStatus InvokeKCCallbackUPP (
   KCEvent keychainEvent,
   KCCallbackInfo *info,
   void *userContext,
   KCCallbackUPP userUPP
);
```

Parameters

keychainEvent

The keychain events you want your application to receive. See "Keychain Events Constants" (page 1140) for a description of possible values. The Keychain Manager tests the bitmask you pass in the `eventMask` parameter of the function `KCAddCallback` (page 1083) to determine which events to pass to your callback function. See "Keychain Events Mask" (page 1142) for a description of this bitmask.

info

A pointer to a structure of type `KCCallbackInfo` (page 1133) that provides information about the keychain event to your callback function. The Keychain Manager passes a pointer to this structure in the `info` parameter of your callback function.

userContext

A pointer to application-defined storage. The Keychain Manager passes this value in the `userContext` parameter of your callback function. Your application can use this to associate any particular call of the `InvokeKCCallbackUPP` function with any particular call of the keychain event callback function.

userUPP

A Universal Procedure Pointer to your keychain event callback function. For information on how to create a keychain event callback function, see `KCCallbackProcPtr` (page 1131).

Return Value

A result code. See "Keychain Manager Result Codes" (page 1154).

Discussion

You should not need to use the function `InvokeKCCallbackUPP`, as the system calls your `KCAddCallback` (page 1083) callback function for you.

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X 10.0 and later.

Carbon Porting Notes

There is no replacement function available.

Declared In

`KeychainCore.h`

KCAddAppleSharePassword

Adds a new AppleShare server password to the default keychain.

Not recommended

```
OSStatus KCAddAppleSharePassword (
   AFPServerSignature *serverSignature,
   StringPtr serverAddress,
   StringPtr serverName,
   StringPtr volumeName,
   StringPtr accountName,
   UInt32 passwordLength,
   const void *passwordData,
   KCItemRef *item
);
```

Parameters

serverSignature

A pointer to a 16-byte Apple File Protocol server signature block. Pass a value of type `AFPServerSignature` (page 1132). Pass `NULL` to match any server signature. The Keychain Manager identifies the location for the password by the information passed in the `serverAddress` and `serverSignature` parameters. You must pass a valid value in at least one of these parameters.

serverAddress

A pointer to a Pascal string containing the server address, which may be specified as an AppleTalk zone name, a DNS domain name (in the format "xxx.yyy.zzz"), or an IP address (in the format "111.222.333.444"). The Keychain Manager identifies the location for the password by the information passed in the `serverAddress` and `serverSignature` parameters. You must pass a valid value in at least one of these parameters.

serverName

A pointer to a Pascal string containing the server name.

volumeName

A pointer to a Pascal string containing the volume name.

accountName

A pointer to a Pascal string containing the account name.

passwordLength

The length of the buffer pointed to by `passwordData`.

passwordData

A pointer to a buffer which will hold the returned password data. Before calling `KCAddAppleSharePassword`, allocate enough memory for the buffer to hold the data you want to store.

item

On return, a pointer to a reference to the added item. Pass `NULL` if you don't want to obtain this reference.

Return Value

A result code. See "Keychain Manager Result Codes" (page 1154). The result code `errKCNoDefaultKeychain` indicates that no default keychain could be found. The result code `errKCDuplicateItem` indicates that you tried to add a password that already exists in the keychain. The result code `errKCDataTooLarge` indicates that you tried to add more data than is allowed for a record of this type.

Discussion

The `KCAddAppleSharePassword` function adds a new AppleShare server password to the default keychain that is uniquely identified by the `serverName`, `volumeName`, and `accountName` parameters, and a location specified either by the `serverAddress` or `serverSignature` parameters. The `KCAddAppleSharePassword` function optionally returns a reference to the newly added item.

Most applications do not need to store AppleShare password data, as this is handled transparently by the AppleShare client software. To be compatible with the AppleShare client, you should store a fully-specified File Manager structure `AFPXVolMountInfo` as the password data.

The `KCAddAppleSharePassword` function automatically calls the function `KCUnlock` (page 1127) to display the Unlock Keychain dialog box if the keychain containing the password is currently locked.

You can also call the function `kcaddapplesharepassword` to add an AppleShare server password to the default keychain. `kcaddapplesharepassword` requires that you pass a pointer to a C string instead of a pointer to a Pascal string for the `serverAddress`, `serverName`, `volumeName`, `accountName`, and `passwordData` parameters.

Version Notes
Available beginning with KeychainLib 1.0. In KeychainLib 1.0, the `kcaddapplesharepassword` function provides the same functionality as `KCAddAppleSharePassword`, except that it accepts C strings rather than Pascal strings as arguments. In KeychainLib 2.0, you should use `KCAddAppleSharePassword`, since `kcaddapplesharepassword` is provided for convenience only and may be removed from the header file at some point in the future.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

It is recommended that you use internet passwords instead of AppleShare passwords. Use the `SecKeychainAddInternetPassword` function in Keychain Services instead.

Declared In
`KeychainHI.h`

kcaddapplesharepassword

Not recommended

```
OSStatus kcaddapplesharepassword (
   AFPServerSignature *serverSignature,
   const char *serverAddress,
   const char *serverName,
   const char *volumeName,
   const char *accountName,
   UInt32 passwordLength,
   const void *passwordData,
   KCItemRef *item
);
```

Discussion
This function is available for convenience only and may be removed. Use the function `KCAddAppleSharePassword` (page 1080) instead.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

It is recommended that you use internet passwords instead of AppleShare passwords. Use the `SecKeychainAddInternetPassword` function in Keychain Services instead.

Declared In
`KeychainHI.h`

KCAddCallback

Registers your keychain event callback function.

Not recommended

```
OSStatus KCAddCallback (
   KCCallbackUPP callbackProc,
   KCEventMask eventMask,
   void *userContext
);
```

Parameters

callbackProc

A Universal Procedure Pointer (UPP) to your keychain event callback function, described in `KCCallbackProcPtr` (page 1131). You indicate the type of keychain events you want to receive by passing a bitmask of the desired events in the `eventMask` parameter. To create a UPP to your callback function, call the function `NewKCCallbackUPP` (page 1130).

eventMask

A bitmask indicating the keychain events that your application wishes to be notified of. See "Keychain Events Mask" (page 1142) for a description of this bitmask. The Keychain Manager tests this mask to determine the keychain events that you wish to receive, and passes these events in the `keychainEvent` parameter of your callback function. See "Keychain Events Constants" (page 1140) for a description of these events.

userContext

A pointer to application-defined storage that will be passed to your callback function. Your application can use this to associate any particular call of the `KCAddCallback` function with any particular call of your keychain event callback function.

Return Value
A result code. See "Keychain Manager Result Codes" (page 1154). The result code `errKCDuplicateCallback` indicates that your callback function is already registered.

Discussion
You can register your callback function by passing a UPP to it in the `callbackProc` parameter of the `KCAddCallback` function. Once you have done so, the Keychain Manager calls the function `InvokeKCCallbackUPP` (page 1079) when the keychain event specified in the `eventMask` parameter occurs. In turn, the function `InvokeKCCallbackUPP` (page 1079) passes the keychain event, information about the event, and application-defined storage to your keychain event callback function.

Version Notes
Available beginning with KeychainLib 1.0.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainAddCallback` function in Keychain Services instead.

Declared In
`KeychainCore.h`

KCAddGenericPassword

Adds a new generic password to the default keychain.

Not recommended

```
OSStatus KCAddGenericPassword (
   StringPtr serviceName,
   StringPtr accountName,
   UInt32 passwordLength,
   const void *passwordData,
   KCItemRef *item
);
```

Parameters

serviceName
: A pointer to a Pascal string containing an application-defined service name.

accountName
: A pointer to a Pascal string containing an application-defined account name.

passwordLength
: The length of the password data to be stored

passwordData
: A pointer to the buffer that holds the returned password data. Before calling the `KCAddGenericPassword` function, allocate enough memory for the buffer to hold the data you want to store.

item
: On return, a pointer to a reference to the added item. Pass `NULL` if you don't want to obtain this reference.

Return Value
A result code. See "Keychain Manager Result Codes" (page 1154). The result code `errKCNoDefaultKeychain` indicates that no default keychain could be found. The result code `errKCDuplicateItem` indicates that you tried to add a password that already exists in the keychain. The result code `errKCDataTooLarge` indicates that you tried to add more data than is allowed for a record of this type.

Discussion
The `KCAddGenericPassword` function adds a new generic password to the default keychain. Required parameters to identify the password are `serviceName` and `accountName`, which are application-defined strings. The `KCAddGenericPassword` function optionally returns a reference to the newly added item.

You can use the `KCAddGenericPassword` function to add passwords for accounts other than Internet or AppleShare. For example, you might add passwords for your database or scheduling programs.

You can also call the function `kcaddgenericpassword` to add a new generic password to the default keychain. The difference between the two functions is that the `kcaddgenericpassword` function takes a pointer to a C string instead of a Pascal string in the `serverAddress`, `serverName`, `volumeName`, `accountName`, and `passwordData` parameters.

The `KCAddGenericPassword` function automatically calls the function `KCUnlock` (page 1127) to display the Unlock Keychain dialog box if the keychain containing the password is currently locked.

Version Notes
Available beginning with KeychainLib 1.0. In KeychainLib 1.0, the `kcaddgenericpassword` function provides the same functionality as the function `KCAddGenericPassword`, except that it accepts C strings rather than Pascal strings as arguments. In KeychainLib 2.0, you should use the `KCAddGenericPassword` function, since the `kcaddgenericpassword` function is provided for convenience only and may be removed from the header file at some point in the future.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainAddGenericPassword` function in Keychain Services instead.

Declared In
`KeychainHI.h`

kcaddgenericpassword

Not recommended

```
OSStatus kcaddgenericpassword (
   const char *serviceName,
   const char *accountName,
   UInt32 passwordLength,
   const void *passwordData,
   KCItemRef *item
);
```

Discussion
This function is available for convenience only and may be removed. Use the function `KCAddGenericPassword` (page 1084) instead.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainAddGenericPassword` function in Keychain Services instead.

Declared In
`KeychainHI.h`

KCAddInternetPassword

Adds a new Internet server password to the default keychain.

Not recommended

```
OSStatus KCAddInternetPassword (
   StringPtr serverName,
   StringPtr securityDomain,
   StringPtr accountName,
   UInt16 port,
   OSType protocol,
   OSType authType,
   UInt32 passwordLength,
   const void *passwordData,
   KCItemRef *item
);
```

Parameters

serverName

A pointer to a Pascal string containing the server name.

securityDomain

A pointer to a Pascal string containing the security domain. This parameter is optional, as not all protocols will require it.

accountName

A pointer to a Pascal string containing the account name.

port

The TCP/IP port number. Pass the constant `kAnyPort`, described in "Default Internet Port Constant" (page 1140), to specify any port.

protocol

The protocol associated with this password. See "Keychain Protocol Type Constants" (page 1151) for a description of possible values. Pass the constant `kAnyProtocol`, described in "Default Internet Protocol And Authentication Type Constants" (page 1140), to specify any protocol.

authType

The authentication scheme used. See "Authentication Type Constants" (page 1136) for a description of possible values. Pass the constant `kAnyAuthType`, described in "Default Internet Protocol And Authentication Type Constants" (page 1140), to specify any authentication scheme.

passwordLength

The length of the buffer pointed to by `passwordData`.

passwordData

A pointer to a buffer that holds the returned password data. Before calling the `KCAddInternetPasswordWithPath` function, allocate enough memory for the buffer to hold the data you want to store.

item

On return, a pointer to a reference to the added item. Pass `NULL` if you don't want to obtain this reference.

Return Value

A result code. See "Keychain Manager Result Codes" (page 1154). The result code `errKCNoDefaultKeychain` indicates that no default keychain could be found. The result code `errKCDuplicateItem` indicates that you tried to add a password that already exists in the keychain. The result code `errKCDataTooLarge` indicates that you tried to add more data than is allowed for a record of this type.

Discussion

The `KCAddInternetPassword` function adds a new Internet server password to the default keychain. Required parameters to identify the password are `serviceName` and `accountName` (you cannot pass `NULL` for both parameters). In addition, some protocols may require an optional value in the `securityDomain` parameter when authentication is requested. `KCAddInternetPassword` optionally returns a reference to the newly added item.

The `KCAddInternetPassword` function automatically calls the function `KCUnlock` (page 1127) to display the Unlock Keychain dialog box if the keychain containing the password is currently locked.

You can also call the function `kcaddinternetpassword` to add a new Internet server password to the default keychain. The `kcaddinternetpassword` function requires that you pass a pointer to a C string instead of a pointer to a Pascal string for the `serverAddress`, `serverName`, `volumeName`, `accountName`, and `passwordData` parameters.

Version Notes

Available beginning with KeychainLib 1.0. In KeychainLib 1.0, the `kcaddinternetpassword` function provides the same functionality as `KCAddInternetPassword`, except that it accepts C strings rather than Pascal strings as arguments. In KeychainLib 2.0, you should use `KCAddInternetPassword`, since `kcaddinternetpassword` is provided for convenience only and may be removed from the header file at some point in the future.

Availability

Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainAddInternetPassword` function in Keychain Services instead.

Declared In

`KeychainHI.h`

kcaddinternetpassword

Not recommended

```
OSStatus kcaddinternetpassword (
   const char *serverName,
   const char *securityDomain,
   const char *accountName,
   UInt16 port,
   OSType protocol,
   OSType authType,
   UInt32 passwordLength,
   const void *passwordData,
   KCItemRef *item
);
```

Discussion

This function is available for convenience only and may be removed. Use the function `KCAddInternetPassword` (page 1086) instead.

Availability

Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainAddInternetPassword` function in Keychain Services instead.

Declared In

`KeychainHI.h`

KCAddInternetPasswordWithPath

Adds a new Internet server password with a specified path to the default keychain.

Not recommended

```
OSStatus KCAddInternetPasswordWithPath (
   StringPtr serverName,
   StringPtr securityDomain,
   StringPtr accountName,
   StringPtr path,
   UInt16 port,
   OSType protocol,
   OSType authType,
   UInt32 passwordLength,
   const void *passwordData,
   KCItemRef *item
);
```

Parameters

serverName

A pointer to a Pascal string containing the server name.

securityDomain

A pointer to a Pascal string containing the security domain. This parameter is optional, as not all protocols will require it.

accountName

A pointer to a Pascal string containing the account name.

path

A pointer to a Pascal string containing additional information that specifies a file or directory on the server specified by the `serverName` parameter. In a typical URL, path information begins directly after the first slash ("/") character following the server name. This parameter is optional.

port

The TCP/IP port number. Pass the constant `kAnyPort`, described in "Default Internet Port Constant" (page 1140), to specify any port.

protocol

The protocol associated with this password. See "Keychain Protocol Type Constants" (page 1151) for a description of possible values. Pass the constant `kAnyProtocol`, described in "Default Internet Protocol And Authentication Type Constants" (page 1140), to specify any protocol.

authType

The authentication scheme used. See "Authentication Type Constants" (page 1136) for a description of possible values. Pass the constant `kAnyAuthType`, described in "Default Internet Protocol And Authentication Type Constants" (page 1140), to specify any authentication scheme.

passwordLength

The length of the buffer pointed to by `passwordData`.

passwordData

A pointer to a buffer which will hold the returned password data. Before calling `KCAddInternetPasswordWithPath`, allocate enough memory for the buffer to hold the data you want to store.

item

On return, a pointer to a reference to the added item. Pass `NULL` if you don't want to obtain this reference.

Return Value

A result code. See "Keychain Manager Result Codes" (page 1154). The result code `errKCNoDefaultKeychain` indicates that no default keychain could be found. The result code `errKCDuplicateItem` indicates that you tried to add a password that already exists in the keychain. The result code `errKCDataTooLarge` indicates that you tried to add more data than is allowed for a record of this type.

Discussion

The `KCAddInternetPasswordWithPath` function enables you to specify path information when adding a new Internet server password to the default keychain. Required parameters to identify the password are `serviceName` and `accountName` (you cannot pass `NULL` for both parameters). In addition, some protocols may require an optional `securityDomain` when authentication is requested. `KCAddInternetPasswordWithPath` optionally returns a reference to the newly added item.

The `KCAddInternetPasswordWithPath` function automatically calls the function `KCUnlock` (page 1127) to display the Unlock Keychain dialog box if the keychain containing the password is currently locked.

You can also call the function `kcaddinternetpasswordwithpath` to add a new Internet server password to the default keychain. The function `kcaddinternetpasswordwithpath` requires that you pass a pointer to a C string instead of a pointer to a Pascal string for the `serverAddress`, `serverName`, `volumeName`, `accountName`, and `passwordData` parameters.

Version Notes

Available beginning with KeychainLib 2.0. In KeychainLib 1.0, the `kcaddinternetpasswordwithpath` function provides the same functionality as the `KCAddInternetPasswordWithPath` function, except that it accepts C strings rather than Pascal strings as arguments. In KeychainLib 2.0, you should use the `KCAddInternetPasswordWithPath` function, since the function `kcaddinternetpasswordwithpath` is provided for convenience only and may be removed from the header file at some point in the future.

Availability

Available in CarbonLib 1.1 and later when KeychainLib 2.0 or later is present.

Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainAddInternetPassword` function in Keychain Services instead.

Declared In

`KeychainHI.h`

kcaddinternetpasswordwithpath

Not recommended

```
OSStatus kcaddinternetpasswordwithpath (
   const char *serverName,
   const char *securityDomain,
   const char *accountName,
   const char *path,
   UInt16 port,
   OSType protocol,
   OSType authType,
   UInt32 passwordLength,
   const void *passwordData,
   KCItemRef *item
);
```

Discussion
This function is available for convenience only and may be removed. Use the function `KCAddInternetPasswordWithPath` (page 1088) instead.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainAddInternetPassword` function in Keychain Services instead.

Declared In
`KeychainHI.h`

KCAddItem

Adds a password or other keychain item to the default keychain.

Not recommended

```
OSStatus KCAddItem (
   KCItemRef item
);
```

Parameters

item

A reference to the keychain item you wish to add. If you pass an existing item in the keychain, the item is updated. If you pass an item that has not been previously added to the keychain and an identical item already exists in the keychain, `KCAddItem` returns the result code `errKCDuplicateItem`.

Return Value
A result code. See "Keychain Manager Result Codes" (page 1154). The result code `errKCNoDefaultKeychain` indicates that no default keychain could be found. The result code `errKCInvalidItemRef` indicates that the specified keychain item reference was invalid. The result code `errKCDuplicateItem` indicates that you tried to add a new item that already exists in the keychain.

Discussion
You can use the `KCAddItem` function to add a password or other keychain item to the permanent data store of the default keychain. If you want to add a password to a keychain other than the default, call the function `KCSetDefaultKeychain` (page 1126) to change the default keychain. The `KCAddItem` function automatically calls the function `KCUnlock` (page 1127) to display the Unlock Keychain dialog box if the keychain containing the item is currently locked.

Version Notes
Available beginning with KeychainLib 1.0.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

When you use Keychain Services to create an item, it is always added to the specified keychain at creation time.

Declared In
`KeychainHI.h`

KCChangeSettings

Displays a dialog box enabling the user to change the name, password, or settings of a keychain.

Not recommended

```
OSStatus KCChangeSettings (
   KCRef keychain
);
```

Parameters

keychain
A reference to an unlocked keychain. Pass in `NULL` to specify the default keychain.

Return Value
A result code. See "Keychain Manager Result Codes" (page 1154). The result code `errUserCanceled` indicates that the user pressed the Cancel button in the Change Settings dialog box. The result code `errKCNoDefaultKeychain` indicates that the default keychain could not be found. The result code `errKCInvalidKeychain` indicates that the specified keychain is invalid.

Discussion
Typically, your application should not call the `KCChangeSettings` function. You would only call the `KCChangeSettings` function in response to a user's request to change keychain settings, name, or password. Note that you cannot change a keychain passphrase directly. You must call the `KCChangeSettings` function and allow the user to change it.

Version Notes
Available beginning with KeychainLib 2.0.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainSetSettings` function in Keychain Services instead.

Declared In
`KeychainHI.h`

KCChooseCertificate

Displays a list of certificates that the user can choose from.

Unsupported

```
OSStatus KCChooseCertificate (
    CFArrayRef items,
    KCItemRef *certificate,
    CFArrayRef policyOIDs,
    KCVerifyStopOn stopOn
);
```

Parameters

items

An array of certificate references.

certificate

If the `items` array only contains one certificate, on return, a pointer to that certificate. In this case, no user interface is displayed.

policyOIDs

An array of trust policy options used for Macintosh file signing. To obtain a pointer to this array, call the function `SecMacGetDefaultPolicyOIDs`.

stopOn

The criteria to use in selecting the certificates to display. See "Certificate Verification Criteria" (page 1139) for a description of this mask.

Return Value

A result code. See "Keychain Manager Result Codes" (page 1154). The result code `userCanceledErr` indicates that the user pressed the Cancel button in the user interface.

Discussion

The `KCChooseCertificate` function displays a list of the certificates from which the user can choose. If only one certificate matches the criteria, the reference is passed back in the `certificate` parameter and no user interface is presented.

Version Notes

Available beginning with KeychainLib 2.0.

Carbon Porting Notes

This function is obsolete. There is currently no replacement.

Declared In

`KeychainHI.h`

KCCopyItem

Copies a password or other keychain item from one keychain to another.

Not recommended

```
OSStatus KCCopyItem (
   KCItemRef item,
   KCRef destKeychain,
   KCItemRef *copy
);
```

Parameters

item

A reference to the keychain item you wish to copy.

destKeychain

A reference to the keychain into which the item is to be copied.

copy

A pointer to a reference to the new copied keychain item.

Return Value

A result code. See "Keychain Manager Result Codes" (page 1154). The result code `errKCReadOnly` indicates that the destination keychain is read only. The result code `errKCNoSuchClass` indicates that the item has an invalid keychain item class. The result code `errKCInvalidItemRef` indicates that the specified keychain item reference was invalid.

Discussion

You can use the `KCCopyItem` function to copy a keychain item from one keychain to another. The `KCCopyItem` function automatically calls the function `KCUnlock` (page 1127) to display the Unlock Keychain dialog box if the keychain containing the item to be copied is currently locked.

Version Notes

Available beginning with KeychainLib 2.0.

Availability

Available in CarbonLib 1.1 and later when KeychainLib 2.0 or later is present.

Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainItemCopyContent` function in Keychain Services instead.

Declared In

`KeychainCore.h`

KCCountKeychains

Determines the number of available keychains.

Not recommended

```
UInt16 KCCountKeychains (
   void
);
```

Parameters

Return Value

The number of available keychains. This includes all keychains in the Keychains folder, as well as any other keychains known to the Keychain Manager.

Discussion
This function reports the number of keychains known to the Keychain Manager. These keychains are created by the function `KCCreateKeychain` (page 1094).

Version Notes
Available beginning with KeychainLib 1.0.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainCopySearchList` function in Keychain Services followed by a call to `CFArrayGetCount` instead.

Declared In
`KeychainCore.h`

KCCreateKeychain

Creates an empty keychain.

Not recommended

```
OSStatus KCCreateKeychain (
   StringPtr password,
   KCRef *keychain
);
```

Parameters

password

A pointer to a Pascal string representing the password string which will be used to protect the new keychain. If you pass `NULL`, the Keychain Setup dialog box will be displayed to obtain it.

keychain

A pointer to a reference to the keychain you wish to create. You create a keychain reference by calling the function `KCMakeKCRefFromFSSpec` (page 1119) or `KCMakeKCRefFromAlias` (page 1119). If you pass a pointer to a keychain reference, the user will not need to be prompted for a name and location; in all other cases, `KCCreateKeychain` will interactively request this information from the user. If you pass a pointer to a `NULL` keychain reference, the Keychain Manager allocates the memory for the keychain reference and returns it in this parameter. Pass a `NULL` pointer if you do not need a reference returned.

Return Value
A result code. See "Keychain Manager Result Codes" (page 1154). The result code `userCanceledErr` indicates that the user pressed the Cancel button in the create keychain. The result code `errKCDuplicateKeychain` indicates that the user tried to create a keychain which already exists. The result code `errKCInvalidKeychain` indicates that the specified keychain is invalid. Additional errors may be returned if the keychain could not be created (for example, a file system or network error may be returned if there is no write access to the storage media).

Discussion
The `KCCreateKeychain` function creates an empty keychain. The `keychain` and `password` parameters are optional. If user interaction to create a keychain is posted, the newly-created keychain is automatically unlocked after creation.

You can also call the function `kccreatekeychain` to create an empty keychain. The function `kccreatekeychain` requires that you pass a pointer to a C string instead of a pointer to a Pascal string in the `password` parameter.

Special Considerations

It is recommended that the `KCCreateKeychain` function not be explicitly called by applications. Instead, you should call one of the add functions in "Storing and Retrieving Passwords" (page 1076) to add a password to the default keychain. If a default keychain does not exist, it is created automatically.

When you are finished with a keychain, you must deallocate its memory by calling the function `KCReleaseKeychain` (page 1122).

Version Notes

Available beginning with KeychainLib 1.0. In KeychainLib 1.0, the `kccreatekeychain` function provides the same functionality as `KCCreateKeychain`. In KeychainLib 2.0, you should use `KCCreateKeychain`, since `kccreatekeychain` is provided for convenience only and may be removed from the header file at some point in the future.

Availability

Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainCreate` function in Keychain Services instead.

Declared In

`KeychainHI.h`

kccreatekeychain

Not recommended

```
OSStatus kccreatekeychain (
   const char *password,
   KCRef *keychain
);
```

Discussion

This function is available for convenience only and may be removed. Use the function `KCCreateKeychain` (page 1094) instead.

Availability

Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainCreate` function in Keychain Services instead.

Declared In

`KeychainHI.h`

KCDeleteItem

Deletes a password or other keychain item from the default keychain.

Not recommended

```
OSStatus KCDeleteItem (
   KCItemRef item
);
```

Parameters

item

A reference to the keychain item you wish to delete. If you pass an item that has not been previously added to the keychain, the function `KCDeleteItem` does nothing and returns `noErr`.

Return Value

A result code. See "Keychain Manager Result Codes" (page 1154). The result code `errKCNoDefaultKeychain` indicates that no default keychain could be found. The result code `errKCInvalidItemRef` indicates that the specified keychain item reference was invalid.

Discussion

You can use the `KCDeleteItem` function to delete a keychain item from the permanent data store of the default keychain. The `KCDeleteItem` function automatically calls the function `KCUnlock` (page 1127) to display the Unlock Keychain dialog box if the keychain containing the item is currently locked.

Special Considerations

The `KCDeleteItem` function does not dispose the memory occupied by the item reference. To do so, call the function `KCReleaseItem` (page 1121) when you are finished with an item.

Version Notes

Available beginning with KeychainLib 1.0.

Availability

Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainItemDelete` function in Keychain Services instead.

Declared In

`KeychainCore.h`

KCFindAppleSharePassword

Finds the first AppleShare password in the default keychain that matches the specified parameters.

Not recommended

```
OSStatus KCFindAppleSharePassword (
   AFPServerSignature *serverSignature,
   ConstStringPtr serverAddress,
   ConstStringPtr serverName,
   ConstStringPtr volumeName,
   ConstStringPtr accountName,
   UInt32 maxLength,
   void *passwordData,
   UInt32 *actualLength,
   KCItemRef *item
);
```

Parameters

serverSignature

A pointer to a 16-byte Apple File Protocol server signature block. Pass a value of type `AFPServerSignature` (page 1132). Pass `NULL` to match any server signature. The Keychain Manager identifies the location for the password by the information passed in the `serverAddress` and `serverSignature` parameters. You must pass a valid value in at least one of these parameters.

serverAddress

A pointer to a Pascal string containing the server address, which may be specified as an AppleTalk zone name, a DNS domain name (in the format "xxx.yyy.zzz"), or an IP address (in the format "111.222.333.444"). The Keychain Manager identifies the location for the password by the information passed in the `serverAddress` and `serverSignature` parameters. You must pass a valid value in at least one of these parameters.

serverName

A pointer to a Pascal string containing the server name. Pass `NULL` to match any server name.

volumeName

A pointer to a Pascal string containing the volume name. Pass `NULL` to match any volume name.

accountName

A pointer to a Pascal string containing the account name. Pass `NULL` to match any account name.

maxLength

The length of the buffer pointed to by `passwordData`.

passwordData

A pointer to a buffer which will hold the returned password data. Before calling `KCFindAppleSharePassword`, allocate enough memory for the buffer to hold the data you want to store. Pass `NULL` if you want to obtain the item reference but not the password data. In this case, you must also pass `NULL` in the `actualLength` parameter. On return, a pointer to the returned password data.

actualLength

On return, the actual length of the password data that was retrieved. If the buffer pointed to by `passwordData` is smaller than the actual length of the data, `KCFindAppleSharePassword` returns the result code `errKCBufferTooSmall`. In this case, your application must allocate a new buffer of sufficient size before calling `KCFindAppleSharePassword` again.

item

On return, a pointer to a reference to the found item. Pass `NULL` if you don't want to obtain this reference.

Return Value
A result code. See "Keychain Manager Result Codes" (page 1154). The result code `errKCNoDefaultKeychain` indicates that no default keychain was found. The result code `errKCItemNotFound` indicates that no matching password item was found. The result code `errKCBufferTooSmall` indicates that your application must allocate a new buffer of sufficient size before calling `KCFindAppleSharePassword` again.

Discussion
The `KCFindAppleSharePassword` function finds the first AppleShare password item which matches the attributes you provide. The buffer specified in the `passwordData` parameter must be large enough to hold the password data, otherwise `KCFindAppleSharePassword` returns the result code `errKCBufferTooSmall`. In this case, your application must allocate a new buffer of sufficient size before calling the `KCFindAppleSharePassword` function again. The `KCFindAppleSharePassword` function optionally returns a reference to the found item.

The `KCFindAppleSharePassword` function automatically calls the function `KCUnlock` (page 1127) to display the Unlock Keychain dialog box if the keychain containing the password is currently locked.

You can also call the function `kcfindapplesharepassword` to find the first AppleShare server password matching specified attributes. `kcfindapplesharepassword` requires that you pass a pointer to a C string instead of a pointer to a Pascal string for the `serverAddress`, `serverName`, `volumeName`, `accountName`, and `passwordData` parameters.

Version Notes
Available beginning with KeychainLib 1.0. In KeychainLib 1.0, the `kcfindapplesharepassword` function provides the same functionality as `KCFindAppleSharePassword`, except that it accepts C strings rather than Pascal strings as arguments. In KeychainLib 2.0, you should use `KCFindAppleSharePassword`, since `kcfindapplesharepassword` is provided for convenience only and may be removed from the header file at some point in the future.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the Keychain Services function `SecKeychainSearchCreateFromAttributes` followed by the `SecKeychainSearchCopyNext` function instead.

Declared In
`KeychainCore.h`

kcfindapplesharepassword
Not recommended

```
OSStatus kcfindapplesharepassword (
   AFPServerSignature *serverSignature,
   const char *serverAddress,
   const char *serverName,
   const char *volumeName,
   const char *accountName,
   UInt32 maxLength,
   void *passwordData,
   UInt32 *actualLength,
   KCItemRef *item
);
```

Discussion
This function is available for convenience only and may be removed. Use the function `KCFindAppleSharePassword` (page 1096) instead.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the Keychain Services function `SecKeychainSearchCreateFromAttributes` followed by the `SecKeychainSearchCopyNext` function instead.

Declared In
`KeychainCore.h`

KCFindFirstItem

Finds the first keychain item in a specified keychain that matches specified attributes.

Not recommended

```
OSStatus KCFindFirstItem (
   KCRef keychain,
   const KCAttributeList *attrList,
   KCSearchRef *search,
   KCItemRef *item
);
```

Parameters

keychain
A reference to the keychain that you wish to search. If you pass a locked keychain, the Unlock Keychain dialog box is displayed. If you pass `NULL`, the `KCFindFirstItem` function searches all unlocked keychains.

attrList
A pointer to a list of 0 or more structures containing information about the keychain item attributes to be matched. Pass `NULL` to match any attribute.

search
On return, a pointer to a reference to the current search criteria.

item
On return, a pointer to the first matching keychain item.

Return Value
A result code. See "Keychain Manager Result Codes" (page 1154). The result code `errKCNoDefaultKeychain` indicates that no default keychain could be found. The result code `errKCItemNotFound` indicates that no matching keychain item was found. The result code `errKCNoSuchAttr` indicates that the specified attribute is undefined for this item class.

Discussion
The `KCFindFirstItem` function returns a reference to the first keychain item in a keychain that matches a list of attributes. The `KCFindFirstItem` function also returns a reference to the search criteria used. You should pass the returned search criteria in the `searchRef` parameter of the function `KCFindNextItem` (page 1108).

The `KCFindFirstItem` function automatically calls the function `KCUnlock` (page 1127) to display the Unlock Keychain dialog box if the keychain containing the item you are searching for is currently locked.

Special Considerations

When you are completely finished with a search, you should the functions `KCReleaseItem` (page 1121) and `KCReleaseSearch` (page 1122) to release the memory occupied by the keychain item and search criteria reference.

Version Notes
Available beginning with KeychainLib 1.0.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the Keychain Services function `SecKeychainSearchCreateFromAttributes` followed by a call to `SecKeychainSearchCopyNext` instead.

Declared In
`KeychainCore.h`

KCFindGenericPassword

Finds the first generic password in the default keychain matching the specified parameters.

Not recommended

```
OSStatus KCFindGenericPassword (
   ConstStringPtr serviceName,
   ConstStringPtr accountName,
   UInt32 maxLength,
   void *passwordData,
   UInt32 *actualLength,
   KCItemRef *item
);
```

Parameters

`serviceName`

A pointer to a Pascal string containing an application-defined service name. Pass `NULL` to match any service name.

accountName

A pointer to a Pascal string containing an application-defined account name. Pass `NULL` to match any account name.

maxLength

The length of the buffer pointed to by `passwordData`.

passwordData

A pointer to the buffer that holds the returned password data. Before calling the `KCFindGenericPassword` function, allocate enough memory for the buffer to hold the data you want to store. Pass `NULL` if you want to obtain the item reference but not the password data. In this case, you must also pass `NULL` in the `actualLength` parameter. On return, a pointer to the returned password data.

actualLength

On return, the actual length of the password data that was retrieved. If the buffer pointed to by the `passwordData` parameter is smaller than the actual length of the data, the `KCFindGenericPassword` function returns the result code `errKCBufferTooSmall`. In this case, your application must allocate a new buffer of sufficient size before calling the `KCFindGenericPassword` function again.

item

On return, a pointer to a reference to the found item. Pass `NULL` if you don't want to obtain this reference.

Return Value

A result code. See "Keychain Manager Result Codes" (page 1154). The result code `errKCNoDefaultKeychain` indicates that no default keychain was found. The result code `errKCItemNotFound` indicates that no matching password item was found. The result code `errKCBufferTooSmall` indicates that your application must allocate a new buffer of sufficient size before calling the function `KCFindGenericPassword` again.

Discussion

The `KCFindGenericPassword` function finds the first generic password item which matches the attributes you provide. The buffer specified in the `passwordData` parameter must be large enough to hold the password data, otherwise the function `KCFindGenericPassword` returns the result code `errKCBufferTooSmall`. In this case, your application must allocate a new buffer of sufficient size before calling the function `KCFindGenericPassword` again. The `KCFindGenericPassword` function optionally returns a reference to the found item.

The `KCFindGenericPassword` function automatically calls the function `KCUnlock` (page 1127) to display the Unlock Keychain dialog box if the keychain containing the password is currently locked.

You can also call the function `kcfindgenericpassword` to find the first generic password matching specified attributes. The `kcfindgenericpassword` function requires that you pass a pointer to a C string instead of a pointer to a Pascal string for the `serverAddress`, `serverName`, `volumeName`, `accountName`, and `passwordData` parameters.

Version Notes

Available beginning with KeychainLib 1.0. In KeychainLib 1.0, the `kcfindgenericpassword` function provides the same functionality as the function `KCFindGenericPassword`, except that it accepts C strings rather than Pascal strings as arguments. In KeychainLib 2.0, you should use the `KCFindGenericPassword` function, since the `kcfindgenericpassword` function is provided for convenience only and may be removed from the header file at some point in the future.

Availability

Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainFindGenericPassword` function in Keychain Services instead.

Declared In

`KeychainCore.h`

kcfindgenericpassword

Not recommended

```
OSStatus kcfindgenericpassword (
   const char *serviceName,
   const char *accountName,
   UInt32 maxLength,
   void *passwordData,
   UInt32 *actualLength,
   KCItemRef *item
);
```

Discussion

This function is available for convenience only and may be removed. Use the function `KCFindGenericPassword` (page 1100) instead.

Availability

Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainFindGenericPassword` function in Keychain Services instead.

Declared In

`KeychainCore.h`

KCFindInternetPassword

Finds the first Internet password in the default keychain that matches the specified parameters.

Not recommended

```
OSStatus KCFindInternetPassword (
   ConstStringPtr serverName,
   ConstStringPtr securityDomain,
   ConstStringPtr accountName,
   UInt16 port,
   OSType protocol,
   OSType authType,
   UInt32 maxLength,
   void *passwordData,
   UInt32 *actualLength,
   KCItemRef *item
);
```

Parameters

serverName

A pointer to a Pascal string containing the server name. Pass `NULL` to match any server name.

securityDomain

A pointer to a Pascal string containing the security domain. Pass `NULL` to match any domain.

accountName

A pointer to a Pascal string containing the account name. Pass `NULL` to match any account name.

port

The TCP/IP port number. Pass the constant `kAnyPort`, described in "Default Internet Port Constant" (page 1140), to match any port.

protocol

The protocol associated with this password. See "Keychain Protocol Type Constants" (page 1151) for a description of possible values. Pass the constant `kAnyProtocol`, described in "Default Internet Protocol And Authentication Type Constants" (page 1140), to match any protocol.

authType

The authentication scheme used. See "Authentication Type Constants" (page 1136) for a description of possible values. Pass the constant `kAnyAuthType`, described in "Default Internet Protocol And Authentication Type Constants" (page 1140), to match any authentication scheme.

maxLength

The length of the buffer pointed to by `passwordData`.

passwordData

A pointer to the buffer that holds the returned password data. Before calling the `KCFindInternetPassword` function, allocate enough memory for the buffer to hold the data you want to store. Pass `NULL` if you want to obtain the item reference but not the password data. In this case, you must also pass `NULL` in the `actualLength` parameter. On return, a pointer to the returned password data.

actualLength

On return, the actual length of the password data that was retrieved. If the buffer pointed to by the `passwordData` parameter is smaller than the actual length of the data, the `KCFindInternetPassword` function returns the result code `errKCBufferTooSmall`. In this case, your application must allocate a new buffer of sufficient size before calling the `KCFindInternetPassword` function again.

item

On return, a pointer to a reference to the found item. Pass `NULL` if you don't want to obtain this reference.

Return Value
A result code. See "Keychain Manager Result Codes" (page 1154). The result code `errKCNoDefaultKeychain` indicates that no default keychain was found. The result code `errKCItemNotFound` indicates that no matching password item was found. The result code `errKCBufferTooSmall` indicates that your application must allocate a new buffer of sufficient size before calling the function `KCFindInternetPassword` again.

Discussion
The `KCFindInternetPassword` function finds the first Internet password item that matches the attributes you provide. The buffer specified in the `passwordData` parameter must be large enough to hold the password data, otherwise the function `KCFindInternetPassword` returns the result code `errKCBufferTooSmall`. In this case, your application must allocate a new buffer of sufficient size before calling the function `KCFindInternetPassword` again. The `KCFindInternetPassword` function optionally returns a reference to the found item.

The `KCFindInternetPassword` function automatically calls the function `KCUnlock` (page 1127) to display the Unlock Keychain dialog box if the keychain containing the password is currently locked.

You can also call the function `kcfindinternetpassword` to find the first Internet password item matching specified attributes. The `kcfindinternetpassword` function requires that you pass a pointer to a C string instead of a Pascal string for the `serverAddress`, `serverName`, `volumeName`, `accountName`, and `passwordData` parameters.

Version Notes
Available beginning with KeychainLib 1.0. In KeychainLib 1.0, the `kcfindinternetpassword` function provides the same functionality as the function `KCFindInternetPassword`, except that it accepts C strings rather than Pascal strings as arguments. In KeychainLib 2.0, you should use the `KCFindInternetPassword` function, since `kcfindinternetpassword` is provided for convenience only and may be removed from the header file at some point in the future.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes
Use the `SecKeychainFindInternetPassword` function in Keychain Services instead.

Declared In
`KeychainCore.h`

kcfindinternetpassword
Not recommended

```
OSStatus kcfindinternetpassword (
   const char *serverName,
   const char *securityDomain,
   const char *accountName,
   UInt16 port,
   OSType protocol,
   OSType authType,
   UInt32 maxLength,
   void *passwordData,
   UInt32 *actualLength,
   KCItemRef *item
);
```

Discussion
This function is available for convenience only and may be removed. Use the function `KCFindInternetPassword` (page 1102) instead.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainFindInternetPassword` function in Keychain Services instead.

Declared In
`KeychainCore.h`

KCFindInternetPasswordWithPath

Finds the first Internet password in the default keychain that matches the specified parameters, including path information.

Not recommended

```
OSStatus KCFindInternetPasswordWithPath (
   ConstStringPtr serverName,
   ConstStringPtr securityDomain,
   ConstStringPtr accountName,
   ConstStringPtr path,
   UInt16 port,
   OSType protocol,
   OSType authType,
   UInt32 maxLength,
   void *passwordData,
   UInt32 *actualLength,
   KCItemRef *item
);
```

Parameters

serverName
A pointer to a Pascal string containing the server name. Pass `NULL` to match any server name.

securityDomain
A pointer to a Pascal string containing the security domain. Pass `NULL` to match any domain.

accountName

A pointer to a Pascal string containing the account name. Pass `NULL` to match any account name.

path

A pointer to a Pascal string containing additional information that specifies a file or directory on the server specified by the `serverName` parameter. In a typical URL, path information begins directly after the first slash ("/") character following the server name. This parameter is optional.

port

The TCP/IP port number. Pass the constant `kAnyPort`, described in "Default Internet Port Constant" (page 1140), to match any port.

protocol

The protocol associated with this password. See "Keychain Protocol Type Constants" (page 1151) for a description of possible values. Pass the constant `kAnyProtocol`, described in "Default Internet Protocol And Authentication Type Constants" (page 1140), to match any protocol.

authType

The authentication scheme used. See "Authentication Type Constants" (page 1136) for a description of possible values. Pass the constant `kAnyAuthType`, described in "Default Internet Protocol And Authentication Type Constants" (page 1140), to match any authentication scheme.

maxLength

The length of the buffer pointed to by `passwordData`.

passwordData

A pointer to a buffer which will hold the returned password data. Before calling the `KCFindInternetPasswordWithPath` function, allocate enough memory for the buffer to hold the data you want to store. Pass `NULL` if you want to obtain the item reference but not the password data. In this case, you must also pass `NULL` in the `actualLength` parameter. On return, a pointer to the returned password data.

actualLength

On return, the actual length of the password data that was retrieved. If the buffer pointed to by the `passwordData` parameter is smaller than the actual length of the data, the function `KCFindInternetPasswordWithPath` returns the result code `errKCBufferTooSmall`. In this case, your application must allocate a new buffer of sufficient size before calling the function `KCFindInternetPasswordWithPath` again.

item

On return, a pointer to a reference to the found item. Pass `NULL` if you don't want to obtain this reference.

Return Value

A result code. See "Keychain Manager Result Codes" (page 1154). The result code `errKCNoDefaultKeychain` indicates that no default keychain was found. The result code `errKCItemNotFound` indicates that no matching password item was found. The result code `errKCBufferTooSmall` indicates that your application must allocate a new buffer of sufficient size before calling the function `KCFindInternetPasswordWithPath` again.

Discussion

The `KCFindInternetPasswordWithPath` function finds the first Internet password item which matches the attributes you provide, including path information. The buffer specified in the `passwordData` parameter must be large enough to hold the password data, otherwise the function `KCFindInternetPasswordWithPath` returns the result code `errKCBufferTooSmall`. In this case, your application must allocate a new buffer of sufficient size before calling the `KCFindInternetPasswordWithPath` function again. The `KCFindInternetPasswordWithPath` function optionally returns a reference to the found item.

The KCFindInternetPasswordWithPath function automatically calls the function KCUnlock (page 1127) to display the Unlock Keychain dialog box if the keychain containing the password is currently locked.

You can also call the function kcfindinternetpasswordwithpath to find the first Internet password item matching specified attributes. The function kcfindinternetpasswordwithpath requires that you pass a pointer to a C string instead of a pointer to a Pascal string for the serverAddress, serverName, volumeName, accountName, and passwordData parameters.

Version Notes
Available beginning with KeychainLib 2.0. In KeychainLib 1.0, the kcfindinternetpassword function provides the same functionality as the function KCFindInternetPassword, except that it accepts C strings rather than Pascal strings as arguments. In KeychainLib 2.0, you should use KCFindInternetPassword, since kcfindinternetpassword is provided for convenience only and may be removed from the header file at some point in the future.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the SecKeychainFindInternetPassword function in Keychain Services instead.

Declared In
KeychainCore.h

kcfindinternetpasswordwithpath

Not recommended

```
OSStatus kcfindinternetpasswordwithpath (
   const char *serverName,
   const char *securityDomain,
   const char *accountName,
   const char *path,
   UInt16 port,
   OSType protocol,
   OSType authType,
   UInt32 maxLength,
   void *passwordData,
   UInt32 *actualLength,
   KCItemRef *item
);
```

Discussion
This function is available for convenience only and may be removed. Use the function KCFindInternetPasswordWithPath (page 1105) instead.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the SecKeychainFindInternetPassword function in Keychain Services instead.

Declared In
KeychainCore.h

KCFindNextItem

Finds the next keychain item matching the previously specified search criteria.

Not recommended

```
OSStatus KCFindNextItem (
   KCSearchRef search,
   KCItemRef *item
);
```

Parameters

search

A reference to the previously-specified search criteria. Pass the reference passed back in the searchRef parameter of the function KCFindFirstItem (page 1099).

item

On return, a pointer to the next matching keychain item, if any.

Return Value

A result code. See "Keychain Manager Result Codes" (page 1154). The result code errKCNoDefaultKeychain indicates that no default keychain could be found. The result code errKCItemNotFound indicates that no matching keychain item was found. The result code errKCInvalidSearchRef indicates that the specified search reference was invalid.

Discussion

The KCFindNextItem function finds the next keychain item matching the search criteria previously specified by a call to the function KCFindFirstItem (page 1099). The KCFindNextItem function returns a reference to the matching item, if any. The KCFindNextItem function automatically calls the function KCUnlock (page 1127) to display the Unlock Keychain dialog box if the keychain containing the item you are searching for is currently locked.

Special Considerations

When you are completely finished with a search, you should use the functions KCReleaseItem (page 1121) and KCReleaseSearch (page 1122) to release the keychain item and search criteria reference.

Version Notes

Available beginning with KeychainLib 1.0.

Availability

Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the SecKeychainSearchCopyNext function in Keychain Services instead.

Declared In

KeychainCore.h

KCFindX509Certificates

Finds the certificates in a keychain that match specified search criteria.

Unsupported

```
OSStatus KCFindX509Certificates (
    KCRef keychain,
    CFStringRef name,
    CFStringRef emailAddress,
    KCCertSearchOptions options,
    CFMutableArrayRef *certificateItems
);
```

Parameters

keychain
A reference to the keychain you want to search. If the keychain is locked, the Unlock Keychain dialog box is automatically displayed.

name
A pointer to a C string containing the certificate owner's common name.

emailAddress
A pointer to a C string containing the certificate owner's email address.

options
The search criteria you wish to use. See "Certificate Search Options" (page 1137) for a description of this mask.

certificateItems
On return, a pointer to a list of the matching certificates. Pass `NULL` if you don't want to obtain these references.

Return Value

A result code. See "Keychain Manager Result Codes" (page 1154). The result code `errKCNoDefaultKeychain` indicates that a default keychain was not found. The result code `errKCBufferTooSmall` indicates that the certificate data was too large for the supplied buffer. In this case, you should allocate a new buffer of sufficient size before calling `KCFindX509Certificates` again. The result code `errKCItemNotFound` indicates that no matching certificate was found.

Version Notes

Available beginning with KeychainLib 2.0.

Carbon Porting Notes

This function is obsolete. There is currently no replacement.

Declared In

`KeychainHI.h`

KCGetAttribute

Determines keychain item data using a keychain item attribute structure.

Not recommended

```
OSStatus KCGetAttribute (
   KCItemRef item,
   KCAttribute *attr,
   UInt32 *actualLength
);
```

Parameters

item

A reference to the keychain item whose attribute data you wish to determine.

attr

A pointer to a structure of type `KCAttribute` (page 1132). Before calling the `KCGetAttribute` function, fill in the `tag`, `length`, and `data` fields (the `data` field should contain a pointer to a buffer of sufficient length for the type of data to be returned). On return, the `KCGetAttribute` function passes back the requested data in the `data` field.

actualLength

On return, a pointer to the actual length of the attribute data. This may be more than the length you allocated in the length field of the attribute structure.

Return Value

A result code. See "Keychain Manager Result Codes" (page 1154). The result code `errKCInvalidItemRef` indicates that the specified keychain item reference was invalid. The result code `errKCNoSuchAttr` indicates that you tried to set an attribute which is undefined for this item class. The result code `errKCBufferTooSmall` indicates that your application must allocate a new buffer of sufficient size before calling `KCGetAttribute` again.

Discussion

You can call the function `KCGetAttribute` or the function `KCGetData` (page 1111) to obtain keychain item data. The difference between the functions is that the function `KCGetData` (page 1111) requires that you pass the length of the data and a pointer to that data as separate parameters rather than fields in a keychain item attribute structure.

If the keychain that contains the item is locked, before calling the `KCGetAttribute` function you should call the function `KCUnlock` (page 1127) to prompt the user to unlock the keychain.

You can determine any of the standard item attributes identified by the following tag constants: `kClassKCItemAttr`, `kCreationDateKCItemAttr`, `kModDateKCItemAttr`, `kDescriptionKCItemAttr`, `kCommentKCItemAttr`, `kLabelKCItemAttr`, `kCreatorKCItemAttr`, `kScriptCodeKCItemAttr`, and `kCustomIconKCItemAttr`. There is additional data you can determine, depending upon the type of keychain item whose data you wish to obtain. See "Keychain Item Attribute Tag Constants" (page 1144) for more information.

Version Notes

Available beginning with KeychainLib 1.0.

Availability

Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainItemCopyAttributesAndData` function in Keychain Services instead.

Declared In

`KeychainCore.h`

KCGetData

Determines keychain item data.

Not recommended

```
OSStatus KCGetData (
   KCItemRef item,
   UInt32 maxLength,
   void *data,
   UInt32 *actualLength
);
```

Parameters

item

A reference to the keychain item whose data you wish to determine.

maxLength

The length of the data buffer pointed to by the `data` parameter.

data

A pointer to the buffer that holds the returned data. Before calling the `KCGetData` function, allocate enough memory for the buffer to hold the data you want to store. On return, a pointer to the attribute data you requested.

actualLength

On return, a pointer to the actual length of the data being retrieved. If the buffer pointed to by the `data` parameter is smaller than the actual length of the data, the `KCGetData` function returns the result code `errKCBufferTooSmall`. In this case, your application must allocate a new buffer of sufficient size before calling `KCGetData` again.

Return Value

A result code. See "Keychain Manager Result Codes" (page 1154). The result code `errKCInvalidItemRef` indicates that the specified keychain item reference was invalid. The result code `errKCBufferTooSmall` indicates that your application must allocate a new buffer of sufficient size before calling `KCGetData` again. The result code `errKCDataNotModifiable` indicates that the data is not available for this item.

Discussion

You can call the function `KCGetData` or the function `KCGetAttribute` (page 1109) to obtain keychain item data. The difference between the functions is that the function `KCGetAttribute` (page 1109) requires that you pass the length of the data and a pointer to that data as fields in a keychain item attribute structure rather than as separate parameters.

If the keychain that contains the item is locked, before calling the function `KCGetData` you should call the function `KCUnlock` (page 1127) to prompt the user to unlock the keychain. You cannot call the `KCGetData` function for a private key.

You can determine any of the standard item attributes identified by the following tag constants: `kClassKCItemAttr`, `kCreationDateKCItemAttr`, `kModDateKCItemAttr`, `kDescriptionKCItemAttr`, `kCommentKCItemAttr`, `kLabelKCItemAttr`, `kCreatorKCItemAttr`, `kScriptCodeKCItemAttr`, and `kCustomIconKCItemAttr`. There is additional data you can determine, depending upon the type of keychain item whose data you wish to obtain. See "Keychain Item Attribute Tag Constants" (page 1144) for more information.

Version Notes

Available beginning with KeychainLib 1.0.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainItemCopyContent` function in Keychain Services instead.

Declared In
`KeychainCore.h`

KCGetDefaultKeychain

Obtains the default keychain.

Not recommended

```
OSStatus KCGetDefaultKeychain (
   KCRef *keychain
);
```

Parameters

keychain

On return, a pointer to default keychain reference.

Return Value
A result code. See "Keychain Manager Result Codes" (page 1154). The result code `errKCNoDefaultKeychain` indicates that there is no default keychain.

Discussion
You can determine the name of the default keychain by passing the returned keychain reference to the function `KCGetKeychainName` (page 1115).

Version Notes
Available beginning with KeychainLib 2.0.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainCopyDefault` function in Keychain Services instead.

Declared In
`KeychainCore.h`

KCGetIndKeychain

Obtains the reference to an indexed keychain.

Not recommended

```
OSStatus KCGetIndKeychain (
   UInt16 index,
   KCRef *keychain
);
```

Parameters

index

An index of the list of available keychains. Pass a value between 1 and the number returned by the function `KCCountKeychains` (page 1093).

keychain

On return, pointer to the keychain reference corresponding to the index in the `index` parameter.

Return Value

A result code. See "Keychain Manager Result Codes" (page 1154). The result code `errKCNoSuchKeychain` indicates that the index value is out of range.

Discussion

To guarantee correct operation, you should call the function `KCCountKeychains` (page 1093) once before calling `KCGetIndKeychain`.

Special Considerations

The memory that the keychain reference occupies must be released by calling the function `KCReleaseKeychain` (page 1122) when you are finished with it.

Version Notes

Available beginning with KeychainLib 1.0.

Availability

Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainCopySearchList` function in Keychain Services followed by a call to `CFArrayGetValueAtIndex` instead.

Declared In

`KeychainCore.h`

KCGetKeychain

Determines the location of a password or other keychain item.

Not recommended

```
OSStatus KCGetKeychain (
   KCItemRef item,
   KCRef *keychain
);
```

Parameters

item

A reference to the keychain item whose keychain location you wish to determine. If you pass a reference to a keychain item whose keychain is locked, the `KCGetKeychain` function returns the result code `errKCInvalidItemRef`.

keychain
On return, a pointer to the keychain containing the specified item.

Return Value
A result code. See "Keychain Manager Result Codes" (page 1154). The result code `errKCInvalidItemRef` indicates that the keychain item reference was invalid.

Discussion
The `KCGetKeychain` function determines the location of a keychain item in an unlocked keychain. It does not search locked keychains. Calling the `KCGetKeychain` function displays the Unlock Keychain dialog box if the keychain containing the item is currently locked.

Special Considerations

The keychain reference returned by `KCGetKeychain` should be released by calling the function `KCReleaseItem` (page 1121).

Version Notes
Available beginning with KeychainLib 1.0.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainItemCopyKeychain` function in Keychain Services instead.

Declared In
`KeychainCore.h`

KCGetKeychainManagerVersion

Determines the version of the Keychain Manager installed on the user's system.

Not Recommended

```
OSStatus KCGetKeychainManagerVersion (
   UInt32 *returnVers
);
```

Parameters
returnVers
On return, a pointer to the version number of the Keychain Manager installed on the current system.

Return Value
A result code. See "Keychain Manager Result Codes" (page 1154).

Discussion
Your application can call the `KCGetKeychainManagerVersion` function to find out which version of the Keychain Manager is installed on the user's system.

Version Notes
Available beginning with KeychainLib 1.0.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainGetVersion` function in Keychain Services instead.

Declared In

`KeychainCore.h`

KCGetKeychainName

Determines the name of a keychain.

Not recommended

```
OSStatus KCGetKeychainName (
   KCRef keychain,
   StringPtr keychainName
);
```

Parameters

keychain

A reference to the keychain whose name you wish to obtain.

keychainName

A pointer to a Pascal string. On return, this string contains the name of the keychain.

Return Value

A result code. See "Keychain Manager Result Codes" (page 1154). The result code `errKCInvalidKeychain` indicates that the keychain is invalid.

Discussion

You can also call the function `kcgetkeychainname` to obtain the name of a keychain. `kcgetkeychainname` requires that you pass a pointer to a C string instead of a pointer to a Pascal string in the `keychainName` parameter.

Version Notes

Available beginning with KeychainLib 2.0.

Availability

Available in CarbonLib 1.1 and later when KeychainLib 2.0 or later is present.

Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainGetPath` function in Keychain Services instead.

Declared In

`KeychainCore.h`

kcgetkeychainname

Not recommended

```
OSStatus kcgetkeychainname (
   KCRef keychain,
   char *keychainName
);
```

Discussion
This function is available for convenience only and may be removed. Use the function `KCGetKeychainName` (page 1115) instead.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainGetPath` function in Keychain Services instead.

Declared In
`KeychainCore.h`

KCGetStatus

Determines the permissions that are set in a keychain.

Not recommended

```
OSStatus KCGetStatus (
   KCRef keychain,
   UInt32 *keychainStatus
);
```

Parameters

keychain

A pointer to the keychain reference whose permissions you wish to determine. Pass `NULL` to obtain the status of the default keychain.

keychainStatus

On return, a pointer to a bitmask that you can test to determine the permissions that are set in a keychain. See "Keychain Status Constants" (page 1153) for a description of this mask.

Return Value
A result code. See "Keychain Manager Result Codes" (page 1154). The result code `errKCNoSuchKeychain` indicates that the specified keychain could not be found. The result code `errKCInvalidKeychain` indicates that the specified keychain is invalid.

Version Notes
Available beginning with KeychainLib 1.0.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainGetStatus` function in Keychain Services instead.

Declared In
KeychainCore.h

KCIsInteractionAllowed

Indicates whether Keychain Manager functions that display a user interaction will do so.

Not recommended

```
Boolean KCIsInteractionAllowed (
   void
);
```

Parameters

Return Value
A Boolean value indicating whether user interaction is permitted. If true, user interaction is allowed, and Keychain Manager functions that display a user interface can do so as appropriate.

Version Notes
Available beginning with KeychainLib 2.0.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the SecKeychainGetUserInteractionAllowed function in Keychain Services instead.

Declared In
KeychainCore.h

KCLock

Locks a keychain.

Not recommended

```
OSStatus KCLock (
   KCRef keychain
);
```

Parameters

keychain

A reference to the keychain to lock. Pass NULL to lock all unlocked keychains.

Return Value
A result code. See "Keychain Manager Result Codes" (page 1154). The result code errKCNoSuchKeychain indicates that specified keychain could not be found. The result code errKCInvalidKeychain indicates that the specified keychain is invalid.

Discussion
Your application should not call the KCLock function unless you are responding to a user's request to lock a keychain. In general, you should leave the keychain unlocked so that the user does not have to unlock it again in another application.

Version Notes
The function `KCLock` replaces the function `KCLockKeychain`, which was available in KeychainLib 1.0.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainLock` function in Keychain Services instead.

Declared In
`KeychainCore.h`

KCMakeAliasFromKCRef

Creates an alias to a keychain reference.

Not Recommended

```
OSStatus KCMakeAliasFromKCRef (
   KCRef keychain,
   AliasHandle *keychainAlias
);
```

Parameters

keychain

A reference to the keychain for which you want to create an alias.

keychainAlias

On return, a pointer to an alias handle to the file referred to by the keychain reference.

Return Value
A result code. See "Keychain Manager Result Codes" (page 1154).

Discussion
You may wish to call the `KCMakeAliasFromKCRef` function to determine the location of a keychain.

Special Considerations

When you are finished with a keychain, you should call the function `KCReleaseKeychain` (page 1122) to deallocate its memory. You should not use the keychain after its memory has been deallocated.

Version Notes
Available beginning with KeychainLib 2.0.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainGetPath` function in Keychain Services followed by calls to the function `FSPathMakeRef` and `FSNewAlias` instead.

Declared In
`KeychainCore.h`

KCMakeKCRefFromAlias

Creates a keychain reference from a keychain alias.

Not Recommended

```
OSStatus KCMakeKCRefFromAlias (
   AliasHandle keychainAlias,
   KCRef *keychain
);
```

Parameters

keychainAlias

A handle to an alias record of the keychain file. Since the keychain is a file, an alias can be made to the keychain file.

keychain

On return, a pointer to a reference to the keychain specified by the alias in the `keychainAlias` parameter.

Return Value

A result code. See "Keychain Manager Result Codes" (page 1154).

Special Considerations

When you are finished with a keychain, you should call the function `KCReleaseKeychain` (page 1122) to deallocate its memory. You should not use the keychain after its memory has been deallocated.

Version Notes

Available beginning with KeychainLib 2.0.

Availability

Available in CarbonLib 1.1 and later when KeychainLib 2.0 or later is present.

Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainOpen` function in Keychain Services instead. If the keychain doesn't exist, use the `SecKeychainCreate` function in Keychain Services.

Declared In

`KeychainCore.h`

KCMakeKCRefFromFSSpec

Creates a keychain reference from a file specification record. (Deprecated in Mac OS X v10.5.)

Not Recommended

```
OSStatus KCMakeKCRefFromFSSpec (
   FSSpec *keychainFSSpec,
   KCRef *keychain
);
```

Parameters

keychainFSSpec

A pointer to a keychain file specification record.

keychain

On return, a pointer to a reference to the keychain specified by the file in the `keychainFSSpec` parameter.

Return Value
A result code. See "Keychain Manager Result Codes" (page 1154).

Special Considerations

When you are finished with a keychain, you should call the function `KCReleaseKeychain` (page 1122) to deallocate its memory. You should not use the keychain after its memory has been deallocated.

Version Notes
Available beginning with KeychainLib 2.0.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 2.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Carbon Porting Notes

Use the `SecKeychainOpen` function in Keychain Services instead. If the keychain doesn't exist, use the `SecKeychainCreate` function in Keychain Services.

Declared In
`KeychainCore.h`

KCNewItem

Creates a reference to a keychain item.

Not recommended

```
OSStatus KCNewItem (
   KCItemClass itemClass,
   OSType itemCreator,
   UInt32 length,
   const void *data,
   KCItemRef *item
);
```

Parameters

itemClass

The type of keychain item that you wish to create. See "Keychain Item Type Constants" (page 1150) for a description of possible values and a description of the `KCItemClass` data type.

itemCreator

The creator code of the application that owns this item.

length

The length of the data to be stored in this item.

data

A pointer to a buffer containing the data to be stored in this item. Before calling `KCNewItem`, allocate enough memory for the buffer to hold the data you want to store.

item

On return, a pointer to a reference to the newly-created item.

Return Value
A result code. See "Keychain Manager Result Codes" (page 1154). The Memory Manager result code `memFullErr` indicates that you did not allocate enough memory in the current heap to create the item.

Discussion
After calling the `KCNewItem` function, you should call the function `KCAddItem` (page 1090) if you wish to permanently store a password or other keychain item. Note that a copy of the data buffer pointed to by the `data` parameter is stored in the newly-created item.

Special Considerations

When you are done with a keychain item, you should call the function `KCReleaseItem` (page 1121) to release its memory. You should not use the item after its memory has been deallocated.

Version Notes
Available beginning with KeychainLib 1.0.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainItemCreateFromContent` function in Keychain Services instead.

Declared In
`KeychainCore.h`

KCReleaseItem

Disposes of the memory occupied by a keychain item reference.

Not recommended

```
OSStatus KCReleaseItem (
   KCItemRef *item
);
```

Parameters

item

A pointer to a keychain item reference. Pass the keychain item reference whose memory you want to release. On return, the reference is set to `NULL` and should not be used again.

Return Value
A result code. See "Keychain Manager Result Codes" (page 1154).

Discussion
You should call the `KCReleaseItem` function to release the memory occupied by a keychain item reference when you are finished with it.

Version Notes
Available beginning with KeychainLib 1.0

Availability
Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `CFRelease` function instead.

Declared In
`KeychainCore.h`

KCReleaseKeychain

Disposes of the memory associated with a keychain reference.

Not recommended

```
OSStatus KCReleaseKeychain (
   KCRef *keychain
);
```

Parameters

keychain

A pointer to a keychain reference. Pass the keychain reference whose memory you want to release. On return, the reference is set to `NULL` and should not be used again.

Return Value
A result code. See "Keychain Manager Result Codes" (page 1154).

Discussion
You should call the `KCReleaseKeychain` function to release the memory occupied by a keychain reference when you are finished with it. You should not use the reference after it has been released.

Version Notes
Available beginning with KeychainLib 2.0.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `CFRelease` function instead.

Declared In
`KeychainCore.h`

KCReleaseSearch

Disposes of the memory occupied by a search criteria reference.

Not recommended

```
OSStatus KCReleaseSearch (
   KCSearchRef *search
);
```

Parameters

search

A pointer to a search criteria reference. Pass the search criteria reference whose memory you want to release. On return, the reference is set to `NULL` and should not be used again.

Return Value

A result code. See "Keychain Manager Result Codes" (page 1154). The result code `errKCInvalidSearchRef` indicates that the specified search reference was invalid.

Discussion

You should call the `KCReleaseSearch` function to release the memory occupied by a search criteria reference when you are completely finished with a search performed by calling the functions `KCFindFirstItem` (page 1099) or `KCFindNextItem` (page 1108).

Version Notes

Available beginning with KeychainLib 1.0

Availability

Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `CFRelease` function instead.

Declared In

`KeychainCore.h`

KCRemoveCallback

Unregisters your keychain event callback function.

Not recommended

```
OSStatus KCRemoveCallback (
   KCCallbackUPP callbackProc
);
```

Parameters

callbackProc

A Universal Procedure Pointer (UPP) to your keychain event callback function that was previously registered with the function `KCAddCallback` (page 1083).

Return Value

A result code. See "Keychain Manager Result Codes" (page 1154). The result code `errKCInvalidCallback` indicates that the callback function was not previously registered.

Discussion

After you pass a UPP to your keychain event callback function to the `KCRemoveCallback` function, it will no longer be called by the Keychain Manager.

Special Considerations

After calling `KCRemoveCallback`, you should call the function `DisposeKCCallbackUPP` (page 1079) to dispose of the UPP to your callback function.

Version Notes

Available beginning with KeychainLib 1.0.

Availability

Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainRemoveCalllback` function in Keychain Services instead.

Declared In

`KeychainCore.h`

KCSetAttribute

Sets or edits keychain item data using a keychain item attribute structure.

Not recommended

```
OSStatus KCSetAttribute (
   KCItemRef item,
   KCAttribute *attr
);
```

Parameters

item

A reference to the keychain item whose data you wish to set or edit.

attr

A pointer to a structure of type `KCAttribute` (page 1132) containing keychain item data you want to set. Before calling the function `KCSetAttribute`, fill in the `tag`, `length`, and `data` fields of this structure with the tag identifying the attribute you wish to modify or set, the length of the attribute data you wish to set, and a pointer to that data, respectively.

Return Value

A result code. See "Keychain Manager Result Codes" (page 1154). The result code `errKCInvalidItemRef` indicates that the keychain item reference was invalid. The result code `errKCNoSuchAttr` indicates that the item attribute you wish to set is undefined for the specified item. The result code `errKCDataTooLarge` indicates that more data was supplied than is allowed for this attribute.

Discussion

You can call the `KCSetAttribute` function or the function `KCSetData` (page 1125) to set or modify keychain item data. The difference between the functions is that the `KCSetData` (page 1125) function requires that you pass the length of the data and a pointer to that data as separate parameters rather than fields in a keychain item attribute structure.

If the keychain that contains the item is locked, before calling the `KCSetAttribute` function you should call the function `KCUnlock` (page 1127) to prompt the user to unlock the keychain. The keychain must permit read/write access in order to modify keychain item data.

You can only set or modify standard item attributes identified by the tag constants `kDescriptionKCItemAttr`, `kCommentKCItemAttr`, `kLabelKCItemAttr`, `kCreatorKCItemAttr`, `kTypeKCItemAttr`, and `kCustomIconKCItemAttr`. In addition, each class of keychain item has attributes specific to that class which may be set or modified. See "Keychain Item Attribute Tag Constants" (page 1144) for more information.

Version Notes
Available beginning with KeychainLib 1.0.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainItemModifyAttributesAndData` function in Keychain Services instead.

Declared In
`KeychainCore.h`

KCSetData

Sets or edits keychain item data.

Not recommended

```
OSStatus KCSetData (
   KCItemRef item,
   UInt32 length,
   const void *data
);
```

Parameters

item
A reference to the keychain item whose data you wish to set.

length
The length of the data buffer pointed to by the `data` parameter.

data
A pointer to a buffer containing the data to be stored in this item. Before calling the `KCSetData` function, allocate enough memory for the buffer to hold the data you want to store.

Return Value
A result code. See "Keychain Manager Result Codes" (page 1154). The result code `errKCInvalidItemRef` indicates that the specified keychain item reference was invalid. The result code `errKCDataTooLarge` indicates that the data was too large for the supplied buffer. The result code `errKCDataNotModifiable` indicates that the data cannot be set for this item.

Discussion
You can call the function `KCSetData` or the function `KCSetAttribute` (page 1124) to set or modify keychain item data. The difference between the functions is that the function `KCSetAttribute` (page 1124) requires that you pass the length of the data buffer as a field in a keychain item attribute structure rather than as a separate parameter.

If the keychain that contains the item is locked, before calling the `KCSetData` function you should call the function `KCUnlock` (page 1127) to prompt the user to unlock the keychain. The keychain must permit read/write access in order to modify keychain item data.

You can set or edit any of the standard item attributes identified by the following tag constants: `kDescriptionKCItemAttr`, `kCommentKCItemAttr`, `kLabelKCItemAttr`, `kCreatorKCItemAttr`, `kTypeKCItemAttr`, and `kCustomIconKCItemAttr`. There is additional data you can set, depending upon the type of keychain item whose data you are manipulating. See "Keychain Item Attribute Tag Constants" (page 1144) for more information.

Version Notes
Available beginning with KeychainLib 1.0.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainItemModifyContent` function in Keychain Services instead.

Declared In
`KeychainCore.h`

KCSetDefaultKeychain

Sets the default keychain.

Not recommended

```
OSStatus KCSetDefaultKeychain (
   KCRef keychain
);
```

Parameters

keychain

A reference to the keychain you wish to make the default.

Return Value
A result code. See "Keychain Manager Result Codes" (page 1154). The result code `errKCNoSuchKeychain` indicates that the specified keychain could not be found. The result code `errKCInvalidKeychain` indicates that the specified keychain is invalid.

Discussion
In most cases, your application should not need to set the default keychain, because this is a choice normally made by the user. You should call the `KCSetDefaultKeychain` function to change where a password or other keychain items are added.

The `KCSetDefaultKeychain` function sets the default keychain regardless of whether the keychain is currently locked.

Version Notes
Available beginning with KeychainLib 2.0.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 2.0 or later is present.

Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainSetDefault` function in Keychain Services instead.

Declared In

`KeychainCore.h`

KCSetInteractionAllowed

Enables or disables Keychain Manager functions that display a user interface.

Not recommended

```
OSStatus KCSetInteractionAllowed (
   Boolean state
);
```

Parameters

state

A flag that indicates whether the Keychain Manager will display a user interface. If you pass `true`, user interaction is allowed. This is the default value. If `false`, Keychain Manager functions that normally display a user interface will instead return an error.

Return Value

A result code. See "Keychain Manager Result Codes" (page 1154).

Discussion

The `KCSetInteractionAllowed` function enables you to control whether the functions `KCLock` (page 1117), `KCUnlock` (page 1127), and `KCChangeSettings` (page 1091) display a user interface. Note that failure to re-enable user interaction will affect other clients of the Keychain Manager. By default, user interaction is permitted.

Version Notes

Available beginning with KeychainLib 2.0.

Availability

Available in CarbonLib 1.1 and later when KeychainLib 2.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainSetUserInteractionAllowed` function in Keychain Services instead.

Declared In

`KeychainCore.h`

KCUnlock

Displays a dialog box that prompts the user for a password before unlocking a keychain.

Not recommended

```
OSStatus KCUnlock (
   KCRef keychain,
   StringPtr password
);
```

Parameters

keychain

A reference to the keychain to unlock. Pass `NULL` to specify the default keychain. If you pass `NULL` and the default keychain is currently locked, the keychain will appear as the default choice. If you pass a locked keychain, the function `KCUnlock` displays the Unlock Keychain dialog box and the keychain appears as the chosen menu item in the keychain popup menu. If the default keychain is currently unlocked, the Unlock Keychain dialog box is not displayed and the `KCUnlock` function returns `noErr`.

password

A pointer to a Pascal string representing the password string for this keychain. Pass `NULL` if the user password is unknown. In this case, The `KCUnlock` function displays the Unlock Keychain dialog box, and the authentication user interface associated with the keychain about to be unlocked. If you specify an invalid password, you will not be able to unlock the keychain with a specified password until the machine is rebooted. In this case, the `KCUnlock` function returns `errKCInteractionRequired`.

Return Value

A result code. See "Keychain Manager Result Codes" (page 1154). The result code `noErr` does not guarantee that the specified keychain is unlocked, because the user can select any available keychain and unlock it. The result code `userCanceledErr` indicates that the user pressed the Cancel button in the Unlock Keychain dialog box. The result code `errKCAuthFailed` indicates that authentication failed because of too many unsuccessful retries. The result code `errKCInteractionRequired` indicates that user interaction is required to unlock the keychain. In this case, you will not be able to unlock the keychain with that password until the machine is rebooted.

Discussion

In most cases, your application does not need to call the `KCUnlock` function directly, since most Keychain Manager functions that require an unlocked keychain call `KCUnlock` automatically. If your application needs to verify that a keychain is unlocked, call the function `KCGetStatus` (page 1116).

You can also call the function `kcunlock` to display a user interface prompting the user to unlock a keychain. The `kcunlock` function requires that you pass a pointer to a C string instead of a pointer to a Pascal string in the `password` parameter.

Special Considerations

It is recommended that the `KCUnlock` function not be explicitly called by applications. Most functions that require an unlocked keychain call the `KCUnlock` function for you.

The memory that the keychain reference occupies must be released by calling the function `KCReleaseKeychain` (page 1122) when you are finished with it.

Version Notes

The `KCUnlock` function replaces the function `KCUnlockKeychain`, which was available in KeychainLib 1.0.

Availability

Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainUnlock` function in Keychain Services instead.

Declared In
KeychainHI.h

kcunlock

Not recommended

```
OSStatus kcunlock (
   KCRef keychain,
   const char *password
);
```

Discussion
This function is available for convenience only and may be removed. Use the function KCUnlock (page 1127) instead.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the SecKeychainUnlock function in Keychain Services instead.

Declared In
KeychainHI.h

KCUpdateItem

Updates a password or other keychain item.

Not recommended

```
OSStatus KCUpdateItem (
   KCItemRef item
);
```

Parameters

item

A reference to the keychain item whose data you wish to update. If you pass an item that has not been previously added to the keychain, the KCUpdateItem function does nothing and returns noErr.

Return Value
A result code. See "Keychain Manager Result Codes" (page 1154). The result code errKCNoDefaultKeychain indicates that no default keychain could be found. The result code errKCInvalidItemRef indicates that the specified keychain item reference was invalid.

Discussion
You can use the KCUpdateItem function to update a password or other keychain item in a keychain's permanent data store after changing its data. The function KCUpdateItem automatically calls the function KCUnlock (page 1127) to display the Unlock Keychain dialog box if the keychain containing the item is currently locked.

Version Notes
Available beginning with KeychainLib 1.0.

Availability
Available in CarbonLib 1.1 and later when KeychainLib 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

Use the `SecKeychainItemModifyContent` function in Keychain Services instead.

Declared In
`KeychainCore.h`

NewKCCallbackUPP

Creates a UPP to your keychain event callback.

Not recommended

```
KCCallbackUPP NewKCCallbackUPP (
   KCCallbackProcPtr userRoutine
);
```

Parameters

`userRoutine`

A pointer to your keychain event callback function. For information on how to create a keychain event callback, see `KCCallbackProcPtr` (page 1131).

Return Value
A UPP to your callback function. You can register your callback function by passing this UPP in the `callbackProc` parameter of the function `KCAddCallback` (page 1083). See the description of the `KCCallbackUPP` data type.

Discussion
The `NewKCCallbackUPP` function creates a pointer to your keychain event callback function. You pass a pointer to your callback function in the `callbackProc` parameter of the function `KCAddCallback` (page 1083) if you want your application to receive data transfer events.

Special Considerations

When you are finished with a UPP to your keychain event callback function, you should dispose of it by calling the function `DisposeKCCallbackUPP` (page 1079).

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

There is no replacement function available.

Declared In
`KeychainCore.h`

Callbacks

KCCallbackProcPtr

Defines a pointer to your keychain event callback that handles user keychain access events.

```
typedef OSStatus (*KCCallbackProcPtr)
(
        KCEvent keychainEvent,
        KCCallbackInfo * info,
        void * userContext
);
```

If you name your function `MyKCCallbackProc`, you would declare it like this:

```
OSStatus MyKCCallbackProc (
        KCEvent keychainEvent,
        KCCallbackInfo * info,
        void * userContext
);
```

Parameters

keychainEvent

The keychain event that your application wishes to be notified of. See "Keychain Events Constants" (page 1140) for a description of possible values. The type of event that can trigger your callback depends on the bitmask you passed in the `eventMask` parameter of the function `KCAddCallback` (page 1083). For more information, see the discussion.

info

A pointer to a structure of type `KCCallbackInfo` (page 1133). On return, the structure contains information about the keychain event that occurred. The Keychain Manager passes this information to your callback function via the `info` parameter of the function `InvokeKCCallbackUPP` (page 1079).

userContext

A pointer to application-defined storage that your application previously passed to the function `KCAddCallback` (page 1083). You can use this value to perform operations such as tracking which instance of a function is operating.

Return Value

A result code. See "Keychain Manager Result Codes" (page 1154). Your keychain event callback function should process the keychain event and return `noErr`.

Discussion

Your keychain event callback function handles those keychain events that you indicate. In order to be notified of these events, you must pass a UPP to your notification callback function in the `callbackProc` parameter of `KCAddCallback` (page 1083). You indicate the type of data transfer events you want to receive via a bitmask in the `eventMask` parameter. When you no longer wish to receive notification of keychain events, you should call the function `KCRemoveCallback` (page 1123) to dispose of the UPP to your keychain event callback function.

Carbon Porting Notes

Use the `SecKeychainCallback` function in Keychain Services instead.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`KeychainCore.h`

Data Types

AFPServerSignature

Represents a 16-byte Apple File Protocol server signature block.

```
typedef UInt8 AFPServerSignature[16];
```

Discussion
The `AFPServerSignature` type represents a 16-byte Apple File Protocol server signature block. You can pass a value of this type in the serverSignature parameter of the functions `KCAddAppleSharePassword` (page 1080) and `KCFindAppleSharePassword` (page 1096) to represent an Apple File Protocol server signature. You can use a value of this type with the keychain item attribute constant `kSignatureKCItemAttr` to specify an Apple File Protocol server signature.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`KeychainCore.h`

KCAttribute

Contains information about a keychain item attribute.

```
typedef SecKeychainAttribute KCAttribute;
```

Discussion
The `KCAttribute` type represents a structure containing information about the attribute of a keychain item. It contains a tag that identifies a particular keychain item attribute value, the length of the attribute value, and a pointer to the attribute value. You can modify attribute data for a keychain item attribute by passing a pointer to this structure in the `attr` parameter of the function `KCSetAttribute` (page 1124). The function `KCGetAttribute` (page 1109) passes back a pointer to this structure in the `attr` parameter.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`KeychainCore.h`

KCAttributeList

Lists attributes in a keychain item.

```
typedef SecKeychainAttributeList KCAttributeList;
```

Discussion
The `KCAttributeList` type represents a list of structures containing information about the attributes in a keychain item. You pass a pointer to this list of 0 or more structures in the `attrList` parameter of the function `KCFindFirstItem` (page 1099) to indicate the attributes to be matched.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`KeychainCore.h`

KCAttrType

Identifies a keychain item attribute value.

```
typedef SecKeychainAttrType KCAttrType;
```

Discussion
The `KCAttrType` type represents a tag that identifies a keychain item attribute value. You can use this value in the `tag` field of the structure `KCAttribute` (page 1132) to identify the keychain item attribute value you wish to set or obtain. See `Keychain Item Attribute Tag Constants` (page 1144) for a description of the Apple-defined tag constants and the data types of the values they identify. Your application can create application-defined tags of type `KCAttrType`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`KeychainCore.h`

KCCallbackInfo

Contains information about a keychain event.

```
struct KCCallbackInfo {
    UInt32 version;
    KCItemRef item;
    long processID[2];
    long event[4];
    KCRef keychain;
};
typedef struct KCCallbackInfo KCCallbackInfo;
```

Fields

`version`
The version of this structure.

`item`
A reference to the keychain item in which the event occurred. If the event did not involve an item, this field is not valid.

`processID`
: A 64-bit quantity containing the process serial number of the process in which the event occurred. This is not available on Mac OS X.

`event`
: The keychain event that occurred. If the event is a system event as indicated by the constant `kSystemKCEvent`, the Keychain client can process events. If the event is not a system event, this field is not valid. This is not available on Mac OS X.

`keychain`
: A reference to the keychain in which the event occurred. If the event did not involve a keychain, this field is not valid.

Discussion
The `KCCallbackInfo` type represents a structure that contains information about the keychain event of which your application wants to be notified. The Keychain Manager passes a pointer to this structure in the `info` parameter of your callback function via the function `InvokeKCCallbackUPP` (page 1079), which invokes your callback function. For information on how to write a keychain event callback function, see `KCCallbackProcPtr` (page 1131).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`KeychainCore.h`

KCCallbackUPP

Defines a data type for the `KCCallbackProcPtr` callback pointer.

```
typedef KCCallbackProcPtr KCCallbackUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`KeychainCore.h`

KCItemRef

Represents a reference to a keychain item.

```
typedef SecKeychainItemRef KCItemRef;
```

Discussion
The `KCItemRef` type represents a reference to an opaque structure that identifies a keychain item. You should call the function `KCNewItem` (page 1120) to create a keychain item reference. The function `KCReleaseItem` (page 1121) disposes of a keychain item reference when no longer needed.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`KeychainCore.h`

KCPublicKeyHash

Represents a 20-byte public key hash.

```
typedef UInt8 KCPublicKeyHash[20];
```

Discussion
The `KCPublicKeyHash` type represents a hash of a public key. You can use the constant `kPublicKeyHashKCItemAttr`, described in `Keychain Item Attribute Tag Constants` (page 1144), to set or retrieve a certificate attribute value of this type.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`KeychainCore.h`

KCRef

Represents a reference to a keychain.

```
typedef SecKeychainRef KCRef;
```

Discussion
The `KCRef` type represents a reference to an opaque structure that identifies a keychain. You should call the function `KCMakeKCRefFromFSSpec` (page 1119) or `KCMakeKCRefFromAlias` (page 1119) to create a keychain reference. The function `KCReleaseKeychain` (page 1122) disposes of a keychain reference when no longer needed. You pass a reference of this type to Keychain Manager functions that operate on a keychain in some way.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`KeychainCore.h`

KCSearchRef

Represents a reference to the current search criteria.

```
typedef SecKeychainSearchRef KCSearchRef;
```

Discussion
The `KCSearchRef` type represents a reference to an opaque structure that identifies the current search criteria. The function `KCFindFirstItem` (page 1099) passes back a reference of this type in the `search` parameter for subsequent calls to the function `KCFindNextItem` (page 1108). You must release this reference when you are finished with a search by calling the function `KCReleaseSearch` (page 1122).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`KeychainCore.h`

KCStatus

Identifies a mask that you can use in determining the permissions that are set in a keychain.

```
typedef SecKeychainStatus KCStatus;
```

Discussion
The `KCStatus` enumeration defines masks your application can use to determine the read and write permissions for a keychain. The function `KCGetStatus` (page 1116) passes back this mask in the `status` parameter.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`KeychainCore.h`

Constants

Authentication Type Constants

Represent the type of authentication to use in storing and retrieving Internet passwords.

```
enum {
    kKCAuthTypeNTLM = 'ntlm',
    kKCAuthTypeMSN = 'msna',
    kKCAuthTypeDPA = 'dpaa',
    kKCAuthTypeRPA = 'rpaa',
    kKCAuthTypeHTTPDigest = 'httd',
    kKCAuthTypeDefault = 'dflt'
};
typedef FourCharCode KCAuthType;
```

Constants

`kKCAuthTypeNTLM`
Specifies Windows NT LAN Manager authentication.
Available in Mac OS X v10.0 and later.
Declared in `KeychainCore.h`.

`kKCAuthTypeMSN`
Specifies Microsoft Network authentication.
Available in Mac OS X v10.0 and later.
Declared in `KeychainCore.h`.

`kKCAuthTypeDPA`
Specifies Distributed Password authentication.
Available in Mac OS X v10.0 and later.
Declared in `KeychainCore.h`.

`kKCAuthTypeRPA`
Specifies Remote Password authentication.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kKCAuthTypeHTTPDigest`
Specifies HTTP Digest Access authentication.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kKCAuthTypeDefault`
Specifies default authentication.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

Discussion
The `KCAuthType` enumeration defines constants you can use to identify the type of authentication to use in storing and retrieving Internet passwords. You can pass a constant of this type in the `authType` parameter of the functions `KCAddInternetPassword` (page 1086), `KCAddInternetPasswordWithPath` (page 1088), `KCFindInternetPassword` (page 1102), and `KCFindInternetPasswordWithPath` (page 1105).

Certificate Search Options

Represent a mask that specifies the search criteria to use when finding certificates.

```
typedef UInt32 KCCertSearchOptions;
enum {
    kCertSearchShift = 0,
    kCertSearchSigningIgnored = 0,
    kCertSearchSigningAllowed = 1 << (kCertSearchShift + 0),
    kCertSearchSigningDisallowed = 1 << (kCertSearchShift + 1),
    kCertSearchSigningMask = ((kCertSearchSigningAllowed) |
        (kCertSearchSigningDisallowed)),
    kCertSearchVerifyIgnored = 0,
    kCertSearchVerifyAllowed = 1 << (kCertSearchShift + 2),
    kCertSearchVerifyDisallowed = 1 << (kCertSearchShift + 3),
    kCertSearchVerifyMask = ((kCertSearchVerifyAllowed) |
        (kCertSearchVerifyDisallowed)),
    kCertSearchEncryptIgnored = 0,
    kCertSearchEncryptAllowed = 1 << (kCertSearchShift + 4),
    kCertSearchEncryptDisallowed = 1 << (kCertSearchShift + 5),
    kCertSearchEncryptMask = ((kCertSearchEncryptAllowed) |
        (kCertSearchEncryptDisallowed)),
    kCertSearchDecryptIgnored = 0,
    kCertSearchDecryptAllowed = 1 << (kCertSearchShift + 6),
    kCertSearchDecryptDisallowed = 1 << (kCertSearchShift + 7),
    kCertSearchDecryptMask = ((kCertSearchDecryptAllowed) |
        (kCertSearchDecryptDisallowed)),
    kCertSearchWrapIgnored = 0,
    kCertSearchWrapAllowed = 1 << (kCertSearchShift + 8),
    kCertSearchWrapDisallowed = 1 << (kCertSearchShift + 9),
    kCertSearchWrapMask = ((kCertSearchWrapAllowed) |
        (kCertSearchWrapDisallowed)),
    kCertSearchUnwrapIgnored = 0,
    kCertSearchUnwrapAllowed = 1 << (kCertSearchShift + 10),
    kCertSearchUnwrapDisallowed = 1 << (kCertSearchShift + 11),
    kCertSearchUnwrapMask = ((kCertSearchUnwrapAllowed) |
        (kCertSearchUnwrapDisallowed)),
    kCertSearchPrivKeyRequired = 1 << (kCertSearchShift + 12),
    kCertSearchAny = 0
};
```

Discussion
The `KCCertSearchOptions` enumeration defines masks that you can use in the `options` parameter of the function `KCFindX509Certificates` (page 1109).

Certificate Usage Options

Represent a mask that specifies the usage options when adding certificates.

```
typedef UInt32 KCCertAddOptions;
enum {
    kSecOptionReserved = 0x000000FF,
    kCertUsageShift = 8,
    kCertUsageSigningAdd         = 1 << (kCertUsageShift + 0),
    kCertUsageSigningAskAndAdd = 1 << (kCertUsageShift + 1),
```

```
    kCertUsageVerifyAdd = 1 << (kCertUsageShift + 2),
    kCertUsageVerifyAskAndAdd = 1 << (kCertUsageShift + 3),
    kCertUsageEncryptAdd = 1 <<(kCertUsageShift + 4),
    kCertUsageEncryptAskAndAdd = 1 << (kCertUsageShift + 5),
    kCertUsageDecryptAdd = 1 << (kCertUsageShift + 6),
    kCertUsageDecryptAskAndAdd = 1 << (kCertUsageShift + 7),
    kCertUsageKeyExchAdd = 1 << (kCertUsageShift + 8),
    kCertUsageKeyExchAskAndAdd = 1 << (kCertUsageShift + 9),
    kCertUsageRootAdd = 1 << (kCertUsageShift + 10),
    kCertUsageRootAskAndAdd = 1 << (kCertUsageShift + 11),
    kCertUsageSSLAdd = 1 << (kCertUsageShift + 12),
    kCertUsageSSLAskAndAdd = 1 << (kCertUsageShift + 13),
    kCertUsageAllAdd = 0x7FFFFF00
};
```

Certificate Verification Criteria

Identify the verification criteria for use when displaying certificates to the user.

```
typedef UInt16 KCVerifyStopOn;
enum {
    kPolicyKCStopOn = 0,
    kNoneKCStopOn = 1,
    kFirstPassKCStopOn = 2,
    kFirstFailKCStopOn = 3
};
```

Constants

`kPolicyKCStopOn`
Indicates that the function `KCChooseCertificate` (page 1092) should use the trust policy options currently in effect.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kNoneKCStopOn`
Indicates that the function `KCChooseCertificate` (page 1092) completes after examining all available certificates.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kFirstPassKCStopOn`
Indicates that the function `KCChooseCertificate` (page 1092) when one certificate meeting the verification criteria is found.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kFirstFailKCStopOn`
Specifies that the function `KCChooseCertificate` (page 1092) completes when one certificate that fails to meet the verification criteria is found.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

Discussion
The `KCVerifyStopOn` enumeration defines constants your application can use to identify the verification criteria to use in selecting certificates. You can pass a constant of this type in the `stopOn` parameter of the function `KCChooseCertificate` (page 1092).

Default Internet Port Constant

Represent the internet ports available.

```
enum {
    kAnyPort = 0
};
```

Constants
`kAnyPort`
Indicates that any Internet port can be used.
Available in Mac OS X v10.1 and later.
Declared in `KeychainCore.h`.

Default Internet Protocol And Authentication Type Constants

Represent the internet protocols and authentication types available.

```
enum {
    kAnyProtocol = 0,
    kAnyAuthType = 0
};
```

Constants
`kAnyProtocol`
Indicates that any Internet protocol can be used.
Available in Mac OS X v10.1 and later.
Declared in `KeychainCore.h`.

`kAnyAuthType`
Indicates that any Internet authentication type can be used.
Available in Mac OS X v10.1 and later.
Declared in `KeychainCore.h`.

Keychain Events Constants

Identify keychain events.

```
typedef UInt16 KCEvent;
enum {
    kIdleKCEvent = 0,
    kLockKCEvent = 1,
    kUnlockKCEvent = 2,
    kAddKCEvent = 3,
    kDeleteKCEvent = 4,
    kUpdateKCEvent = 5,
    kPasswordChangedKCEvent = 6,
    kSystemKCEvent = 8,
    kDefaultChangedKCEvent = 9,
    kDataAccessKCEvent = 10,
    kKeychainListChangedKCEvent = 11
};
```

Constants

`kIdleKCEvent`
Indicates a `NULL` event.
Available in Mac OS X v10.0 and later.
Declared in `KeychainCore.h`.

`kLockKCEvent`
Indicates that the keychain was locked.
Available in Mac OS X v10.0 and later.
Declared in `KeychainCore.h`.

`kUnlockKCEvent`
Indicates that the keychain was unlocked.
Available in Mac OS X v10.0 and later.
Declared in `KeychainCore.h`.

`kAddKCEvent`
Indicates that an item was added to a keychain.
Available in Mac OS X v10.0 and later.
Declared in `KeychainCore.h`.

`kDeleteKCEvent`
Indicates that an item was deleted from a keychain.
Available in Mac OS X v10.0 and later.
Declared in `KeychainCore.h`.

`kUpdateKCEvent`
Indicates that a keychain item was updated.
Available in Mac OS X v10.0 and later.
Declared in `KeychainCore.h`.

`kPasswordChangedKCEvent`
Indicates that the identity of the keychain was changed.
Available in Mac OS X v10.0 and later.
Declared in `KeychainCore.h`.

`kSystemKCEvent`
Indicates that the keychain client can process events.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kDefaultChangedKCEvent`
Indicates that the default keychain has changed.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kDataAccessKCEvent`
Indicates that a process has called the function `KCGetData` (page 1111) to access a keychain item's data.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kKeychainListChangedKCEvent`
Indicates that the list of keychains has changed.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

Discussion
The `KCEvent` enumeration defines constants that identify the Keychain-related events your callback function wishes to receive. The Keychain Manager tests a mask that you pass in the `eventMask` parameter of the function `KCAddCallback` (page 1083) to determine the data transfer events your notification callback function wishes to receive. It passes these events in the `keychainEvent` parameter of the function `InvokeKCCallbackUPP` (page 1079). For a description of the Keychain-related event masks, see `Keychain Events Mask` (page 1142).

Keychain Events Mask

Identify a mask that you can use to set the keychain events you wish to receive.

```
typedef UInt16 KCEventMask;
enum {
    kIdleKCEventMask = 1 << kIdleKCEvent,
    kLockKCEventMask = 1 << kLockKCEvent,
    kUnlockKCEventMask = 1 << kUnlockKCEvent,
    kAddKCEventMask = 1 << kAddKCEvent,
    kDeleteKCEventMask = 1 << kDeleteKCEvent,
    kUpdateKCEventMask = 1 << kUpdateKCEvent,
    kPasswordChangedKCEventMask = 1 << kPasswordChangedKCEvent,
    kSystemEventKCEventMask = 1 << kSystemKCEvent,
    kDefaultChangedKCEventMask = 1 << kDefaultChangedKCEvent,
    kDataAccessKCEventMask = 1 << kDataAccessKCEvent,
    kEveryKCEventMask = 0xFFFF
};
```

Constants
`kIdleKCEventMask`
If the bit specified by this mask is set, your callback function will be invoked during a `NULL` event.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kLockKCEventMask`

If the bit specified by this mask is set, your callback function will be invoked when the keychain is locked.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kUnlockKCEventMask`

If the bit specified by this mask is set, your callback function will be invoked when the keychain is unlocked.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kAddKCEventMask`

If the bit specified by this mask is set, your callback function will be invoked when an item is added to the keychain.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kDeleteKCEventMask`

If the bit specified by this mask is set, your callback function will be invoked when an item is removed from the keychain.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kUpdateKCEventMask`

If the bit specified by this mask is set, your callback function will be invoked when a keychain item is updated.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kPasswordChangedKCEventMask`

If the bit specified by this mask is set, your callback function will be invoked when the keychain identity is changed.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kSystemEventKCEventMask`

If the bit specified by this mask is set, your callback function will be invoked when the keychain client processes an event.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kDefaultChangedKCEventMask`

If the bit specified by this mask is set, your callback function will be invoked when the default keychain is changed.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kDataAccessKCEventMask`

If the bit specified by this mask is set, your callback function will be invoked when a process calls the function `KCGetData` (page 1111).

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kEveryKCEventMask`

If the bit specified by this mask is set, your callback function will be invoked when any of the above Keychain-related events occur.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

Discussion

The `KCEventMask` enumeration defines masks your application can use to set Keychain event bits. You pass this mask in the `eventMask` parameter of the function `KCAddCallback` (page 1083), thereby defining the Keychain-related events to which your callback will respond. The Keychain Manager uses this mask to test which events your callback function will handle. It passes these events in the `keychainEvent` parameter of the function `InvokeKCCallbackUPP` (page 1079). For a description of Keychain-related events, see `Keychain Events Constants` (page 1140).

Keychain Item Attribute Tag Constants

Represent tags that identify keychain item attribute values.

```
enum {
    kClassKCItemAttr = 'clas',
    kCreationDateKCItemAttr = 'cdat',
    kModDateKCItemAttr = 'mdat',
    kDescriptionKCItemAttr = 'desc',
    kCommentKCItemAttr = 'icmt',
    kCreatorKCItemAttr = 'crtr',
    kTypeKCItemAttr = 'type',
    kScriptCodeKCItemAttr = 'scrp',
    kLabelKCItemAttr = 'labl',
    kInvisibleKCItemAttr = 'invi',
    kNegativeKCItemAttr = 'nega',
    kCustomIconKCItemAttr = 'cusi',
    kAccountKCItemAttr = 'acct',
    kServiceKCItemAttr = 'svce',
    kGenericKCItemAttr = 'gena',
    kSecurityDomainKCItemAttr = 'sdmn',
    kServerKCItemAttr = 'srvr',
    kAuthTypeKCItemAttr = 'atyp',
    kPortKCItemAttr = 'port',
    kPathKCItemAttr = 'path',
    kVolumeKCItemAttr = 'vlme',
    kAddressKCItemAttr = 'addr',
    kSignatureKCItemAttr = 'ssig',
    kProtocolKCItemAttr = 'ptcl',
    kSubjectKCItemAttr = 'subj',
    kCommonNameKCItemAttr = 'cn  ',
    kIssuerKCItemAttr = 'issu',
    kSerialNumberKCItemAttr = 'snbr',
    kEMailKCItemAttr = 'mail',
    kPublicKeyHashKCItemAttr = 'hpky',
    kIssuerURLKCItemAttr = 'iurl',
    kEncryptKCItemAttr = 'encr',
    kDecryptKCItemAttr = 'decr',
    kSignKCItemAttr = 'sign',
    kVerifyKCItemAttr = 'veri',
    kWrapKCItemAttr = 'wrap',
    kUnwrapKCItemAttr = 'unwr',
    kStartDateKCItemAttr = 'sdat',
    kEndDateKCItemAttr = 'edat'
};
typedef FourCharCode KCItemAttr;
```

Constants

`kClassKCItemAttr`

Identifies the class attribute. You use this tag to set or get a value of type `KCItemClass` that indicates whether the item is an AppleShare, Internet, or generic password, or a certificate. See "KCPublicKeyHash" (page 1135) for a description of possible values.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kCreationDateKCItemAttr`

Identifies the creation date attribute. You use this tag to set or get a value of type `UInt32` that indicates the date the item was created.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kModDateKCItemAttr`
Identifies the modification date attribute. You use this tag to set or get a value of type `UInt32` that indicates the last time the item was updated.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kDescriptionKCItemAttr`
Identifies the description attribute. You use this tag to set or get a value of type `string` that represents a user-visible string describing this item.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kCommentKCItemAttr`
Identifies the comment attribute. You use this tag to set or get a value of type `string` that represents a user-editable string containing comments for this item.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kCreatorKCItemAttr`
Identifies the creator attribute. You use this tag to set or get a value of type `OSType` that represents the item's creator.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kTypeKCItemAttr`
Identifies the type attribute. You use this tag to set or get a value of type `OSType` that represents the item's type.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kScriptCodeKCItemAttr`
Identifies the script code attribute. You use this tag to set or get a value of type `ScriptCode` that represents the script code for all strings.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kLabelKCItemAttr`
Identifies the label attribute. You use this tag to set or get a value of type `string` that represents a user-editable string containing the label for this item.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kInvisibleKCItemAttr`
Identifies the invisible attribute. You use this tag to set or get a value of type `Boolean` that indicates whether the item is invisible.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kNegativeKCItemAttr`

Identifies the negative attribute. You use this tag to set or get a value of type `Boolean` that indicates whether there is a valid password associated with this keychain item. This is useful if your application doesn't want a password for some particular service to be stored in the keychain, but prefers that it always be entered by the user. The item (typically invisible and with zero-length data) acts as a placeholder to say “don't use me.”

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kCustomIconKCItemAttr`

Identifies the custom icon attribute. You use this tag to set or get a value of type `Boolean` that indicates whether the item has an application-specific icon. To do this, you must also set the attribute value identified by the tag kTypeKCItemAttr to a file type for which there is a corresponding icon in the desktop database, and set the attribute value identified by the tag `kCreatorKCItemAttr` to an appropriate application creator type. If a custom icon corresponding to the item's type and creator can be found in the desktop database, it will be displayed by Keychain Access. Otherwise, default icons are used.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kAccountKCItemAttr`

Identifies the account attribute. You use this tag to set or get a value of type `Str63` that represents the user account. It also applies to generic and AppleShare passwords.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kServiceKCItemAttr`

Identifies the service attribute for a generic password. You use this tag to set or get a value of type `Str63` that represents the service.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kGenericKCItemAttr`

Identifies the generic attribute for a generic password. You use this tag to set or get a value of untyped bytes that represents a user-defined attribute.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kSecurityDomainKCItemAttr`

Identifies the security domain attribute for an internet password. You use this tag to set or get a value of type `Str63` that represents the Internet security domain.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kServerKCItemAttr`

Identifies the server attribute for an internet password. You use this tag to set or get a value of type `string` that represents the Internet server’s domain name or IP address.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kAuthTypeKCItemAttr`
Identifies the authentication type attribute for an internet password. You use this tag to set or get a value of type `KCAuthType` that represents the Internet authentication scheme.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kPortKCItemAttr`
Identifies the port attribute for an internet password. You use this tag to set or get a value of type `UInt16` that represents the Internet port.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kPathKCItemAttr`
Identifies the path attribute for an internet password. You use this tag to set or get a value of type `Str255` that represents the path.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kVolumeKCItemAttr`
Identifies the volume attribute for an AppleShare password. You use this tag to set or get a value of type `Str63` that represents the AppleShare volume.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kAddressKCItemAttr`
Identifies the address attribute for an AppleShare password. You use this tag to set or get a value of type `string` that represents the zone name, or the IP or domain name that represents the server address.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kSignatureKCItemAttr`
Identifies the server signature attribute for an AppleShare password. You use this tag to set or get a value of type `KCPublicKeyHash` (page 1135) that represents the server signature block.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kProtocolKCItemAttr`
Identifies the protocol attribute for an AppleShare or internet password. You use this tag to set or get a value of type `KCProtocolType` that represents the Internet protocol.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kSubjectKCItemAttr`
Identifies the subject attribute for a certificate. You use this tag to set or get `DER`-encoded data that represents the subject distinguished name.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kCommonNameKCItemAttr`
Identifies the common name attribute for a certificate. You use this tag to set or get a `UTF8`-encoded string that represents the common name.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kIssuerKCItemAttr`
Identifies the issuer attribute for a certificate. You use this tag to set or get a `DER`-encoded data that represents the issuer distinguished name.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kSerialNumberKCItemAttr`
Identifies the serial number attribute for a certificate. You use this tag to set or get a `DER`-encoded data that represents the serial number.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kEMailKCItemAttr`
Identifies the email attribute for a certificate. You use this tag to set or get an `ASCII`-encoded string that represents the issuer's email address.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kPublicKeyHashKCItemAttr`
Identifies the public key hash attribute for a certificate. You use this tag to set or get a value of type `KCPublicKeyHash` (page 1135) that represents the hash of the public key.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kIssuerURLKCItemAttr`
Identifies the issuer URL attribute for a certificate. You use this tag to set or get an `ASCII`-encoded string that represents the URL of the certificate issuer.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kEncryptKCItemAttr`
Identifies the encrypt attribute for a certificate or key. You use this tag to set or get a value of type `Boolean` that indicates whether the item can encrypt.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kDecryptKCItemAttr`
Identifies the decrypt attribute for a certificate or key. You use this tag to set or get a value of type `Boolean` that indicates whether the item can decrypt.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kSignKCItemAttr`
Identifies the sign attribute for a certificate or key. You use this tag to set or get a value of type `Boolean` that indicates whether the item can sign.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kVerifyKCItemAttr`
Identifies the verify attribute for a certificate or key. You use this tag to set or get a value of type `Boolean` that indicates whether the item can verify.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kWrapKCItemAttr`
Identifies the wrap attribute for a certificate or key. You use this tag to set or get a value of type `Boolean` that indicates whether the item can wrap.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kUnwrapKCItemAttr`
Identifies the unwrap attribute for a certificate or key. You use this tag to set or get a value of type `Boolean` that indicates whether the item can unwrap.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kStartDateKCItemAttr`
Identifies the start date attribute for a certificate or key. You use this tag to set or get a value of type `UInt32` that indicates the start date.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kEndDateKCItemAttr`
Identifies the end date attribute for a certificate or key. You use this tag to set or get a value of type `UInt32` that indicates the end date.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

Discussion
The `KCItemAttr` enumeration defines the Apple-defined tag constants that identify keychain item attribute values. Your application can use one of these tags in the `tag` field of the structure `KCAttribute` (page 1132) to identify the keychain item attribute value you wish to set or retrieve. Your application can create application-defined tags of type `KCAttrType` (page 1133).

Keychain Item Type Constants

Identify the type of keychain item.

```
enum {
    kCertificateKCItemClass = 'cert',
    kAppleSharePasswordKCItemClass = 'ashp',
    kInternetPasswordKCItemClass = 'inet',
    kGenericPasswordKCItemClass = 'genp'
};
typedef FourCharCode KCItemClass;
```

Constants

`kCertificateKCItemClass`
Specifies that the item is a digital certificate.
Available in Mac OS X v10.0 and later.
Declared in `KeychainCore.h`.

`kAppleSharePasswordKCItemClass`
Specifies that the item is an AppleShare password.
Available in Mac OS X v10.0 and later.
Declared in `KeychainCore.h`.

`kInternetPasswordKCItemClass`
Specifies that the item is an Internet password.
Available in Mac OS X v10.0 and later.
Declared in `KeychainCore.h`.

`kGenericPasswordKCItemClass`
Specifies that the item is a generic password.
Available in Mac OS X v10.0 and later.
Declared in `KeychainCore.h`.

Discussion

The `KCItemClass` enumeration defines constants your application can use to specify the type of the keychain item you wish to create, dispose, add, delete, update, copy, or locate. You pass a constant of this type to the functions `KCNewItem` (page 1120), `KCReleaseItem` (page 1121), `KCAddItem` (page 1090), `KCDeleteItem` (page 1096), `KCUpdateItem` (page 1129), `KCCopyItem` (page 1092), and `KCGetKeychain` (page 1113). You can also use these constants with the tag constant `kClassKCItemAttr`, described in `Keychain Item Attribute Tag Constants` (page 1144).

Keychain Protocol Type Constants

Identify the protocol to use in storing and retrieving Internet passwords.

```
enum {
    kKCProtocolTypeFTP = 'ftp ',
    kKCProtocolTypeFTPAccount = 'ftpa',
    kKCProtocolTypeHTTP = 'http',
    kKCProtocolTypeIRC = 'irc ',
    kKCProtocolTypeNNTP = 'nntp',
    kKCProtocolTypePOP3 = 'pop3',
    kKCProtocolTypeSMTP = 'smtp',
    kKCProtocolTypeSOCKS = 'sox ',
    kKCProtocolTypeIMAP = 'imap',
    kKCProtocolTypeLDAP = 'ldap',
    kKCProtocolTypeAppleTalk = 'atlk',
    kKCProtocolTypeAFP = 'afp ',
    kKCProtocolTypeTelnet = 'teln'
};
typedef FourCharCode KCProtocolType;
```

Constants

`kKCProtocolTypeFTP`
Specifies the File Transfer Protocol.
Available in Mac OS X v10.0 and later.
Declared in `KeychainCore.h`.

`kKCProtocolTypeFTPAccount`
Specifies the File Transfer Protocol Account.
Available in Mac OS X v10.0 and later.
Declared in `KeychainCore.h`.

`kKCProtocolTypeHTTP`
Specifies the HyperText Transfer Protocol.
Available in Mac OS X v10.0 and later.
Declared in `KeychainCore.h`.

`kKCProtocolTypeIRC`
Specifies the Internet Relay Channel Protocol.
Available in Mac OS X v10.0 and later.
Declared in `KeychainCore.h`.

`kKCProtocolTypeNNTP`
Specifies the Network News Transfer Protocol.
Available in Mac OS X v10.0 and later.
Declared in `KeychainCore.h`.

`kKCProtocolTypePOP3`
Specifies the Post Office 3 Protocol.
Available in Mac OS X v10.0 and later.
Declared in `KeychainCore.h`.

`kKCProtocolTypeSMTP`
Specifies the Simple Mail Transfer Protocol.
Available in Mac OS X v10.0 and later.
Declared in `KeychainCore.h`.

`kKCProtocolTypeSOCKS`
Specifies the Secure Proxy Server Protocol.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kKCProtocolTypeIMAP`
Specifies the Internet Message Access Protocol.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kKCProtocolTypeLDAP`
Specifies the Lightweight Directory Access Protocol.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kKCProtocolTypeAppleTalk`
Specifies the AppleTalk Protocol.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kKCProtocolTypeAFP`
Specifies the AppleTalk File Protocol.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kKCProtocolTypeTelnet`
Specifies the Telnet Protocol.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

Discussion
The `KCProtocolType` enumeration defines constants you can use to identify the type of authentication to use in storing and retrieving Internet passwords. You can pass a constant of this type in the `protocol` parameter of the functions `KCAddInternetPassword` (page 1086), `KCAddInternetPasswordWithPath` (page 1088), `KCFindInternetPassword` (page 1102), and `KCFindInternetPasswordWithPath` (page 1105).

Keychain Status Constants

Identify the keychain status.

```
enum {
    kUnlockStateKCStatus = 1,
    kRdPermKCStatus = 2,
    kWrPermKCStatus = 4
};
```

Constants
`kUnlockStateKCStatus`
If the bit specified by this mask is set (bit 0), the keychain is unlocked.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kRdPermKCStatus`

If the bit specified by this mask is set (bit 1), the keychain is unlocked with read permission.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

`kWrPermKCStatus`

If the bit specified by this mask is set (bit 2), the keychain is unlocked with write permission.

Available in Mac OS X v10.0 and later.

Declared in `KeychainCore.h`.

Result Codes

The most common result codes returned by Keychain Manager are listed below.

Result Code	Value	Description
`errKCNotAvailable`	-25291	Indicates that the Keychain Manager was not loaded. Available in Mac OS X v10.0 and later.
`errKCReadOnly`	-25292	Returned by the function `KCCopyItem` to indicate that the keychain file is read-only and cannot be edited. Available in Mac OS X v10.0 and later.
`errKCAuthFailed`	-25293	Returned by the function `KCUnlock` to indicate that the authentication failed (too many unsuccessful retries). Available in Mac OS X v10.0 and later.
`errKCNoSuchKeychain`	-25294	Returned by the functions `KCUnlock`, `KCSetDefaultKeychain`, `KCGetStatus`, and `KCGetIndKeychain` to indicate that the specified keychain was not found. Available in Mac OS X v10.0 and later.
`errKCInvalidKeychain`	-25295	Returned by the functions `KCUnlock`, `KCSetDefaultKeychain`, `KCGetStatus`, `KCGetKeychainName`, `KCChangeSettings`, and `KCCreateKeychain` to indicate that the keychain is not valid. Available in Mac OS X v10.0 and later.
`errKCDuplicateKeychain`	-25296	Returned by the function `KCCreateKeychain` to indicate that your application tried to create a keychain that already exists. Available in Mac OS X v10.0 and later.
`errKCDuplicateCallback`	-25297	Returned by the function `KCAddCallback` to indicate that your callback function was already registered. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
errKCInvalidCallback	-25298	Returned by the function KCRemoveCallback to indicate that the callback function was not previously registered. Available in Mac OS X v10.0 and later.
errKCDuplicateItem	-25299	Returned by the functions KCAddAppleSharePassword, KCAddInternetPassword, KCAddInternetPasswordWithPath, KCAddGenericPassword, and KCAddItem to indicate that you tried to add an existing keychain item to the keychain. Available in Mac OS X v10.0 and later.
errKCItemNotFound	-25300	Returned by the functions KCFindAppleSharePassword, KCFindInternetPassword, KCFindInternetPasswordWithPath, KCFindGenericPassword, KCFindNextItem, and KCFindFirstItem to indicate that no matching item was found. Available in Mac OS X v10.0 and later.
errKCBufferTooSmall	-25301	Returned by the functions KCFindAppleSharePassword, KCFindInternetPassword, KCFindInternetPasswordWithPath, KCFindGenericPassword, KCGetAttribute, KCGetData, and KCFindX509Certificates to indicate that the buffer was not large enough to contain the password data. Available in Mac OS X v10.0 and later.
errKCDataTooLarge	-25302	Returned by the functions KCAddAppleSharePassword, KCAddInternetPassword, KCAddInternetPasswordWithPath, KCAddGenericPassword, KCSetAttribute, and KCSetData to indicate that the data is too large. Available in Mac OS X v10.0 and later.
errKCNoSuchAttr	-25303	Returned by the functions KCSetAttribute, KCGetAttribute, and KCFindFirstItem to indicate that no such attribute exists. Available in Mac OS X v10.0 and later.
errKCInvalidItemRef	-25304	Returned by the functions KCSetAttribute, KCGetAttribute, KCSetData, KCGetData, KCAddItem, KCDeleteItem, KCUpdateItem, KCCopyItem, and KCGetKeychain to indicate that the keychain item reference is invalid. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
errKCInvalidSearchRef	-25305	Returned by the functions KCFindNextItem and KCReleaseSearch to indicate that the specified search reference is invalid. Available in Mac OS X v10.0 and later.
errKCNoSuchClass	-25306	Returned by the function KCCopyItem to indicate that the item class does not exist. Available in Mac OS X v10.0 and later.
errKCNoDefaultKeychain	-25307	Returned by the functions KCChangeSettings, KCSetDefaultKeychain, KCGetDefaultKeychain, KCAddAppleSharePassword, KCAddInternetPassword, KCAddInternetPasswordWithPath, KCAddGenericPassword, KCFindAppleSharePassword, KCFindInternetPassword, KCFindInternetPasswordWithPath, KCFindGenericPassword, KCCopyItem, KCAddItem, KCDeleteItem, KCUpdateItem, KCFindNextItem, KCFindFirstItem, and KCFindX509Certificates to indicate that there is no default keychain. Available in Mac OS X v10.0 and later.
errKCInteractionNotAllowed	-25308	Returned by the functions KCCreateKeychain, KCChangeSettings, KCUnlock, and KCGetData (the latter two only when the Unlock Dialog and Allow Access dialog boxes are needed) to indicate that there is no start-up keychain. Available in Mac OS X v10.0 and later.
errKCReadOnlyAttr	-25309	Returned by the function KCSetAttribute to indicate that the keychain item attribute is read-only. Available in Mac OS X v10.0 and later.
errKCWrongKCVersion	-25310	Indicates that the wrong version of Keychain Manager is installed to perform this operation. Available in Mac OS X v10.0 and later.
errKCKeySizeNotAllowed	-25311	Indicates that the key size is illegal. Available in Mac OS X v10.0 and later.
errKCNoStorageModule	-25312	Returned by functions that prompts the loading of the Keychain Manager to indicate that the storage module is not found. Available in Mac OS X v10.0 and later.
errKCNoCertificateModule	-25313	Returned when a function is required for a certificate and the certificate module is not found. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
`errKCNoPolicyModule`	-25314	Returned when a function is required for a trust policy and the policy module is not found. Available in Mac OS X v10.0 and later.
`errKCInteractionRequired`	-25315	Returned by the function `KCUnlock` to indicate that user interaction is required for this operation. Available in Mac OS X v10.0 and later.
`errKCDataNotAvailable`	-25316	Indicates that the requested data is not available. Available in Mac OS X v10.0 and later.
`errKCDataNotModifiable`	-25317	Returned by the functions `KCSetData` and `KCGetData` to indicate that the data cannot be modified. Available in Mac OS X v10.0 and later.
`errKCCreateChainFailed`	-25318	Returned by the functions `KCChooseCertificate` and `KCFindX509Certificates` to indicate that the attempt to create a new keychain failed. Available in Mac OS X v10.0 and later.

List Manager Reference (Not Recommended)

Framework: Carbon/Carbon.h
Declared in Lists.h

Overview

Important: The List Manager is deprecated in Mac OS X version 10.5 and later. The replacement API is the Data Browser. For more information, see *Data Browser Programming Guide*.

In Mac OS 9 and earlier, the List Manager allowed applications to create, manipulate, and display scrolling lists of data items in a window. The List Manager was included in Carbon to facilitate the porting of legacy applications to Mac OS X. For Carbon applications, the Data Browser provides a more convenient way to present data for browsing and to create easily customized lists whose columns can be sorted, moved, and resized.

You should not use the List Manager in new application development.

Functions by Task

Accessing and Manipulating Cell Data

`LAddToCell` (page 1177) Deprecated in Mac OS X v10.5
Appends data to the data already contained in a cell.

`LClrCell` (page 1179) Deprecated in Mac OS X v10.5
Clears the data contained in a cell.

`LGetCell` (page 1183) Deprecated in Mac OS X v10.5
Copies a cell's data.

`LGetCellDataLocation` (page 1183) Deprecated in Mac OS X v10.5
Finds the memory location of cell data.

`LSetCell` (page 1190) Deprecated in Mac OS X v10.5
Changes the data contained in a cell.

Adding and Deleting Columns and Rows To and From a List

`LAddColumn` (page 1175) Deprecated in Mac OS X v10.5
Adds one or more columns to a list.

`LAddRow` (page 1176) Deprecated in Mac OS X v10.5
Adds one or more rows to a list.

`LDelColumn` (page 1180) Deprecated in Mac OS X v10.5
Deletes one or more columns from a list.

`LDelRow` (page 1181) Deprecated in Mac OS X v10.5
Deletes one or more rows from a list.

Changing the Size of Cells and Lists

`LCellSize` (page 1178) Deprecated in Mac OS X v10.5
Changes the size of cells in a list.

`LSize` (page 1193) Deprecated in Mac OS X v10.5
Changes the size of a list.

Creating and Disposing of Lists

`LDispose` (page 1181) Deprecated in Mac OS X v10.5
Disposes of the memory associated with a list.

`LNew` (page 1186) Deprecated in Mac OS X v10.5
Creates a new list in a window.

Creating and Managing Universal Procedure Pointers

`DisposeListClickLoopUPP` (page 1164) Deprecated in Mac OS X v10.5
Disposes of the universal procedure pointer (UPP) to a list click loop callback function.

`DisposeListDefUPP` (page 1165) Deprecated in Mac OS X v10.5
Disposes of the universal procedure pointer (UPP) to a list definition callback function.

`DisposeListSearchUPP` (page 1165) Deprecated in Mac OS X v10.5
Disposes of the universal procedure pointer (UPP) to a list search callback function.

`InvokeListClickLoopUPP` (page 1173) Deprecated in Mac OS X v10.5
Calls your list click loop callback function.

`InvokeListDefUPP` (page 1173) Deprecated in Mac OS X v10.5
Calls your list definition callback function.

`InvokeListSearchUPP` (page 1174) Deprecated in Mac OS X v10.5
Calls your list search callback function

`NewListClickLoopUPP` (page 1194) Deprecated in Mac OS X v10.5
Creates a new universal procedure pointer (UPP) to a list click loop callback function.

`NewListDefUPP` (page 1194) Deprecated in Mac OS X v10.5
Creates a new universal procedure pointer (UPP) to a list definition callback function.

`NewListSearchUPP` (page 1195) Deprecated in Mac OS X v10.5
Creates a new universal procedure pointer (UPP) to a list search callback function.

Determining or Changing the Selection

`LGetSelect` (page 1184) Deprecated in Mac OS X v10.5
Gets information about which cells are selected.

`LSetSelect` (page 1192) Deprecated in Mac OS X v10.5
Selects or deselects a cell.

Getting Information About Cells

`LLastClick` (page 1185) Deprecated in Mac OS X v10.5
Determines the coordinates of the last cell clicked in a particular list.

`LNextCell` (page 1187) Deprecated in Mac OS X v10.5
Finds the next cell in a given row, in a given column, or in an entire list.

`LRect` (page 1188) Deprecated in Mac OS X v10.5
Finds a rectangle that encloses a cell.

Modifying a List's Appearance

`LAutoScroll` (page 1178) Deprecated in Mac OS X v10.5
Scrolls a list so that the first selected cell is in the upper-left corner of the list's visible rectangle.

`LDraw` (page 1182) Deprecated in Mac OS X v10.5
Draws a cell in a list.

`LScroll` (page 1189) Deprecated in Mac OS X v10.5
Scrolls a list a specified number of rows and columns.

`LSetDrawingMode` (page 1191) Deprecated in Mac OS X v10.5
Changes the automatic drawing mode specified when creating a list.

Responding to Events Affecting Lists

`LActivate` (page 1175) Deprecated in Mac OS X v10.5
Activates or deactivates a list.

`LClick` (page 1179) Deprecated in Mac OS X v10.5
Processes a mouse-down event in a list.

`LUpdate` (page 1193) Deprecated in Mac OS X v10.5
Responds to an update event.

Searching a List for a Particular Item

`LSearch` (page 1190) Deprecated in Mac OS X v10.5
Finds a cell whose data matches data that you specify.

Miscellaneous

`CreateCustomList` (page 1163) Deprecated in Mac OS X v10.5

`GetListActive` (page 1165) Deprecated in Mac OS X v10.5

`GetListCellIndent` (page 1166) Deprecated in Mac OS X v10.5

`GetListCellSize` (page 1166) Deprecated in Mac OS X v10.5

`GetListClickLocation` (page 1167) Deprecated in Mac OS X v10.5

`GetListClickLoop` (page 1167) Deprecated in Mac OS X v10.5

`GetListClickTime` (page 1167) Deprecated in Mac OS X v10.5

`GetListDataBounds` (page 1168) Deprecated in Mac OS X v10.5

`GetListDataHandle` (page 1168) Deprecated in Mac OS X v10.5

`GetListDefinition` (page 1169) Deprecated in Mac OS X v10.5

`GetListFlags` (page 1169) Deprecated in Mac OS X v10.5

`GetListHorizontalScrollBar` (page 1169) Deprecated in Mac OS X v10.5

`GetListMouseLocation` (page 1170) Deprecated in Mac OS X v10.5

`GetListPort` (page 1170) Deprecated in Mac OS X v10.5

`GetListRefCon` (page 1171) Deprecated in Mac OS X v10.5

`GetListSelectionFlags` (page 1171) Deprecated in Mac OS X v10.5

`GetListUserHandle` (page 1171) Deprecated in Mac OS X v10.5

`GetListVerticalScrollBar` (page 1172) Deprecated in Mac OS X v10.5

GetListViewBounds (page 1172) Deprecated in Mac OS X v10.5

GetListVisibleCells (page 1173) Deprecated in Mac OS X v10.5

RegisterListDefinition (page 1195) Deprecated in Mac OS X v10.5

SetListCellIndent (page 1196) Deprecated in Mac OS X v10.5

SetListClickLoop (page 1196) Deprecated in Mac OS X v10.5

SetListClickTime (page 1197) Deprecated in Mac OS X v10.5

SetListFlags (page 1197) Deprecated in Mac OS X v10.5

SetListLastClick (page 1198) Deprecated in Mac OS X v10.5

SetListPort (page 1198) Deprecated in Mac OS X v10.5

SetListRefCon (page 1198) Deprecated in Mac OS X v10.5

SetListSelectionFlags (page 1199) Deprecated in Mac OS X v10.5

SetListUserHandle (page 1199) Deprecated in Mac OS X v10.5

SetListViewBounds (page 1200) Deprecated in Mac OS X v10.5

Functions

CreateCustomList

(Deprecated in Mac OS X v10.5.)

```
OSStatus CreateCustomList (
   const Rect *rView,
   const ListBounds *dataBounds,
   Point cellSize,
   const ListDefSpec *theSpec,
   WindowRef theWindow,
   Boolean drawIt,
   Boolean hasGrow,
   Boolean scrollHoriz,
   Boolean scrollVert,
   ListHandle *outList
);
```

Parameters

rView

dataBounds

cellSize

theSpec

theWindow

drawIt

hasGrow

scrollHoriz

scrollVert

outList

Return Value
A result code.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Lists.h`

DisposeListClickLoopUPP

Disposes of the universal procedure pointer (UPP) to a list click loop callback function. (Deprecated in Mac OS X v10.5.)

```
void DisposeListClickLoopUPP (
   ListClickLoopUPP userUPP
);
```

Parameters

userUPP

Discussion
See the `ListClickLoopProcPtr` (page 1200) callback for more information.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
Lists.h

DisposeListDefUPP

Disposes of the universal procedure pointer (UPP) to a list definition callback function. (Deprecated in Mac OS X v10.5.)

```
void DisposeListDefUPP (
   ListDefUPP userUPP
);
```

Parameters
userUPP

Discussion
See the ListDefProcPtr (page 1201) callback for more information.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
Lists.h

DisposeListSearchUPP

Disposes of the universal procedure pointer (UPP) to a list search callback function. (Deprecated in Mac OS X v10.5.)

```
void DisposeListSearchUPP (
   ListSearchUPP userUPP
);
```

Parameters
userUPP

Discussion
See the ListSearchProcPtr (page 1204) callback for more information.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
Lists.h

GetListActive

(Deprecated in Mac OS X v10.5.)

```
Boolean GetListActive (
   ListHandle list
);
```

Parameters

list

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
Lists.h

GetListCellIndent

(Deprecated in Mac OS X v10.5.)

```
Point * GetListCellIndent (
   ListHandle list,
   Point *indent
);
```

Parameters

list

indent

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
Lists.h

GetListCellSize

(Deprecated in Mac OS X v10.5.)

```
Point * GetListCellSize (
   ListHandle list,
   Point *size
);
```

Parameters

list

size

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Lists.h`

GetListClickLocation

(Deprecated in Mac OS X v10.5.)

```
Point * GetListClickLocation (
   ListHandle list,
   Point *click
);
```

Parameters

list

click

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Lists.h`

GetListClickLoop

(Deprecated in Mac OS X v10.5.)

```
ListClickLoopUPP GetListClickLoop (
   ListHandle list
);
```

Parameters

list

Return Value
See the description of the `ListClickLoopUPP` data type.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Lists.h`

GetListClickTime

(Deprecated in Mac OS X v10.5.)

```
SInt32 GetListClickTime (
   ListHandle list
);
```

Parameters

list

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Lists.h`

GetListDataBounds

(Deprecated in Mac OS X v10.5.)

```
ListBounds * GetListDataBounds (
   ListHandle list,
   ListBounds *bounds
);
```

Parameters

list

bounds

Return Value
See the description of the `ListBounds` data type.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Lists.h`

GetListDataHandle

(Deprecated in Mac OS X v10.5.)

```
DataHandle GetListDataHandle (
   ListHandle list
);
```

Parameters

list

Return Value
See the description of the `DataHandle` data type.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Lists.h`

GetListDefinition

(Deprecated in Mac OS X v10.5.)

```
Handle GetListDefinition (
   ListHandle list
);
```

Parameters
list

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Lists.h`

GetListFlags

(Deprecated in Mac OS X v10.5.)

```
OptionBits GetListFlags (
   ListHandle list
);
```

Parameters
list

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Lists.h`

GetListHorizontalScrollBar

(Deprecated in Mac OS X v10.5.)

```
ControlRef GetListHorizontalScrollBar (
   ListHandle list
);
```

Parameters

list

Return Value
See the Control Manager documentation for a description of the `ControlRef` data type.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Lists.h`

GetListMouseLocation

(Deprecated in Mac OS X v10.5.)

```
Point * GetListMouseLocation (
   ListHandle list,
   Point *mouse
);
```

Parameters

list

mouse

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Lists.h`

GetListPort

(Deprecated in Mac OS X v10.5.)

```
CGrafPtr GetListPort (
   ListHandle list
);
```

Parameters

list

Return Value
See the QuickDraw Manager documentation for a description of the `CGrafPtr` data type.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Lists.h`

GetListRefCon

(Deprecated in Mac OS X v10.5.)

```
SInt32 GetListRefCon (
   ListHandle list
);
```

Parameters
list

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Lists.h`

GetListSelectionFlags

(Deprecated in Mac OS X v10.5.)

```
OptionBits GetListSelectionFlags (
   ListHandle list
);
```

Parameters
list

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Lists.h`

GetListUserHandle

(Deprecated in Mac OS X v10.5.)

```
Handle GetListUserHandle (
   ListHandle list
);
```

Parameters

list

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Lists.h`

GetListVerticalScrollBar

(Deprecated in Mac OS X v10.5.)

```
ControlRef GetListVerticalScrollBar (
   ListHandle list
);
```

Parameters

list

Return Value
See the Control Manager documentation for a description of the `ControlRef` data type.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Lists.h`

GetListViewBounds

(Deprecated in Mac OS X v10.5.)

```
Rect * GetListViewBounds (
   ListHandle list,
   Rect *view
);
```

Parameters

list

view

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Lists.h`

GetListVisibleCells

(Deprecated in Mac OS X v10.5.)

```
ListBounds * GetListVisibleCells (
   ListHandle list,
   ListBounds *visible
);
```

Parameters

list

visible

Return Value
See the description of the `ListBounds` data type.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Lists.h`

InvokeListClickLoopUPP

Calls your list click loop callback function. (Deprecated in Mac OS X v10.5.)

```
Boolean InvokeListClickLoopUPP (
   ListClickLoopUPP userUPP
);
```

Parameters

userUPP

Discussion
See the `ListClickLoopProcPtr` (page 1200) callback for more information.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`Lists.h`

InvokeListDefUPP

Calls your list definition callback function. (Deprecated in Mac OS X v10.5.)

```
void InvokeListDefUPP (
   short lMessage,
   Boolean lSelect,
   Rect *lRect,
   Cell lCell,
   short lDataOffset,
   short lDataLen,
   ListHandle lHandle,
   ListDefUPP userUPP
);
```

Parameters

lMessage

lSelect

lRect

lCell

lDataOffset

lDataLen

lHandle

userUPP

Discussion
See the `ListDefProcPtr` (page 1201) callback for more information.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`Lists.h`

InvokeListSearchUPP

Calls your list search callback function (Deprecated in Mac OS X v10.5.)

```
short InvokeListSearchUPP (
   Ptr aPtr,
   Ptr bPtr,
   short aLen,
   short bLen,
   ListSearchUPP userUPP
);
```

Parameters

aPtr

bPtr

aLen

bLen

userUPP

Return Value

Discussion
See the `ListSearchProcPtr` (page 1204) callback for more information.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`Lists.h`

LActivate

Activates or deactivates a list. (Deprecated in Mac OS X v10.5.)

```
void LActivate (
   Boolean act,
   ListHandle lHandle
);
```

Parameters

act

Indicates whether the list should be activated. Specify `TRUE` to activate the list. Specify `FALSE` to deactivate the list.

lHandle

The list to be activated or deactivated.

Discussion
If a list is being deactivated, this function removes highlighting from selected cells and hides the scroll bars. If a list is being activated, the function highlights selected cells and shows the scroll bars.

This function has no effect on a list's size box, if one exists.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Lists.h`

LAddColumn

Adds one or more columns to a list. (Deprecated in Mac OS X v10.5.)

```
short LAddColumn (
   short count,
   short colNum,
   ListHandle lHandle
);
```

Parameters

count

The number of columns to add.

colNum

The column number of the first column to add.

lHandle
The list to which to add the columns.

Return Value
The column number of the first column added, which is equal to the value specified by the `colNum` parameter if that value is a valid column number. If the column number specified by `colNum` is not already in the list, then new last columns are added. The value returned by this function thus has significance only in this case.

Discussion
This function inserts columns starting at the column specified by the `colNum` parameter. If there is insufficient memory in the heap to add the new columns, this function may fail to add the new columns although it returns a positive function result. Be sure there is enough memory in the heap to allocate the new columns before calling `LAddColumn`.

Columns whose column numbers are initially greater than `colNum` have their column numbers increased by `count`.

If the automatic drawing mode is enabled and the columns added by the function are visible, then the list (including its scroll bars) is updated. New cells created by a call to this function are initially empty.

You may add columns to a list that does not yet have rows. The `dataBounds` field of the list record reflects that the list has columns, but you can only access cells when both rows and columns have been added.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Lists.h`

LAddRow

Adds one or more rows to a list. (Deprecated in Mac OS X v10.5.)

```
short LAddRow (
   short count,
   short rowNum,
   ListHandle lHandle
);
```

Parameters
count
The number of rows to add.

rowNum
The row number of the first row to add.

lHandle
The list to add the rows to.

Return Value
The row number of the first row added, which is equal to the value specified by the `rowNum` parameter if that value is a valid row number. If the row number specified by `rowNum` is not already in the list, then new last rows are added. The value returned by this function thus has significance only in this case.

Discussion
This function inserts rows starting at the row specified by the `rowNum` parameter. If there is insufficient memory in the heap to add the new rows, the function may fail to add the new rows although it returns a positive function result. Be sure there is enough memory in the heap to allocate the new rows before calling this function.

Rows whose row numbers are initially greater than `rowNum` have their row numbers increased by `count`.

If the automatic drawing mode is enabled and the rows added by this function are visible, then the list (including its scroll bars) is updated. New cells created by a call to this function are initially empty.

You may add rows to a list that does not yet have columns. The `dataBounds` field of the list record reflects that the list has rows, but you can only access cells when both rows and columns have been added.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Lists.h`

LAddToCell

Appends data to the data already contained in a cell. (Deprecated in Mac OS X v10.5.)

```
void LAddToCell (
   const void *dataPtr,
   short dataLen,
   Cell theCell,
   ListHandle lHandle
);
```

Parameters

`dataPtr`
A pointer to the data to be appended.

`dataLen`
The length in bytes of the data to be appended.

`theCell`
The coordinates of the cell to which the data should be appended.

`lHandle`
The list containing the cell given in the `theCell` parameter.

Discussion
If the cell coordinates specified by the parameter `theCell` are invalid, then this function does nothing.

If the data of a visible cell is changed and the automatic drawing mode is enabled, the function updates the list.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Lists.h`

LAutoScroll

Scrolls a list so that the first selected cell is in the upper-left corner of the list's visible rectangle. (Deprecated in Mac OS X v10.5.)

```
void LAutoScroll (
   ListHandle lHandle
);
```

Parameters

`lHandle`

The list to be scrolled.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Lists.h`

LCellSize

Changes the size of cells in a list. (Deprecated in Mac OS X v10.5.)

```
void LCellSize (
   Point cSize,
   ListHandle lHandle
);
```

Parameters

`cSize`

The new size of each cell in the list. This function sets the `cellSize` field of the list record of the list to the value of the `cSize` parameter. That is, the list's new cells will be of width `cSize.h` and of height `cSize.v`.

All cells in a list must be the same size.

`lHandle`

The list whose cells' size is being changed.

Discussion
The function updates the list's visible rectangle to contain cells of the specified size. However, it does not redraw any cells.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Lists.h`

LClick

Processes a mouse-down event in a list. (Deprecated in Mac OS X v10.5.)

```
Boolean LClick (
   Point pt,
   EventModifiers modifiers,
   ListHandle lHandle
);
```

Parameters

pt

The location in local coordinates of the mouse-down event. Your application can simply call `GlobalToLocal(myEvent.where)` and then pass `myEvent.where` in this parameter.

If the `pt` parameter specifies a portion of the list's visible rectangle, then cells are selected with an algorithm that depends on the list's selection flags and on the `modifiers` parameter. If the user drags the cursor above or below the list's visible rectangle and vertical autoscrolling is enabled, then the List Manager vertically autoscrolls the list. If the user drags the cursor to the right or the left of the list's visible rectangle and horizontal autoscrolling is enabled, then the List Manager horizontally autoscrolls the list.

If the `pt` parameter specifies a point within the list's scroll bar, then the List Manager calls the scroll bar's control definition function to track the cursor and it scrolls the list appropriately.

modifiers

An integer value corresponding to the `modifiers` field of the event record.

lHandle

The list in which the mouse-down event occurred.

Return Value

`TRUE` if the click was a double-click, or `FALSE` otherwise.

Discussion

The `LClick` function handles all user interaction until the user releases the mouse button.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Lists.h`

LClrCell

Clears the data contained in a cell. (Deprecated in Mac OS X v10.5.)

```
void LClrCell (
   Cell theCell,
   ListHandle lHandle
);
```

Parameters

theCell

The coordinates of the cell to be cleared.

lHandle
The list containing the cell given in the `theCell` parameter.

Discussion
If the cell coordinates specified by the `theCell` parameter are invalid, then the function does nothing.

If the data of a visible cell is cleared and the automatic drawing mode is enabled, the function updates the list.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Lists.h`

LDelColumn

Deletes one or more columns from a list. (Deprecated in Mac OS X v10.5.)

```
void LDelColumn (
   short count,
   short colNum,
   ListHandle lHandle
);
```

Parameters
count
The number of columns to delete, or 0 to delete all columns.
colNum
The column number of the first column to delete.
lHandle
The list from which to delete the columns.

Discussion
This function deletes columns starting at the column specified by the `colNum` parameter. If the column specified by `colNum` is invalid, then nothing is done.

Your application can quickly delete all columns from a list (and thus delete all cell data) simply by setting the `count` parameter to 0. The number of rows is left unchanged. Your application can achieve the same effect by setting the `colNum` parameter to `(**lHandle).dataBounds.left` and setting the `count` parameter to a value greater than `(**lHandle).dataBounds.right - (**lHandle).dataBounds.left`.

Columns whose column numbers are initially greater than `colNum` have their column numbers decreased by `count`.

If the automatic drawing mode is enabled and one or more of the columns deleted by this function are visible, then the list (including its scroll bars) is updated.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Lists.h`

LDelRow

Deletes one or more rows from a list. (Deprecated in Mac OS X v10.5.)

```
void LDelRow (
   short count,
   short rowNum,
   ListHandle lHandle
);
```

Parameters

`count`
The number of rows to delete, or 0 to delete all rows.

`rowNum`
The row number of the first row to delete.

`lHandle`
The list from which to delete the rows.

Discussion
This function deletes rows starting at the row specified by the `rowNum` parameter. If the row specified by `rowNum` is invalid, then nothing is done.

Your application can quickly delete all rows from a list (and thus delete all cell data) simply by setting the `count` parameter to 0. The number of columns is left unchanged. Your application can achieve the same effect by setting the `rowNum` parameter to `(**lHandle).dataBounds.top` and setting the `count` parameter to a value greater than `(**lHandle).dataBounds.bottom - (**lHandle).dataBounds.top`.

Rows whose row numbers are initially greater than `rowNum` have their row numbers decreased by `count`.

If the automatic drawing mode is enabled and one or more of the rows deleted by the function are visible, then the list (including its scroll bars) is updated.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Lists.h`

LDispose

Disposes of the memory associated with a list. (Deprecated in Mac OS X v10.5.)

```
void LDispose (
   ListHandle lHandle
);
```

Parameters

lHandle

The list to be disposed of.

Discussion

This function releases all memory allocated by the List Manager in creating a list. First, it issues a close request to the list definition function and calls the Control Manager function `DisposeControl` for the list's scroll bars (if any). The function then uses the Memory Manager to free the memory referenced by the cells field, then disposes of the list record itself.

Because it disposes of data associated with cells in your list, there is no need to clear the data from list cells or to delete individual rows and columns before calling this function.

This function does not dispose of any memory associated with a list that the List Manager has not allocated. In particular, it does not dispose of any memory referenced by the `userHandle` field of the list record. Your application is responsible for deallocating any memory it has allocated through the `userHandle` field before calling this function.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Lists.h`

LDraw

Draws a cell in a list. (Deprecated in Mac OS X v10.5.)

```
void LDraw (
   Cell theCell,
   ListHandle lHandle
);
```

Parameters

theCell

The cell to draw.

lHandle

The list containing the cell identified by the parameter `theCell`.

Discussion

The List Manager makes the list's graphics port the current port, sets the clipping region to the cell's rectangle, and calls the list definition function to draw the cell. It restores the clipping region and port before exiting.

Ordinarily, you should only need to use this function when the automatic drawing mode has been disabled.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`Lists.h`

LGetCell

Copies a cell's data. (Deprecated in Mac OS X v10.5.)

```
void LGetCell (
   void *dataPtr,
   short *dataLen,
   Cell theCell,
   ListHandle lHandle
);
```

Parameters

dataPtr
A pointer to the location to which to copy the cell's data.

dataLen
On input, a pointer to the maximum number of bytes to copy. On return, a pointer to the number of bytes actually copied.

theCell
The cell whose data is to be copied.

lHandle
The list containing the cell specified by the parameter `theCell`.

Discussion
If the cell data is longer than `dataLen`, only `dataLen` bytes are copied and the `dataLen` parameter is unchanged. If the cell data is shorter than `dataLen`, then the function sets `dataLen` to the length in bytes of the cell's data.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Lists.h`

LGetCellDataLocation

Finds the memory location of cell data. (Deprecated in Mac OS X v10.5.)

```
void LGetCellDataLocation (
   short *offset,
   short *len,
   Cell theCell,
   ListHandle lHandle
);
```

Parameters

offset

On return, a pointer to the offset of the cell's data, specified from the beginning of the data handle referenced by the `cells` field of the list record.

len

On return, a pointer to the length of the cell's data in bytes.

theCell

The cell whose data's location is sought.

lHandle

The list containing the cell specified by the parameter `theCell`.

Discussion

Your application can use this function to read cell data. The `cells` field of the list record contains a handle to a relocatable block used to store all cell data. When this function returns, the `offset` parameter contains the offset of the specified cell's data in this relocatable block, and the `len` parameter specifies the length in bytes of the cell's data. In other words, the first byte of cell data is located at `Ptr(ORD4(lHandle^^.cells^) + offset)`, and the last byte of cell data is located at `Ptr(ORD4(lHandle^^.cells^) + offset + len)`. Your application should not modify the contents of the `cells` field directly. To change a cell's data, use the `LSetCell` (page 1190) function or the `LAddToCell` (page 1177) function.

If the cell coordinates specified by the parameter `theCell` are invalid, then the function sets the `offset` and `len` parameters to –1.

This function is also available as the `LFind` function.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Lists.h`

LGetSelect

Gets information about which cells are selected. (Deprecated in Mac OS X v10.5.)

```
Boolean LGetSelect (
   Boolean next,
   Cell *theCell,
   ListHandle lHandle
);
```

Parameters

next

Indicates whether the function should check only the cell specified by the parameter `theCell`, or whether it should try to find the next selected cell. If `next` is `TRUE`, then the function searches the list for the first selected cell beginning at the cell specified by `theCell`. (In particular, it first checks cells in row `theCell.v`, and then cells in the next row, and so on.)

If `next` is `FALSE`, then the function checks only the cell specified by the parameter `theCell`.

theCell

On input, a pointer to the first cell whose selection status should be checked. If `next` is `TRUE`, then, on return this parameter is a pointer to the next selected cell greater than or equal to the cell specified on input. Otherwise, this parameter remains unchanged.

lHandle

The list in which the selection is being checked.

Return Value

`TRUE` if `next` is `TRUE` and the function finds a selected cell, or if `next` is `FALSE` and the cell specified by `theCell` is selected. If this function does not find a selected cell, `FALSE`.

Special Considerations

This function is contained in a resource of resource type `'PACK'`. Calling it could result in the loading of the package resource and the allocation of memory. Thus, your application should not call this function from within an interrupt, such as in a completion function or VBL task.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Lists.h`

LLastClick

Determines the coordinates of the last cell clicked in a particular list. (Deprecated in Mac OS X v10.5.)

```
Cell LLastClick (
   ListHandle lHandle
);
```

Parameters

lHandle

The list to be checked for the last cell clicked.

Return Value

The cell coordinates of the last cell clicked. If the user has not clicked a cell since the creation of the list, then both the `h` and `v` fields of the cell returned contain negative numbers. See the description of the `Cell` data type.

Discussion
Note that the last cell clicked is not necessarily the last cell selected. The user could Shift-click in one cell and then drag the cursor to select other cells.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Lists.h`

LNew

Creates a new list in a window. (Deprecated in Mac OS X v10.5.)

```
ListHandle LNew (
   const Rect *rView,
   const ListBounds *dataBounds,
   Point cSize,
   short theProc,
   WindowRef theWindow,
   Boolean drawIt,
   Boolean hasGrow,
   Boolean scrollHoriz,
   Boolean scrollVert
);
```

Parameters

rView
A pointer to the rectangle in which to display the list, in local coordinates of the window specified by the `theWindow` parameter. This rectangle does not include the area to be taken up by the list's scroll bars.

dataBounds
A pointer to the initial data bounds for the list. By setting the `left` and `top` fields of this rectangle to (0,0) and the `right` and `bottom` fields to (`kMyInitialColumns`, `kMyInitialRows`), your application can create a list that has `kMyInitialColumns` columns and `kMyInitialRows` rows.

cSize
The size of each cell in the list. If your application specifies (0,0) and is using the default list definition function, the List Manager sets the `v` coordinate of this parameter to the sum of the ascent, descent, and leading of the current font, and it sets the `h` coordinate using the following formula:

```
cSize.h = (rView.right - rView.left) / (dataBounds.right - dataBounds.left).
```

theProc
The resource ID of the list definition function to use for the list. To use the default list definition function, which supports the display of unstyled text, specify a resource ID of 0.

theWindow
A pointer to the window in which to install the list.

drawIt

Indicates whether the List Manager should initially enable the automatic drawing mode. When the automatic drawing mode is enabled, the List Manager automatically redraws the list whenever a change is made to it. You can later change this setting using the `LSetDrawingMode` (page 1191) function. Your application should leave the automatic drawing mode disabled only for short periods of time when making changes to a list (by, for example, adding rows and columns).

hasGrow

Indicates whether the List Manager should leave room for a size box. The List Manager does not actually draw the grow icon. Usually, your application can draw it with the Window Manager's `DrawGrowIcon` function.

scrollHoriz

Indicates whether the list should contain a horizontal scroll bar. Specify `TRUE` if your list requires a horizontal scroll bar; specify `FALSE` otherwise.

scrollVert

Indicates whether the list should contain a vertical scroll bar. Specify `TRUE` if your list requires a vertical scroll bar; specify `FALSE` otherwise.

Return Value

A handle to the newly created list, or if the function cannot allocate the list, `NULL`. This might happen if there is not enough memory available or if the function cannot load the resource specified by the `theProc` parameter. If it returns successfully, then all of the fields of the list record referenced by the returned handle are correctly set. See the description of the `ListHandle` data type.

Discussion

If the list contains a horizontal or vertical scroll bar and the window specified by the parameter `theWindow` is visible, this function draws the scroll bar for the new list in the window just outside the list's visible rectangle specified by the `rView` parameter. This function does not, however, draw a 1-pixel border around the list's visible rectangle.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Lists.h`

LNextCell

Finds the next cell in a given row, in a given column, or in an entire list. (Deprecated in Mac OS X v10.5.)

```
Boolean LNextCell (
   Boolean hNext,
   Boolean vNext,
   Cell *theCell,
   ListHandle lHandle
);
```

Parameters

hNext

Indicates whether the function should check columns other than the current column. To get the next cell in a row, set this parameter to `TRUE` and set `vNext` to `FALSE`. The function then tries to find a cell whose coordinates are greater than those of the cell specified in `theCell` parameter but that is in the same row as `theCell`.

To get the next cell in a column, set this parameter to `FALSE` and set `vNext` to `TRUE`. The function then tries to find a cell whose coordinates are greater than those of the cell specified in `theCell` but that is in the same column as `theCell`.

To get the next cell in a list, set both this parameter and `vNext` to `TRUE`. This function then tries to find a cell whose coordinates are greater than those of the cell specified in the parameter `theCell`.

vNext

Indicates whether the function should check rows other than the current row. To get the next cell in a row, set this parameter to `FALSE` and set `hNext` to `TRUE`. The function then tries to find a cell whose coordinates are greater than those of the cell specified in `theCell` parameter but that is in the same row as `theCell`.

To get the next cell in a column, set this parameter to `TRUE` and set `hNext` to `FALSE`. The function then tries to find a cell whose coordinates are greater than those of the cell specified in `theCell` but that is in the same column as `theCell`.

To get the next cell in a list, set both this parameter and `hNext` to `TRUE`. This function then tries to find a cell whose coordinates are greater than those of the cell specified in the parameter `theCell`.

theCell

A pointer to the coordinates of the current cell. On return, a pointer to the next cell in the list, column or row being searched. If there are no more cells in the list, column or row, this parameter remains unchanged.

lHandle

The list in which to find the next cell.

Return Value

`TRUE`, if the function finds the next cell in the list, column or row being searched. `FALSE`, if the cell initially specified by `theCell` is the last in the row, column or list being searched. Also `FALSE` when both `hNext` and `vNext` are `FALSE`.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Lists.h`

LRect

Finds a rectangle that encloses a cell. (Deprecated in Mac OS X v10.5.)

```
void LRect (
   Rect *cellRect,
   Cell theCell,
   ListHandle lHandle
);
```

Parameters

cellRect

On return, a pointer to the rectangle enclosing the cell, specified in local coordinates of the list's graphics port. This rectangle is not necessarily within the list's rectangle.

theCell

The cell for which an enclosing rectangle is sought. This function does not check whether the cell is actually contained within the list's visible rectangle.

If this parameter specifies cell coordinates not contained within the list, this function sets the `cellRect` parameter to (0,0,0,0).

lHandle

The list containing the cell specified by the parameter `theCell`.

Discussion
Because the List Manager automatically draws cells, few applications need to call this function directly.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`Lists.h`

LScroll

Scrolls a list a specified number of rows and columns. (Deprecated in Mac OS X v10.5.)

```
void LScroll (
   short dCols,
   short dRows,
   ListHandle lHandle
);
```

Parameters

dCols

The number of columns to scroll. Specify a positive number to scroll down (that is, each cell moves up), and a negative number to scroll up.

dRows

The number of rows to scroll. Specify a positive number to scroll right (that is, each cell moves left), and a negative number to scroll left.

lHandle

The list to be scrolled.

Discussion
The List Manager will not scroll beyond the data bounds of the list. If the automatic drawing mode is enabled, this function does all necessary updating of the list.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Lists.h`

LSearch

Finds a cell whose data matches data that you specify. (Deprecated in Mac OS X v10.5.)

```
Boolean LSearch (
   const void *dataPtr,
   short dataLen,
   ListSearchUPP searchProc,
   Cell *theCell,
   ListHandle lHandle
);
```

Parameters

dataPtr
A pointer to the data being searched for.

dataLen
The length in bytes of the data being searched for.

searchProc
A pointer to a function to be used to compare the data being searched for with cell data. If `NULL`, the Text Utilities Package function `IUMagIDString` is used.

If either the function pointed to by `searchProc` or `IUMagIDString` returns 0, `LSearch` has found a match; otherwise, it checks the next cell in the list.

theCell
A pointer to the first cell to be searched. If the function finds a match, this parameter is, on return, a pointer to the coordinates of the first cell whose data matches the data being searched for.

lHandle
The list to be searched.

Return Value
If the function finds a match, `TRUE`. Otherwise, `FALSE`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Lists.h`

LSetCell

Changes the data contained in a cell. (Deprecated in Mac OS X v10.5.)

```
void LSetCell (
   const void *dataPtr,
   short dataLen,
   Cell theCell,
   ListHandle lHandle
);
```

Parameters

dataPtr

A pointer to the new data for a cell.

dataLen

The length in bytes of the new data.

theCell

The coordinates of the cell to hold the new data.

lHandle

The list containing the cell given in the `theCell` parameter.

Discussion

Any previous cell data in `theCell` is replaced. If there is insufficient memory in the heap, the function may fail to set the cell's data. If the cell coordinates specified by the `theCell` parameter are invalid, the function does nothing.

If the data of a visible cell is changed and the automatic drawing mode is enabled, the function updates the list.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Lists.h`

LSetDrawingMode

Changes the automatic drawing mode specified when creating a list. (Deprecated in Mac OS X v10.5.)

```
void LSetDrawingMode (
   Boolean drawIt,
   ListHandle lHandle
);
```

Parameters

drawIt

Indicates whether the List Manager should enable the automatic drawing mode. Specify `TRUE` to enable the automatic drawing mode. Specify `FALSE` to disable the automatic drawing mode.

lHandle

The list whose drawing mode is being changed.

Discussion
Your application can use the `LSetDrawingMode` function to enable or disable automatic drawing of lists. If your application uses `LSetDrawingMode` to temporarily disable list drawing, then it must call the `LDraw` (page 1182) function to draw a cell when its appearance changes, or when new rows or columns are added to the list. .

While the automatic drawing mode is turned off, all cell drawing and highlighting are disabled, and the scroll bar does not function properly. Thus, your application should disable the automatic drawing mode only for short periods of time. After enabling it, your application should ensure that the list is redrawn.

This function is also available as the `LDoDraw` function.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Lists.h`

LSetSelect

Selects or deselects a cell. (Deprecated in Mac OS X v10.5.)

```
void LSetSelect (
   Boolean setIt,
   Cell theCell,
   ListHandle lHandle
);
```

Parameters

setIt
Indicates whether the function should select or deselect the specified cell. Specify `TRUE` to select the cell. If the cell is already selected, the function does nothing. Specify `FALSE` to deselect the cell. If the cell is already deselected, the function does nothing.

theCell
The cell to be selected or deselected.

lHandle
The list containing the cell to be selected or deselected.

Discussion
If a cell's selection status is changed and the cell is visible, `LSetSelect` redraws the cell.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Lists.h`

LSize

Changes the size of a list. (Deprecated in Mac OS X v10.5.)

```
void LSize (
   short listWidth,
   short listHeight,
   ListHandle lHandle
);
```

Parameters

listWidth

The new width (in pixels) of the list's visible rectangle.

listHeight

The new height (in pixels) of the list's visible rectangle.

lHandle

The list whose size is being changed.

Discussion

This function adjusts the lower-right side of the list so that the list's visible rectangle is the width and height specified by the `listWidth` and `listHeight` parameters.

Because the list's visible rectangle does not include room for the scroll bars, your application should make `listWidth` 15 pixels less than the desired width of the list if it contains a vertical scroll bar, and it should make `listHeight` 15 pixels less than the desired height of the list if it contains a horizontal scroll bar.

The contents of the list and the scroll bars are adjusted and redrawn as necessary. However, this function does not draw a border around the list's rectangle. Also, it does not erase any portions of the old list that may still be visible. This approach should not be a problem if your application only calls `LSize` after the user resizes a window containing a list in its lower-right corner.

Usually, you need to call this function only after calling the Window Manager function `SizeWindow`.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Lists.h`

LUpdate

Responds to an update event. (Deprecated in Mac OS X v10.5.)

```
void LUpdate (
   RgnHandle theRgn,
   ListHandle lHandle
);
```

Parameters

theRgn

The visible region of the list's port after a call to the Window Manager's `BeginUpdate` function.

lHandle
The list to be updated.

Discussion
This function redraws all visible cells in the list specified by the `lHandle` parameter that intersect the region specified by the parameter `theRgn`. It also redraws the scroll bars if they intersect the region.

You should bracket calls to `LUpdate` by calls to the Window Manager functions `BeginUpdate` and `EndUpdate`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Lists.h`

NewListClickLoopUPP

Creates a new universal procedure pointer (UPP) to a list click loop callback function. (Deprecated in Mac OS X v10.5.)

```
ListClickLoopUPP NewListClickLoopUPP (
   ListClickLoopProcPtr userRoutine
);
```

Parameters
userRoutine

Return Value
See the description of the `ListClickLoopUPP` data type.

Discussion
See the `ListClickLoopProcPtr` (page 1200) callback for more information.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`Lists.h`

NewListDefUPP

Creates a new universal procedure pointer (UPP) to a list definition callback function. (Deprecated in Mac OS X v10.5.)

```
ListDefUPP NewListDefUPP (
   ListDefProcPtr userRoutine
);
```

Parameters

userRoutine

Return Value
See the description of the `ListDefUPP` data type.

Discussion
See the `ListDefProcPtr` (page 1201) callback for more information.

Carbon Porting Notes

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`Lists.h`

NewListSearchUPP

Creates a new universal procedure pointer (UPP) to a list search callback function. (Deprecated in Mac OS X v10.5.)

```
ListSearchUPP NewListSearchUPP (
   ListSearchProcPtr userRoutine
);
```

Parameters

userRoutine

Return Value
See the description of the `ListSearchUPP` data type.

Discussion
See the `ListSearchProcPtr` (page 1204) callback for more information.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`Lists.h`

RegisterListDefinition

(Deprecated in Mac OS X v10.5.)

```
OSStatus RegisterListDefinition (
   SInt16 inResID,
   ListDefSpecPtr inDefSpec
);
```

Parameters

inResID

inDefSpec

Return Value
A result code.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Lists.h`

SetListCellIndent

(Deprecated in Mac OS X v10.5.)

```
void SetListCellIndent (
   ListHandle list,
   Point *indent
);
```

Parameters

list

indent

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Lists.h`

SetListClickLoop

(Deprecated in Mac OS X v10.5.)

```
void SetListClickLoop (
   ListHandle list,
   ListClickLoopUPP clickLoop
);
```

Parameters

list

clickLoop

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Lists.h`

SetListClickTime

(Deprecated in Mac OS X v10.5.)

```
void SetListClickTime (
   ListHandle list,
   SInt32 time
);
```

Parameters

list

time

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Lists.h`

SetListFlags

(Deprecated in Mac OS X v10.5.)

```
void SetListFlags (
   ListHandle list,
   OptionBits listFlags
);
```

Parameters

list

listFlags

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`Lists.h`

SetListLastClick

(Deprecated in Mac OS X v10.5.)

```
void SetListLastClick (
   ListHandle list,
   Cell *lastClick
);
```

Parameters

list

lastClick

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Lists.h`

SetListPort

(Deprecated in Mac OS X v10.5.)

```
void SetListPort (
   ListHandle list,
   CGrafPtr port
);
```

Parameters

list

port

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Lists.h`

SetListRefCon

(Deprecated in Mac OS X v10.5.)

```
void SetListRefCon (
   ListHandle list,
   SInt32 refCon
);
```

Parameters

list

refCon

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

Lists.h

SetListSelectionFlags

(Deprecated in Mac OS X v10.5.)

```
void SetListSelectionFlags (
   ListHandle list,
   OptionBits selectionFlags
);
```

Parameters

list

selectionFlags

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

Lists.h

SetListUserHandle

(Deprecated in Mac OS X v10.5.)

```
void SetListUserHandle (
   ListHandle list,
   Handle userHandle
);
```

Parameters

list

userHandle

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`Lists.h`

SetListViewBounds

(Deprecated in Mac OS X v10.5.)

```
void SetListViewBounds (
   ListHandle list,
   const Rect *view
);
```

Parameters

list

view

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Lists.h`

Callbacks

ListClickLoopProcPtr

Defines a pointer to a list click loop callback function. Your list click loop callback function overrides the standard click-loop function that is used to select cells and automatically scroll a list.

```
typedef Boolean (*ListClickLoopProcPtr)
(
);
```

If you name your function `MyListClickLoopProc`, you would declare it like this:

```
Boolean MyListClickLoopProc ();
```

Parameters

Return Value
A value indicating whether the `LClick` function should continue tracking the mouse. Your function should return `TRUE` if you wish `LClick` to continue to track the mouse, and `FALSE` if `LClick` should stop and return immediately.

Discussion
If your application defines a custom click-loop function, then the `LClick` (page 1179) function repeatedly calls the function until the user releases the mouse button. A click-loop function may perform any processing desired when it is executed.

Because no parameters are passed to the click-loop function, your click-loop function probably needs to access a global variable that contains a handle to the list record, which contains information about the location of the cursor and other information potentially of interest to a click-loop function. You might also create a global variable that stores the state of the modifier keys immediately before a call to the `LClick` function. You would need to set these global variables immediately before calling `LClick`.

The pointer to your function, which you provide in the list record structure, should be a universal procedure pointer (UPP). The definition of the UPP data type for your list click loop function is as follows:

```
typedef (ListClickLoopProcPtr) ListClickLoopUPP;
```

Before using your list click loop function, you must first create a new universal procedure pointer to it, using the `NewListClickLoopUPP` (page 1194) function, as shown here:

```
ListClickLoopUPP MyListClickLoopUPP;
MyListClickLoopUPP = NewListClickLoopUPP(&MyListClickLoopProc)
```

You then use `MyListClickLoopUPP` in the `lClickLoop` field of the `ListRec` (page 1209) structure for your list. The `LClick` (page 1179) function calls your list click loop function while the user holds down the mouse button. If you wish to call your own list click loop function, you can use the `InvokeListClickLoopUPP` (page 1173) function:

```
continueTracking = InvokeListClickLoopUPP(MyListClickLoopUPP);
```

When you are finished using your list click loop callback function, you should dispose of the universal procedure pointer associated with it, using the `DisposeListClickLoopUPP` (page 1164) function.

```
DisposeListClickLoopUPP(MyListClickLoopUPP);
```

A click-loop function does not execute at interrupt time. Instead, it is called directly by the `LClick` function. Thus, a click-loop function can allocate memory, and it does not need to adjust the value contained in the A5 register.

Special Considerations

A click-loop function does not execute at interrupt time. Instead, it is called directly by the `LClick` function. Thus, a click-loop function can allocate memory, and it does not need to adjust the value contained in the A5 register.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Lists.h`

ListDefProcPtr

Defines a pointer to a list definition callback function. Your list definition callback function defines a custom list display.

```
typedef void (*ListDefProcPtr) (
    SInt16 lMessage,
    Boolean lSelect,
    Rect *lRect,
    Cell lCell,
    SInt16 lDataOffset,
    SInt16 lDataLen,
    ListHandle lHandle
);
```

If you name your function `MyListDefProc`, you would declare it like this:

```
void MyListDefProc (
    SInt16 lMessage,
    Boolean lSelect,
    Rect * lRect,
    Cell lCell,
    SInt16 lDataOffset,
    SInt16 lDataLen,
    ListHandle lHandle
);
```

Parameters

lMessage

A value that identifies the operation to be performed. See "List Definition Constants" (page 1214).

lSelect

Indicates whether the cell specified by the `lCell` parameter should be highlighted. This parameter is defined only for the `lDrawMessage` and `lHiliteMsg` messages.

lRect

A pointer to the rectangle (in local coordinates of the list's graphics port) that encloses the specified cell. Although this parameter is defined as a pointer, your list definition function must not change the coordinates of the rectangle. This parameter is defined only for the `lDrawMessage` and `lHiliteMsg` messages.

lCell

The coordinates of the cell to be drawn or highlighted. This parameter is defined only for the `lDrawMessage` and `lHiliteMsg` messages.

lDataOffset

The location of the cell data associated with the specified cell. The location is specified as an offset from the beginning of the relocatable block referenced by the `cells` field of the list record. This parameter is defined only for the `lDrawMessage` and `lHiliteMsg` messages.

lDataLen

The length in bytes of the cell data associated with the specified. This parameter is defined only for the `lDrawMessage` and `lHiliteMsg` messages.

lHandle

A handle to the list for which a message is being sent. Your application can access the list's list record, or it can call List Manager functions to manipulate the list.

Discussion

Your application can write a list definition function to customize list display. For example, you can write a list definition function to support the display of color icons. A custom list definition function must be compiled as a code resource of type `'LDEF'` and added to the resource file of the application that needs to use it.

The List Manager calls your list definition function whenever an application using the function creates a new list with the `LNew` (page 1186) function, needs a cell to be drawn, needs a cell's highlighting state to be reversed, or has called the `LDispose` (page 1181) function to dispose of a list.

The pointer to your list definition function should be a universal procedure pointer (UPP). The definition of the UPP data type for your definition function is as follows:

```
typedef (ListDefProcPtr) ListDefUPP;
```

Before using your list definition function, you must first create a new universal procedure pointer to it, using the `NewListDefUPP` (page 1194) function, as shown here:

```
ListDefUPP MyListDefUPP;
MyListDefUPP = NewListDefUPP(&MyListDefProc)
```

The List Manager automatically invokes your list definition function when a new list is created. If you wish to call your own list definition callback function, you can use the `InvokeListDefUPP` (page 1173) function:

```
InvokeListDefUPP(lMessage, lSelect, &lRect, lCell, lDataOffset,
       lDataLen, lHandle, MyListDefUPP)
```

When you are finished with your list definition function, you should dispose of the universal procedure pointer associated with it, using the `DisposeListDefUPP` (page 1165) function.

```
DisposeListDefUPP(MyListDefUPP);
```

Because a list definition function is stored in a code resource, it cannot have its own global variables that it accesses through the A5 register. (Some development systems, however, may allow code resources to access global variables through some other register, such as A4. See your development system's documentation for more information.) If your list definition function needs access to global data, it might store a handle to such data in the `refCon` or `userHandle` fields of the list record; however, applications would not then be able to use these fields for their own purposes.

Special Considerations

Because a list definition function is stored in a code resource, it cannot have its own global variables that it accesses through the A5 register. (Some development systems, however, may allow code resources to access global variables through some other register, such as A4. See your development system's documentation for more information.) If your list definition function needs access to global data, it might store a handle to such data in the `refCon` or `userHandle` fields of the list record; however, applications would not then be able to use these fields for their own purposes.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Lists.h`

ListNotificationProcPtr

```
typedef void (*ListNotificationProcPtr)
(
    ListHandle theList,
    ListNotification notification,
    SInt32 param
);
```

If you name your function `MyListNotificationProc`, you would declare it like this:

```
void MyListNotificationProc (
    ListHandle theList,
    ListNotification notification,
    SInt32 param
);
```

Parameters

theList

notification

param

ListSearchProcPtr

Defines a pointer to a list search callback function. Your list search callback function compares data in a search field to the contents of a list cell.

```
typedef SInt16 (*ListSearchProcPtr) (
    Ptr aPtr,
    Ptr bPtr,
    SInt16 aLen,
    SInt16 bLen
);
```

If you name your function `MyListSearchProc`, you would declare it like this:

```
short MyListSearchProc (
    Ptr aPtr,
    Ptr bPtr,
    short aLen,
    short bLen
);
```

Parameters

aPtr

A pointer to the data contained in a cell.

bPtr

A pointer to the data for which you are searching.

aLen

The number of bytes of data contained in the cell.

bLen

The number of bytes of data for which you are searching.

Return Value
If the cell data matches the search data, your function should return 0. Otherwise, your search function should return 1.

Discussion
You can pass a pointer to your search function as the third parameter to the `LSearch` function. A search function must compare the data defined by the `aPtr` and `aLen` parameters with the data defined by the `bPtr` and `bLen` parameters. Your function can use any technique you choose to compare the data.

If you do not wish to create your own search function, your application can specify `NULL` as a parameter to `LSearch`, in place of a pointer to your function. `LSearch` then uses the Text Utilities function `IUMagIDString`, the default search function. The `IUMagIDString` function returns 0 if the search data exactly matches the cell data, but `IUMagIDString` considers the strings `'Rose'` and `'rosé'` to be equivalent. If your application simply needs a search function that works like `IUMagIDString` but considers `'Rose'` to be different from `'rosé'`, the Text Utilities provides the case-sensitive comparison function `IUMagString`. Instead of writing a custom function, your application can simply pass `@IUMagString` as the third parameter to the `LSearch` function.

The pointer which you pass to the `LSearch` function should be a universal procedure pointer (UPP). The definition of the UPP data type for your search function is as follows:

```
typedef (ListSearchProcPtr) ListSearchUPP;
```

Before using your search function, you must first create a universal procedure pointer to it, using the `NewListSearchUPP` `NewListSearchUPP` (page 1195) function, as shown here:

```
ListSearchUPP MyListSearchUPP;
MyListSearchUPP = NewListSearchUPP(&MyListSearchProc)
```

You then pass `MyListSearchUPP` to the `LSearch` function, which will call your custom search function on each cell it searches. If you wish to call your own list search function, use the `InvokeListSearchUPP` (page 1174) function:

```
isMatch = InvokeListSearchUPP(aPtr, bPtr, aLen, bLen,
                    MyListSearchUPP);
```

When you are finished with your list search callback function, you should dispose of the universal procedure pointer associated with it, using the `DisposeListSearchUPP` (page 1165) function:

```
DisposeListSearchUPP(MyListSearchUPP);
```

A search function does not execute at interrupt time. Instead, it is called directly by the `LSearch` function. Thus, a search function can allocate memory, and it does not need to adjust the value contained in the A5 register.

Special Considerations

A search function does not execute at interrupt time. Instead, it is called directly by the `LSearch` function. Thus, a search function can allocate memory, and it does not need to adjust the value contained in the A5 register.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Lists.h`

Data Types

Cell

```
typedef Point Cell;
```

Discussion
The `Cell` data type defines a cell record. The functions `LGetSelect` (page 1184), `LSetSelect` (page 1192), `LSetCell` (page 1190), `LAddToCell` (page 1177), `LClrCell` (page 1179), `LGetCellDataLocation` (page 1183), `LGetCell` (page 1183), `LDraw` (page 1182), `LSearch` (page 1190), `LNextCell` (page 1187), `LRect` (page 1188), and `LLastClick` (page 1185) use the `Cell` data type to specify the coordinates of a cell in a list.

Note that column and row numbers are 0-based. Also note that this reference designates cells using the notation (column–1, row–1), so that a cell with coordinates (2,5) is in the third column and sixth row of a list. You specify a cell with coordinates (2,5) by setting the cell's `h` field to 2 and its `v` field to 5.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Lists.h`

DataArray

```
typedef  DataArray[32001];
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Lists.h`

DataHandle

```
typedef  DataPtr * DataHandle;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Lists.h`

DataPtr

```
typedef  char * DataPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
Lists.h

ListBounds

```
typedef Rect ListBounds;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
Lists.h

ListClickLoopUPP

```
typedef ListClickLoopProcPtr ListClickLoopUPP;
```

Discussion
For more information, see the description of the ListClickLoopUPP () callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
Lists.h

ListDefSpec

```
struct ListDefSpec {
    ListDefType defType
    union {
        ListDefUPP userProc;
    } u;
};
typedef struct ListDefSpec ListDefSpec;
typedef ListDefSpec * ListDefSpecPtr;
```

Fields
defType
ListDefUPP

Availability
Available in Mac OS X v10.0 and later.

Declared In
Lists.h

ListDefType

```
typedef UInt32 ListDefType;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Lists.h`

ListDefUPP

```
typedef ListDefProcPtr ListDefUPP;
```

Discussion
For more information, see the description of the ListDefUPP () callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Lists.h`

ListNotification

```
typedef SInt32 ListNotification;
```

ListNotificationUPP

```
typedef ListNotificationProcPtr ListNotificationUPP;
```

ListRec

```
struct ListRec {
    Rect rView;
    GrafPtr port;
    Point indent;
    Point cellSize;
    ListBounds visible;
    ControlRef vScroll;
    ControlRef hScroll;
    SInt8 selFlags;
    Boolean lActive;
    SInt8 lReserved;
    SInt8 listFlags;
    long clikTime;
    Point clikLoc;
    Point mouseLoc;
    ListClickLoopUPP lClickLoop;
    Cell lastClick;
    long refCon;
    Handle listDefProc;
    Handle userHandle;
    ListBounds dataBounds;
    DataHandle cells;
    short maxIndex;
    short cellArray[1];
};
typedef struct ListRec ListRec;
typedef ListRec * ListPtr;
typedef ListPtr * ListHandle
```

Fields

`rView`
: The rectangle in which the list's visible rectangle is located, in local coordinates of the graphics port specified by the `port` field. Note that the list's visible rectangle does not include the area needed for the list's scroll bars. The width of a vertical scroll bar (which equals the height of a horizontal scroll bar) is 15 pixels.

`port`
: The graphics port of the window containing the list.

`indent`
: The location, relative to the upper-left corner of a cell, at which drawing should begin. List definition functions should set this field to a value appropriate to the type of data that a cell in a list is to contain.

`cellSize`
: The size in pixels of each cell in the list. When your application creates a list, it can either specify the cell size or let the List Manager calculate the cell size. You should not change the `cellSize` field directly; if you need to change the cell size after creating a list, use the `LCellSize` (page 1178) function.

`visible`
: The cells in a list that are visible within the area specified by the `rView` field. The List Manager sets the `left` and `top` fields of `visible` to the coordinates of the first visible cell; however, the List Manager sets the `right` and `bottom` fields so that each is 1 greater than the horizontal and vertical coordinates of the last visible cell. For example, if a list contains 4 columns and 10 rows but only the first 2 columns and the first 5 rows are visible (that is, the last visible cell has coordinates (1,4)), the List Manager sets the `visible` field to (0,0,2,5).

`vScroll`
: A control handle for a list's vertical scroll bar, or `NULL` if a list does not have a vertical scroll bar.

`hScroll`
: A control handle for a list's horizontal scroll bar, or `NULL` if a list does not have a horizontal scroll bar.

`selFlags`
: Indicates the selection flags for a list. When your application creates a list, the List Manager clears the `selFlags` field to 0. This defines the List Manager's default selection algorithm. To change the default behavior for a particular list, set the desired bits in the list's `selFlags` field. See "Selection Flags" (page 1217).

`lActive`
: Indicates whether the list is active (`TRUE` if active, `FALSE` if inactive).

`lReserved`
: Reserved.

`listFlags`
: Indicates whether the List Manager should automatically scroll the list if the user clicks the list and then drags the cursor outside the list display rectangle. See "List Flags" (page 1215) for the values used in this field.

  By default, the List Manager enables horizontal autoscrolling for a list if the list includes a horizontal scroll bar, and enables vertical autoscrolling for a list if the list includes a vertical scroll bar.

`clikTime`
: The time in ticks of the last click in the list. If your application depends on the value contained in this field, then your application should update the field if the application selects a list item in response to keyboard input.

`clikLoc`
: The location in local coordinates of the last click in the list.

`mouseLoc`
: Indicates the current location of the cursor in local coordinates. This value is continuously updated by the `LClick` function after the user clicks a list.

`lClickLoop`
: A universal procedure pointer to your click loop callback function, which is repeatedly called by the `LClick` (page 1179) function, or NULL if the default click-loop function is to be used.

`lastClick`
: The coordinates of the last cell in the list that was clicked. This may not be the same as the last cell selected if the user selects a range of cells by Shift-dragging or Command-dragging. If your application depends on the value contained in this field, then your application should update the field whenever your application selects a list item in response to keyboard input.

`refCon`
: 4 bytes for use by your application.

`listDefProc`
: A handle to the code for the list definition function that defines how the list is drawn.

`userHandle`
: 4 bytes that your application can use as needed. For example, your application might use this field to store a handle to additional storage associated with the list. However, the `LDispose` (page 1181) function does not automatically release this storage when disposing of the list.

`dataBounds`
: The range of cells in a list. When your application creates a list, it specifies the initial bounds of the list. As your application adds rows and columns, the List Manager updates this field. The List Manager sets the `left` and `top` fields of `dataBounds` to the coordinates of the first cell in the list; the List Manager sets the `right` and `bottom` fields so that each is 1 greater than the horizontal and vertical coordinates of the last cell. For example, if a list contains 4 columns and 10 rows (that is, the last cell in the list has coordinates (3,9)), the List Manager sets the `dataBounds` field to (0,0,4,10).

`cells`
: A handle to a relocatable block used to store cell data. Your application should not change the contents of this relocatable block directly.

`maxIndex`
: Used internally.

`cellArray`
: Offsets to data that indicate the location of different cells' data within the data handle specified by the `cells` parameter. Your application should not access this field directly.

Discussion
Functions in the List Manager interface use the `ListHandle` datatype to identify a list. The `ListHandle` type uses a `ListRec` structure to maintain information about a list. The `ListRec` data type defines a list record.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Lists.h`

ListRef

```
typedef ListHandle ListRef;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Lists.h`

ListSearchUPP

```
typedef ListSearchProcPtr ListSearchUPP;
```

Discussion
For more information, see the description of the ListSearchUPP () callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Lists.h`

StandardIconListCellDataRec

```
struct StandardIconListCellDataRec {
    Handle iconHandle;
    short font;
    short face;
    short size;
    Str255 name;
};
typedef struct StandardIconListCellDataRec StandardIconListCellDataRec;
typedef StandardIconListCellDataRec * StandardIconListCellDataPtr;
```

Fields
`iconHandle`
`font`
`face`
`size`
`name`

Discussion

Version Notes

Carbon Porting Notes

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Lists.h`

Constants

kListDefProcPtr

```
enum {
    kListDefProcPtr = 0,
    kListDefUserProcType = kListDefProcPtr,
    kListDefStandardTextType = 1,
    kListDefStandardIconType = 2
};
```

Constants

`kListDefProcPtr`
Available in Mac OS X v10.0 and later.
Declared in `Lists.h`.

`kListDefUserProcType`
Available in Mac OS X v10.0 and later.
Declared in `Lists.h`.

`kListDefStandardTextType`
Available in Mac OS X v10.0 and later.
Declared in `Lists.h`.

`kListDefStandardIconType`
Available in Mac OS X v10.0 and later.
Declared in `Lists.h`.

lDrawingModeOff

```
enum {
    lDrawingModeOff = 8,
    lDoVAutoscroll = 2,
    lDoHAutoscroll = 1
};
```

Constants

`lDrawingModeOff`
Available in Mac OS X v10.0 and later.
Declared in `Lists.h`.

`lDoVAutoscroll`
Set this bit to 1 if you wish to allow automatic vertical scrolling.
Available in Mac OS X v10.0 and later.
Declared in `Lists.h`.

`lDoHAutoscroll`
Set this bit to 1 if you wish to allow automatic horizontal scrolling.
Available in Mac OS X v10.0 and later.
Declared in `Lists.h`.

lDrawingModeOffBit

```
enum {
    lDrawingModeOffBit = 3,
    lDoVAutoscrollBit = 1,
    lDoHAutoscrollBit = 0
};
```

Constants

`lDrawingModeOffBit`
Available in Mac OS X v10.0 and later.
Declared in `Lists.h`.

`lDoVAutoscrollBit`
Available in Mac OS X v10.0 and later.
Declared in `Lists.h`.

`lDoHAutoscrollBit`
Available in Mac OS X v10.0 and later.
Declared in `Lists.h`.

List Definition Constants

```
enum {
    lInitMsg = 0,
    lDrawMsg = 1,
    lHiliteMsg = 2,
    lCloseMsg = 3
};
```

Constants

`lInitMsg`
In response to the `lInitMsg` message, your list definition function should perform any special initialization needed for a list. For example, the function might set fields of the list record, such as the `cellSize` and `indent` fields, to appropriate values. Your list definition function does not necessarily need to do anything in response to the initialization message. If it does nothing, then memory is still allocated for the list, and fields of the list record are set to the same values as they would be set to if the default list definition function were being used.
Available in Mac OS X v10.0 and later.
Declared in `Lists.h`.

`lDrawMsg`
Your list definition function should draw the cell specified by the `theCell` parameter after receiving an `lDrawMsg` message. The function must ensure that it does not draw anywhere but within the rectangle specified by the `cellRect` parameter. If the `selected` parameter is `TRUE`, then your list definition function should draw the cell in its highlighted state; otherwise, it should draw the cell without highlighting. When drawing, your list definition function should take care not to permanently change any characteristics of the drawing environment.
Available in Mac OS X v10.0 and later.
Declared in `Lists.h`.

`lHiliteMsg`

Your list definition function should respond to the `lHiliteMsg` message by reversing the selection status of the cell contained within the rectangle specified by the `cellRect` parameter. If a cell is highlighted, your list definition function should remove the highlighting; if a cell is not highlighted, your list definition function should highlight it.

Available in Mac OS X v10.0 and later.

Declared in `Lists.h`.

`lCloseMsg`

The List Manager sends your list definition function an `lCloseMsg` message before it disposes of a list and its data. Your list definition function need only respond to this message if additional memory has been allocated for the list. For example, your list definition function might allocate a relocatable block in response to the `lInitMsg` message. In this case, your list definition function would need to dispose of this relocatable block in response to the `lCloseMsg` message. Or, if your list definition function defines cells simply to contain pointers or handles to data stored elsewhere in memory, it would need to dispose of that memory in response to the `lCloseMsg` message.

Available in Mac OS X v10.0 and later.

Declared in `Lists.h`.

Discussion
The List Manager passes these values to your `ListDefProcPtr` (page 1201) function to identify the operation to be performed.

List Flags

Constants

Discussion
The following constants define bits in the `listFlags` field of the `ListRec` (page 1209) structure that determine whether horizontal autoscrolling and vertical autoscrolling are enabled:.

listNotifyNothing

```
enum {
    listNotifyNothing = 'nada',
    listNotifyClick = 'clik',
    listNotifyDoubleClick = 'dblc',
    listNotifyPreClick = 'pclk'
};
```

Constants

`listNotifyNothing`

`listNotifyClick`

`listNotifyDoubleClick`

`listNotifyPreClick`

lOnlyOneBit

```
enum {
    lOnlyOneBit = 7,
    lExtendDragBit = 6,
    lNoDisjointBit = 5,
    lNoExtendBit = 4,
    lNoRectBit = 3,
    lUseSenseBit = 2,
    lNoNilHiliteBit = 1
};
```

Constants

`lOnlyOneBit`
Available in Mac OS X v10.0 and later.
Declared in `Lists.h`.

`lExtendDragBit`
Available in Mac OS X v10.0 and later.
Declared in `Lists.h`.

`lNoDisjointBit`
Available in Mac OS X v10.0 and later.
Declared in `Lists.h`.

`lNoExtendBit`
Available in Mac OS X v10.0 and later.
Declared in `Lists.h`.

`lNoRectBit`
Available in Mac OS X v10.0 and later.
Declared in `Lists.h`.

`lUseSenseBit`
Available in Mac OS X v10.0 and later.
Declared in `Lists.h`.

`lNoNilHiliteBit`
Available in Mac OS X v10.0 and later.
Declared in `Lists.h`.

Selection Flags

```
enum {
    lOnlyOne = -128,
    lExtendDrag = 64,
    lNoDisjoint = 32,
    lNoExtend = 16,
    lNoRect = 8,
    lUseSense = 4,
    lNoNilHilite = 2
};
```

Constants

`lOnlyOne`
Specify this value if you wish to allow only one item to be selected at once.
Available in Mac OS X v10.0 and later.
Declared in `Lists.h`.

`lExtendDrag`
Specify this value if you wish to enable selection of multiple items by dragging without the Shift key.
Available in Mac OS X v10.0 and later.
Declared in `Lists.h`.

`lNoDisjoint`
Specify this value if you wish to prevent discontinuous selections using the Command key.
Available in Mac OS X v10.0 and later.
Declared in `Lists.h`.

`lNoExtend`
Specify this value if you wish to prevent extending Shift key selections. All items are deselected before responding to Shift-click.
Available in Mac OS X v10.0 and later.
Declared in `Lists.h`.

`lNoRect`
Specify this value if you wish to select all items in the cursor's path during Shift-drag.
Available in Mac OS X v10.0 and later.
Declared in `Lists.h`.

`lUseSense`
Specify this value if you wish to allow the user to deselect one or more items using the Shift key
Available in Mac OS X v10.0 and later.
Declared in `Lists.h`.

`lNoNilHilite`
Specify this value if you wish to disable the highlighting of empty cells.
Available in Mac OS X v10.0 and later.
Declared in `Lists.h`.

Discussion
The `ListRec` (page 1209) structure uses these values in the `selFlags` field to indicate the List Manager's default selection algorithm. Use these values additively to select more than one selection option.

Menu Manager Reference

Framework: Carbon/Carbon.h

Declared in Menus.h

Overview

You can use the Menu Manager to create and manage the menus in your application. Menus allow the user to view or choose from a list of choices and commands that your application provides. All Mac OS applications should provide these standard menus: the Application menu, the File menu, and the Edit menu.

Carbon supports the Menu Manager, with the following changes:

- Your application must use the functions defined by the Menu Manager whenever it creates and disposes of Menu Manager data structures. Some applications, for example, create menus by using the Resource Manager function `GetResource` (instead of the Menu Manager function `GetMenu`) and dispose of them by calling `ReleaseResource` instead of `DisposeMenu`. This practice is not supported in Carbon.
- The `MenuInfo` structure is opaque in Carbon. You must revise your application so that it accesses Menu Manager data structures only through accessor functions.
- Menu color information tables are not recommended in Carbon.
- You are strongly encouraged to adopt the standard Mac OS menu definition functions (also known as MDEFs) in your application. Your menus will then inherit the systemwide appearance and, furthermore, take advantage of other Menu Manager enhancements planned for the future. If you decide to customize the appearance of a menu and you are deploying your application for Mac OS X v10.3 and later, you should use a custom HIView instead of a custom menu definition function.

Functions by Task

Function descriptions are grouped by the tasks for which you use the functions. For an alphabetical list of functions, go to the API index at the end of the document.

Creating and Disposing of Menus

`CreateNewMenu` (page 1246)
Creates a new, untitled, empty menu.

`DuplicateMenu` (page 1255)
Creates a new menu that is a copy of another menu.

`DisposeMenu` (page 1253)
Decrements the retain count of a menu.

`CreateCustomMenu` (page 1245)
Creates a new, untitled, empty menu using a custom menu definition function.

`RegisterMenuDefinition` (page 1318)
Registers a binding between a resource ID and a menu definition function.

`SetMenuDefinition` (page 1329)
Sets the menu definition structure for a menu.

`GetMenuDefinition` (page 1271)
Obtains the menu definition structure for a menu.

`CreateStandardFontMenu` (page 1247)
Creates a standard font menu.

`UpdateStandardFontMenu` (page 1349)
Updates a standard Font menu.

`ReleaseMenu` (page 1319) Deprecated in Mac OS X v10.5
Decrements the retain count of a menu.

`RetainMenu` (page 1321) Deprecated in Mac OS X v10.5
Increments the reference count of a menu.

Manipulating the Root Menu

`AcquireRootMenu` (page 1229)
Get the menu whose contents are displayed in the menubar.

`SetRootMenu` (page 1347)
Sets the menu whose contents are displayed in the menubar.

Manipulating the Menu Bar

`HideMenuBar` (page 1291)
Conceals the menu bar.

`ShowMenuBar` (page 1348)
Displays the menu bar.

`IsMenuBarVisible` (page 1303)
Reports whether the menu bar is currently visible.

`FlashMenuBar` (page 1259)
Highlights a menu title or the entire menu bar.

`SetMenuBar` (page 1327)
Sets the current menu list to a specified menu list.

`GetMenuBar` (page 1268)
Gets a handle to a copy of the current menu list.

`DrawMenuBar` (page 1255)
Draws the menu bar based on the current menu list.

`InvalMenuBar` (page 1301)
Invalidates the menu bar.

`GetMBarHeight` (page 1265)
Determines the current height of the menu bar.

`DuplicateMenuBar` (page 1256)
Duplicates a menubar handle.

`DisposeMenuBar` (page 1253)
Releases a menubar handle.

Adding and Removing Menus

`InsertMenu` (page 1294)
Inserts an existing menu into the current menu list.

`DeleteMenu` (page 1248)
Deletes an existing menu from the current menu list.

`ClearMenuBar` (page 1237)
Deletes all menus from the current menu list.

`GetMenuRetainCount` (page 1286) Deprecated in Mac OS X v10.5
Returns the retain count of this menu.

Manipulating and Accessing Menu Characteristics

`GetMenuRef` (page 1286)
Obtains a menu reference corresponding to a menu ID.

`GetMenuHandle` (page 1273)
Obtains a menu reference corresponding to a menu ID.

`SetMenuTitleWithCFString` (page 1346)
Sets the title of a menu to the text contained in a CFString.

`CopyMenuTitleAsCFString` (page 1243)
Returns a CFString containing the title of a menu.

`GetMenuType` (page 1289)
Gets the display type (pulldown, hierarchical, or popup) of a menu.

`SetMenuTitleIcon` (page 1345)
Sets the title of a menu to be an icon.

`GetMenuTitleIcon` (page 1288)
Retrieves the icon, if any, being used as the title of a menu.

`SetMenuID` (page 1333)
Assigns a menu ID to a menu.

`GetMenuID` (page 1274)
Obtains the ID of a menu.

`LMGetTheMenu` (page 1308)
Returns the menu ID of the currently highlighted menu in the menu bar.

`SetMenuHeight` (page 1332)
Set the height of a menu.

`GetMenuHeight` (page 1274)
Obtains the height of a menu, in pixels.

`SetMenuWidth` (page 1347)
Sets the width of a menu.

`GetMenuWidth` (page 1289)
Obtains the width of the menu, in pixels.

`CalcMenuSize` (page 1234)
Recalculates the horizontal and vertical dimensions of a menu.

`IsMenuSizeInvalid` (page 1306)
Determines if a menu's size is invalid and should be recalculated.

`GetMenuAttributes` (page 1268)
Gets the attributes of a menu.

`ChangeMenuAttributes` (page 1235)
Changes the attributes of a menu.

`IsValidMenu` (page 1308)
Determines if a menu is valid.

`SetMenuFlashCount` (page 1331) Deprecated in Mac OS X v10.5
Specifies whether a menu should fade slowly or immediately disappear when closing.

Drawing Menus and Menu Items

`HiliteMenu` (page 1291)
Highlights or unhighlights menu titles.

`UpdateInvalidMenuItems` (page 1348)
Redraws the invalid items of an open menu.

`InvalidateMenuEnabling` (page 1299)
Requests that the menu's enable state be recalculated.

`InvalidateMenuItems` (page 1299)
Invalidates a group of menu items so that they will be redrawn when `UpdateInvalidMenuItems` is next called.

`InvalidateMenuSize` (page 1300)
Invalidates the menu size so that it will be recalculated when next displayed.

`IsMenuBarInvalid` (page 1302)
Determines if the menubar is invalid and should be redrawn.

`IsMenuItemInvalid` (page 1305)
Determines if a menu item is invalid and should be redrawn.

`EraseMenuBackground` (page 1258) Deprecated in Mac OS X v10.5
Erases the menu background to prepare for additional drawing.

`ScrollMenuImage` (page 1322) Deprecated in Mac OS X v10.5
Scrolls a portion of the menu image.

Adding and Deleting Menu Items

`CopyMenuItems` (page 1242)
Copies menu items from one menu to another.

`DeleteMenuItem` (page 1249)
Deletes an item from a menu.

`DeleteMenuItems` (page 1250)
Deletes multiple menu items.

Associating Data With Menu Items

`SetMenuItemProperty` (page 1341)
Associates data with a menu item.

`SetMenuCommandProperty` (page 1329)
Sets the property data for a menu item with the specified command ID.

`GetMenuItemProperty` (page 1281)
Obtains a piece of data that has been previously associated with a menu item.

`GetMenuCommandProperty` (page 1270)
Retrieves property data for a menu item with the specified command ID.

`GetMenuItemPropertySize` (page 1283)
Obtains the size of a piece of data that has been previously associated with a menu item.

`GetMenuCommandPropertySize` (page 1271)
Retrieves the size of the property data for a menu item with the specified command ID.

`RemoveMenuItemProperty` (page 1320)
Removes a piece of data that has been previously associated with a menu item.

`RemoveMenuCommandProperty` (page 1319)
Removes a property from a menu item with the specified command ID.

`GetMenuItemPropertyAttributes` (page 1282)
Gets the attributes of a menu item property.

`ChangeMenuItemPropertyAttributes` (page 1236)
Changes the attributes of a menu item property.

`SetMenuItemRefCon` (page 1342)
Sets application-specific information for a menu item.

`GetMenuItemRefCon` (page 1284)
Obtains application-specific information for a menu item.

Enabling Menus and Menu Items

`CountMenuItems` (page 1244)
Obtains the number of menu items in a menu

`CountMenuItemsWithCommandID` (page 1244)
Counts the menu items with a specified command ID.

`EnableMenuItem` (page 1257)
Enables a menu item or a menu.

`EnableMenuCommand` (page 1257)
Enables the menu item with a specified command ID.

`DisableMenuItem` (page 1251)
Disables a menu item or a menu.

`DisableMenuCommand` (page 1251)
Disables the menu item with a specified command ID.

`EnableAllMenuItems` (page 1256)
Enables all items in a menu.

`DisableAllMenuItems` (page 1250)
Disables all items in a menu.

`IsMenuItemEnabled` (page 1303)
Reports whether a given menu or menu item is enabled.

`IsMenuCommandEnabled` (page 1303)
Determines if the menu item with a specified command ID is enabled.

`MenuHasEnabledItems` (page 1310)
Determines if any items in a menu are enabled.

`EnableMenuItemIcon` (page 1258)
Enables the icon associated with a menu item.

`DisableMenuItemIcon` (page 1252)
Disables the icon associated with a menu item.

`IsMenuItemIconEnabled` (page 1304)
Reports whether a given menu item icon is enabled.

Manipulating Menu Item Text

`SetMenuItemTextWithCFString` (page 1344)
Sets the text of a menu item to the text contained in a CFString.

`CopyMenuItemTextAsCFString` (page 1243)
Returns a CFString containing the text of a menu item.

`AppendMenuItemTextWithCFString` (page 1231)
Appends a new menu item with text from a CFString.

`InsertMenuItemTextWithCFString` (page 1296)
Inserts a new menu item with text from a CFString.

`GetMenuFont` (page 1272)
Obtains the font used in a menu.

`SetMenuFont` (page 1331)
Sets the font to be used in a menu.

`GetMenuItemFontID` (page 1277)
Obtains a menu item's font ID.

`SetMenuItemFontID` (page 1335)
Sets the font for a menu item.

`GetFontFamilyFromMenuSelection` (page 1260)
Gets the font family reference and style from the menu identifier and menu item number returned by the function `MenuSelect`.

`SetItemStyle` (page 1325)
Sets a menu item's text style.

`GetItemStyle` (page 1264)
Returns a menu item's text style.

`GetMenuItemTextEncoding` (page 1285) Deprecated in Mac OS X v10.5
Obtains the text encoding used for a menu item's text.

`SetMenuItemTextEncoding` (page 1343) Deprecated in Mac OS X v10.5
Sets the text encoding for a menu item's text.

Manipulating and Accessing Menu Item Characteristics

`SetMenuItemIconHandle` (page 1338)
Sets a menu item's icon.

`GetMenuItemIconHandle` (page 1279)
Obtains a handle to a menu item's icon.

`SetItemMark` (page 1324)
Sets the mark of a menu item.

`SetMenuCommandMark` (page 1328)
Locates the menu item with a specified command ID and sets its mark character.

`GetItemMark` (page 1263)
Returns a menu item's mark.

`GetMenuCommandMark` (page 1269)
Locates the menu item with a specified command ID and returns its mark character.

`CheckMenuItem` (page 1237)
Adds or removes a check mark from a menu item.

`SetMenuItemIndent` (page 1338)
Sets the indent level of a menu item.

`GetMenuItemIndent` (page 1279)
Gets the indent level of a menu item.

`SetMenuItemKeyGlyph` (page 1339)
Sets the command key glyph code for a menu item.

`GetMenuItemKeyGlyph` (page 1280)
Obtains the keyboard glyph for a menu item's keyboard equivalent.

`SetMenuItemModifiers` (page 1340)
Sets the modifier key(s) that must be pressed with a character key to select a particular menu item.

`GetMenuItemModifiers` (page 1281)
Obtains the modifier keys that must be pressed with a character key to select a particular menu item.

`SetMenuItemCommandID` (page 1333)
Sets a menu item's command ID.

`GetMenuItemCommandID` (page 1275)
Obtains a menu item's command ID.

`SetMenuItemHierarchicalMenu` (page 1337)
Attaches a submenu to a menu item.

`GetMenuItemHierarchicalMenu` (page 1278)
Returns the submenu attached to a menu item.

`SetMenuItemCommandKey` (page 1334)
Sets the keyboard equivalent of a menu item.

`GetMenuItemCommandKey` (page 1276)
Gets the keyboard equivalent of a menu item.

`GetIndMenuItemWithCommandID` (page 1261)
Finds a menu item with a specified command ID.

`SetMenuItemData` (page 1335)
Sets multiple attributes of a menu item at once.

`CopyMenuItemData` (page 1241)
Obtains multiple menu item attributes at once.

`GetMenuItemAttributes` (page 1275)
Gets the attributes of a menu item.

`ChangeMenuItemAttributes` (page 1235)
Changes the attributes of a menu item.

`GetItemIcon` (page 1262) Deprecated in Mac OS X v10.5
Returns a menu item's icon or text encoding.

`GetMenuItemHierarchicalID` (page 1277) Deprecated in Mac OS X v10.5
Obtains the menu ID of a specified submenu.

`SetItemIcon` (page 1323) Deprecated in Mac OS X v10.5
Sets a menu item's icon or text encoding.

`SetMenuItemHierarchicalID` (page 1336) Deprecated in Mac OS X v10.5
Attaches a submenu to a menu item.

Responding to Menu Events and User Selections

`IsMenuKeyEvent` (page 1305)
Determines if an event corresponds to a menu command key.

`GetMenuTrackingData` (page 1288)
Gets information about the menu currently selected by the user.

`CancelMenuTracking` (page 1234)
Cancels menu tracking.

`MenuSelect` (page 1312)
Allows the user to choose a menu item from a menu in the menu bar.

`MenuChoice` (page 1309)
Returns the menu ID and index of the menu item under the cursor.

`IsShowContextualMenuEvent` (page 1307)
Determines whether a particular Carbon event could invoke a contextual menu.

`ContextualMenuSelect` (page 1239)
Displays a contextual menu.

PopUpMenuSelect (page 1316)
Displays a pop-up menu without using the standard pop-up control definition function.

Handling Contextual Menu Plugins

CMPluginExamineContext (page 1238)
An application-defined callback function that examines the context in a contextual menu CFPlugin.

CMPluginHandleSelection (page 1238)
An application-defined callback function that handles menu item selection in a contextual menu CFPlugin.

CMPluginPostMenuCleanup (page 1239)
An application-defined callback function that handles any post-selection cleanup in a contextual menu CFPlugin.

Obsolete Functions

GetItemCmd (page 1262)
Returns the value of a menu item's keyboard equivalent field.

GetMenu (page 1266)
Creates a menu from the specified menu and extended menu resources.

GetMenuExcludesMarkColumn (page 1272)
Returns whether a menu contains space for mark characters.

GetNewMBar (page 1290)
Reads in the definition of a menu bar from an 'MBAR' resource.

IsShowContextualMenuClick (page 1306)
Determines whether a particular event could invoke a contextual menu.

MenuEvent (page 1309)
Maps a keyboard combination from the event structure to the keyboard equivalent of a menu item in a menu in the current menu list.

SetItemCmd (page 1322)
Sets the value of the keyboard equivalent field of a menu item.

SetMenuExcludesMarkColumn (page 1330)
Sets whether a menu contains space for mark characters.

AppendMenu (page 1229) Deprecated in Mac OS X v10.5
Appends one or more items to a menu previously created.

AppendMenuItemText (page 1231) Deprecated in Mac OS X v10.5
Appends a menu item to a menu.

AppendResMenu (page 1232) Deprecated in Mac OS X v10.5
Searches all resource files open to your application for a given resource type and appends the names of any resources it finds to a specified menu.

DeleteMCEntries (page 1248) Deprecated in Mac OS X v10.5
Deletes a menu item entry, a menu title entry, the menu bar entry, or all menu item entries of a specific menu from your application's menu color information table.

DisposeMCInfo (page 1252) Deprecated in Mac OS X v10.5
Disposes of a menu color information table.

DisposeMenuDefUPP (page 1254) Deprecated in Mac OS X v10.5
Disposes of universal procedure pointer to a custom menu definition.

GetMCEntry (page 1265) Deprecated in Mac OS X v10.5
Gets information about an entry in an application's menu color information table.

GetMCInfo (page 1266) Deprecated in Mac OS X v10.5
Returns a handle to a copy of your application's menu color information table.

GetMenuItemText (page 1285) Deprecated in Mac OS X v10.5
Obtains the text of a menu item.

GetMenuTitle (page 1287) Deprecated in Mac OS X v10.5
Obtains the title of the menu

InitContextualMenus (page 1292) Deprecated in Mac OS X v10.5
Adds a program to the system registry of contextual menu clients.

InsertFontResMenu (page 1293) Deprecated in Mac OS X v10.5
Inserts menu items from a font resource.

InsertIntlResMenu (page 1293) Deprecated in Mac OS X v10.5
Inserts menu items from an internationalized resource.

InsertMenuItem (page 1295) Deprecated in Mac OS X v10.5
Inserts one or more items into a menu previously created.

InsertMenuItemText (page 1296) Deprecated in Mac OS X v10.5
Inserts a menu item into a menu.

InsertResMenu (page 1297) Deprecated in Mac OS X v10.5
Searches all resource files open to your application for a given resource type and inserts the names of any resources it finds in the specified menu.

InvokeMenuDefUPP (page 1301) Deprecated in Mac OS X v10.5
Calls your custom menu definition through a universal procedure pointer.

MenuKey (page 1311) Deprecated in Mac OS X v10.5
Maps a character key with the command key to determine the keyboard equivalent of a menu item in a menu in the current menu list.

NewMenu (page 1314) Deprecated in Mac OS X v10.5
Creates an empty menu with a specified title and menu ID.

NewMenuDefUPP (page 1315) Deprecated in Mac OS X v10.5
Creates a new univeral procedure pointer to your custom menu definition.

ProcessIsContextualMenuClient (page 1317) Deprecated in Mac OS X v10.5
Determines whether a given program is a contextual menu client.

SetMCEntries (page 1326) Deprecated in Mac OS X v10.5
Sets entries in an application's menu color information table.

SetMCInfo (page 1326) Deprecated in Mac OS X v10.5
Makes a copy of your application's menu color information table.

SetMenuItemText (page 1342) Deprecated in Mac OS X v10.5
Sets menu item text to a specified string.

SetMenuTitle (page 1345) Deprecated in Mac OS X v10.5
Sets the title of a menu.

Functions

AcquireRootMenu

Get the menu whose contents are displayed in the menubar.

```
MenuRef AcquireRootMenu (
   void
);
```

Return Value
The current root menu. See page for a description of the `MenuRef` data type. If no root menu currently exists, the Menu Manager creates one and returns its menu reference.

Discussion
This function increments the reference count of the root menu. The caller should call `ReleaseMenu` (page 1319) when done with the menu.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

AppendMenu

Appends one or more items to a menu previously created. (Deprecated in Mac OS X v10.5.)

Not recommended

```
void AppendMenu (
    MenuRef menu,
    ConstStr255Param data
);
```

Parameters

menu

The menu to which you wish to append the menu item or items.

data

A Pascal string that defines the characteristics of the new menu item or items. Note that in most cases you should store the text of a menu item in a resource, so that your menu items can be more easily localized. The `AppendMenu` function appends the menu items in the order in which they are listed in the data parameter.

Discussion
Note that unless you are supporting legacy code, you should use the `AppendMenuItemTextWithCFString` (page 1231) function instead.

The AppendMenu function appends any defined menu items to a previously-created menu. The menu items are added to the end of the menu. You specify the text of any menu items and their characteristics in the data parameter. You can embed metacharacters in the string to define various characteristics of a menu item.

Table 15-1 (page 1230) lists the metacharacters that you can specify in the data parameter:

Table 15-1 Metacharacters available to pass in AppendMenu

Metacharacter	Description
; or Return	Separates menu items.
^	When followed by an icon number, defines the icon for the item. If the keyboard equivalent field contains 0x1C, this number is interpreted as a text encoding.
!	When followed by a character, defines the mark for the item. If the keyboard equivalent field contains 0x1B or the equivalent constant hMenuCmd, this value is interpreted as the menu ID of a submenu of this menu item.
<	When followed by one or more of the characters B, I, U, O, and S, defines the character style of the item to Bold, Italic, Underline, Outline, or Shadow, respectively.
/	When followed by a character, defines the keyboard equivalent for the item. When followed by 0x1B or the equivalent constant hMenuCmd, specifies that this menu item has a submenu. To specify that the menu item has a text encoding, small icon, or reduced icon, use the SetItemCmd function to set the keyboard equivalent field to 0x1C, 0x1D, or 0x1E, respectively.
(	Defines the menu item as disabled.

You can specify any, all, or none of these metacharacters in the text string. The metacharacters that you specify aren't displayed in the menu item. (To use any of these metacharacters in the text of a menu item, first use AppendMenu, specifying at least one character as the item's text, and then use the function SetMenuItemText (page 1342) to set the item's text to the desired string.)

If you add menu items using the AppendMenu function, you should define the text and any marks or keyboard equivalents in resources for easier localization.

You can specify the first character that defines the text of a menu item as a hyphen to create a divider line. The string in the data parameter can be blank (containing one or more spaces), but it should not be an empty string.

If you do not define a specific characteristic of a menu item, the AppendMenu function assigns the default characteristic to the menu item. If you do not define any characteristic other than the text for a menu item, the AppendMenu function inserts the menu item so that it appears in the menu as an enabled item, without an icon or a mark, in the plain character style, and without a keyboard equivalent.

You can use AppendMenu to append items to a menu regardless of whether the menu is in the current menu list.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Related Sample Code
Simple DrawSprocket

Declared In
`Menus.h`

AppendMenuItemText

Appends a menu item to a menu. (Deprecated in Mac OS X v10.5.)

Not recommended

```
OSStatus AppendMenuItemText (
   MenuRef menu,
   ConstStr255Param inString
);
```

Parameters

menu

The menu to which the item is to be appended.

inString

A Pascal string containing the text of the menu item to append. You can pass a string containing any characters, and these characters will be presented verbatim in the menu item.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
Note that unless you are supporting legacy code, you should use the `AppendMenuItemTextWithCFString` (page 1231) function instead.

The `AppendMenuItemText` function appends the menu item containing the specified string to a menu, without evaluating the string for metacharacters, as the pre–Mac OS 8.5 Menu Manager function `AppendMenu` does. You may wish to use `AppendMenuItemText` if you need to present non-alphanumeric characters in a menu item.

The appended menu item appears at the end of the menu as an enabled item. If you wish to place the menu item elsewhere than at the end of the menu you should use the function `InsertMenuItemText` (page 1296).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Menus.h`

AppendMenuItemTextWithCFString

Appends a new menu item with text from a CFString.

```
OSStatus AppendMenuItemTextWithCFString (
   MenuRef inMenu,
   CFStringRef inString,
   MenuItemAttributes inAttributes,
   MenuCommand inCommandID,
   MenuItemIndex *outNewItem
);
```

Parameters

menu

The menu to which to append the new item.

inString

The text of the new item.

inAttributes

The attributes of the new item.

inCommandID

The command ID of the new item.

outNewItem

On exit, the index of the new item. May be `NULL` if the caller does not need this information.

Return Value

A result code. See "Menu Manager Result Codes" (page 1399).

Discussion

The Menu Manager will make its own copy of the CFString before returning from this function. Modifying the string after calling `AppendMenuItemTextWithCFString` will have no effect on the menu item's actual text. The caller may release the string after the call.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

BSDLLCTest

HID Explorer

Declared In

`Menus.h`

AppendResMenu

Searches all resource files open to your application for a given resource type and appends the names of any resources it finds to a specified menu. (Deprecated in Mac OS X v10.5.)

Not recommended

```
void AppendResMenu (
   MenuRef theMenu,
   ResType theType
);
```

Parameters

theMenu

The menu to which to append the names of any resources of a given type that `AppendResMenu` finds.

theType

A four-character code that identifies the resource type for which to search.

Discussion

Unless you must support legacy code, you should not use functions like `AppendResMenu` ,which assumes that menu items to append are stored in resources.

If you want to insert fonts into a menu, you should call `CreateStandardFontMenu` (page 1247) instead.

The `AppendResMenu` function searches all resource files open to your application for resources of the type defined by the parameter `theType`. It appends the names of any resources it finds of the given type to the end of the specified menu. `AppendResMenu` appends the names of found resources in alphabetical order; it does not alphabetize items already in the menu. The `AppendResMenu` function does not add resources with names that begin with a period (.) or a percent sign (%) to the menu.

`AppendResMenu` assigns default characteristics to each menu item. Each appended menu item appears in the menu as an enabled item, without an icon or a mark, in the plain character style, and without a keyboard equivalent.

Note that for applications using CarbonLib, you no longer need to call `AppendResMenu` add resources of type `'DRVR '` to your Apple menu; CarbonLib does this for you automatically.

If you specify that `AppendResMenu` append resources of type `'FONT '` or `'FOND '`, the Menu Manager performs special processing for any resources it finds that have font numbers greater than 0x4000. If the script system associated with the font name is installed in the system, `AppendResMenu` stores information in the `itemDefinitions` array (in the `itemIcon` and `itemCmd` fields for that item) in the menu structure. This allows the Menu Manager to display the font name in the correct script.

Special Considerations

The `AppendResMenu` function calls the Resource Manager function `SetResLoad` (specifying `true` in the `load` parameter) before returning. The `AppendResMenu` function reads the resource data of the resources it finds into memory. If your application does not want the Resource Manager to read resource data into memory when your application calls other functions that read resources, you need to call `SetResLoad` and specify `false` in the `load` parameter after `AppendResMenu` returns.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Related Sample Code

Simple DrawSprocket

Declared In

`Menus.h`

CalcMenuSize

Recalculates the horizontal and vertical dimensions of a menu.

```
void CalcMenuSize (
   MenuRef theMenu
);
```

Parameters

theMenu

The menu whose dimensions need recalculating.

Discussion

The `CalcMenuSize` function uses the menu definition function of the specified menu to calculate the dimensions of the menu. In most cases, your application does not need to use the `CalcMenuSize` function.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Menus.h`

CancelMenuTracking

Cancels menu tracking.

```
OSStatus CancelMenuTracking (
   MenuRef inRootMenu,
   Boolean inImmediate,
   UInt32 inDismissalReason
);
```

Parameters

inRootMenu

The root menu of the tracking session to dismiss. For menu bar tracking, pass the result from `AcquireRootMenu` (page 1229); for popup menu tracking, pass the menu that was passed to `PopUpMenuSelect` (page 1316).

inImmediate

Pass `true` if you want the open menus to disappear immediately, `false` if you want them to fade out.

inDismissalReason

Why the menu is being dismissed. This value is passed in the `kEventMenuEndTracking` event. You can pass the constants in "Menu Dismissal Constants" (page 1397). If you pass zero here, the `kEventMenuEndTracking` event contains `kMenuDismissedByCancelMenuTracking`.

Return Value

A result code. See "Menu Manager Result Codes" (page 1399).

Availability

Available in Mac OS X v10.3 and later.

Not available to 64-bit applications.

Declared In
Menus.h

ChangeMenuAttributes

Changes the attributes of a menu.

```
OSStatus ChangeMenuAttributes (
   MenuRef menu,
   MenuAttributes setTheseAttributes,
   MenuAttributes clearTheseAttributes
);
```

Parameters

menu
The menu whose attributes you want to change.

setTheseAttributes
The attributes to add to the menu. See "Menu Attribute Constants" (page 1375) for a list of possible values.

clearTheseAttributes
The attributes to remove from the menu. See "Menu Attribute Constants" (page 1375) for a list of possible values.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
Menus.h

ChangeMenuItemAttributes

Changes the attributes of a menu item.

```
OSStatus ChangeMenuItemAttributes (
   MenuRef menu,
   MenuItemIndex item,
   MenuItemAttributes setTheseAttributes,
   MenuItemAttributes clearTheseAttributes
);
```

Parameters

menu
The menu.

item
The index of the menu item.

setTheseAttributes
The attributes to add to the menu item. See "Menu Item Attribute Constants" (page 1377) for a list of possible values.

clearTheseAttributes
The attributes to remove from the menu item.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

ChangeMenuItemPropertyAttributes

Changes the attributes of a menu item property.

```
OSStatus ChangeMenuItemPropertyAttributes (
   MenuRef menu,
   MenuItemIndex item,
   OSType propertyCreator,
   OSType propertyTag,
   OptionBits attributesToSet,
   OptionBits attributesToClear
);
```

Parameters
menu
The menu containing the menu item whose properties you want ot change.
item
The index of the menu item.
propertyCreator
The creator code of the property.
propertyTag
The property tag.
attributesToSet
The attributes to add to the menu item property.
attributesToClear
The attributes to remove from the menu item property.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
Currently you can only specify the `kMenuPropertyPersistent` attribute (See "Menu Item Property Attribute Constant" (page 1395)), which currently has no effect on Mac OS X. Therefore, `SetMenuItemPropertyAttributes` is not useful at this time.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
Menus.h

CheckMenuItem

Adds or removes a check mark from a menu item.

```
void CheckMenuItem (
    MenuRef theMenu,
    short item,
    Boolean checked
);
```

Parameters

theMenu
The menu containing the menu item to check or uncheck.

item
The menu index of the item to check or uncheck.

checked
Pass `true` to add a check, `false` to remove it.

Discussion
You can also add or remove a check mark using the `SetItemMark` (page 1324) function.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
Menus.h

ClearMenuBar

Deletes all menus from the current menu list.

```
void ClearMenuBar (
   void
);
```

Discussion
The `ClearMenuBar` function deletes all menus from the current menu list. `ClearMenuBar` decrements the reference count of each menu in the menu list; if the reference count reaches zero, the memory associated with the menu is released. To explicitly release the memory occupied by a menu, use `DisposeMenu` (page 1253).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
Menus.h

CMPluginExamineContext

An application-defined callback function that examines the context in a contextual menu CFPlugin.

```
OSStatus CMPluginExamineContext (
   void *thisInstance,
   const AEDesc *inContext,
   AEDescList *outCommandPairs
);
```

Parameters

thisInstance

The instance of this plugin.

inContext

The Apple event descriptor describing the selection that invoked the contextual menu.

outCommandPairs

On return, `outCommandPairs` points to an array of Apple event descriptors, each of which contains information about a menu item to display in the contextual menu.

Return Value

A result code. See "Menu Manager Result Codes" (page 1399).

Discussion

Your contextual menu plugin must implement this function to determine what menu items to display given the user selection. The Apple event descriptor for each menu item contains the information to display in the item (such as text) and a command ID.

Availability

Available in Mac OS X v10.1 and later.

Declared In

`Menus.h`

CMPluginHandleSelection

An application-defined callback function that handles menu item selection in a contextual menu CFPlugin.

```
OSStatus CMPluginHandleSelection (
   void *thisInstance,
   AEDesc *inContext,
   SInt32 inCommandID
);
```

Parameters

thisInstance

The instance of this plugin.

inContext

The Apple event descriptor describing the selection that invoked the contextual menu.

inCommandID

The command ID associated with the user's menu item selection.

Return Value

A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
Your contextual menu plugin must implement this function to handle a user's selection of an item in the contextual menu. The command ID indicates which of the choices you gave in `CMPluginExamineContext` the user selected, so you can use that and the user selection stored in the `inContext` parameter to execute the conxtexual menu selection appropriately.

Availability
Available in Mac OS X v10.1 and later.

Declared In
`Menus.h`

CMPluginPostMenuCleanup

An application-defined callback function that handles any post-selection cleanup in a contextual menu CFPlugin.

```
void CMPluginPostMenuCleanup (
   void *thisInstance
);
```

Parameters

`thisInstance`
The instance of this plugin.

Discussion
Your contextual menu plugin must implement this function to handle any necessary cleanup required after displaying the contextual menu. If no cleanup is needed, you can simply return from this function.

Availability
Available in Mac OS X v10.1 and later.

Declared In
`Menus.h`

ContextualMenuSelect

Displays a contextual menu.

```
OSStatus ContextualMenuSelect (
   MenuRef inMenu,
   Point inGlobalLocation,
   Boolean inReserved,
   UInt32 inHelpType,
   ConstStr255Param inHelpItemString,
   const AEDesc *inSelection,
   UInt32 *outUserSelectionType,
   MenuID *outMenuID,
   MenuItemIndex *outMenuItem
);
```

Parameters

inMenu

The menu containing application commands to display. The caller creates this menu based on the current context, the mouse location, and the current selection (if it was the target of the mouse). If you pass `NULL`, only system commands are displayed.

inGlobalLocation

The location (in global coordinates) of the mouse near which the menu is to be displayed.

inReserved

Reserved for future use. Pass `false` for this parameter.

inHelpType

An identifier specifying the type of help provided by the application; see "Contextual Menu Help Type Constants" (page 1369).

inHelpItemString

The string containing the text to be displayed for the help menu item. This string is unused unless you also pass the constant `kCMOtherHelp` in the `inHelpType` parameter.

inSelection

A pointer to an object specifier for the current selection. This allows the system to examine the selection and add special system commands accordingly. Passing a value of `NULL` indicates that no selection should be examined, and most likely, no special system actions will be included.

outUserSelectionType

A pointer to an unsigned 32-bit value. On return, the value indicates what the user selected from the contextual menu; see "Contextual Menu Selection Type Constants" (page 1370) for a list of possible values.

outMenuID

A pointer to a signed 16-bit value. On return, if `outUserSelectionType` is set to `kCMMenuItemSelected`, the value is set to the menu ID of the chosen item.

outMenuItem

A pointer to an unsigned 16-bit value. On return, if `outUserSelectionType` is set to `kCMMenuItemSelected`, the value is set to the index of the menu item chosen.

Return Value

A result code. See "Menu Manager Result Codes" (page 1399). Returns `userCanceledErr` and sets `outUserSelectionType` to `kCMNothingSelected` if the user selects an item that requires no additional actions on your part.

Discussion
If your application uses the standard window handler, you may want to install handlers for the `kEventWindowContextualMenuSelect` or `kEventControlContextualMenuClick` events and call `ContextualMenuSelect` from within your handler. The standard window handler automatically detects contextual menu clicks and sends the `kEventWindowContextualMenuSelect` and `kEventControlContextualMenuClick` events.

If the `IsShowContextualMenuEvent` function returns `true` or you receive the appropriate contextual menu Carbon event, you should call the `ContextualMenuSelect` function after generating your own menu and preparing an Apple Event descriptor (`AEDesc`) that describes the item for which your application is displaying a contextual menu. This descriptor may contain an object specifier or raw data and will be passed to all contextual menu plug-ins.

The system will add other items before displaying the contextual menu, and it will remove those items before returning, leaving the menu in its original state.

After all the system commands are added, the contextual menu is displayed and tracked. If the user selects one of the system items, it is handled by the system and the call returns as though the user didn't select anything from the menu. If the user selects any other item (or no item at all), the Menu Manager passes back appropriate values in the parameters `outUserSelectionType`, `outMenuID`, and `outMenuItem`.

Your application should provide visual feedback indicating the item that was clicked upon. For example, a click on an icon should highlight the icon, while a click on editable text should not eliminate the current selection.

If the `outUserSelectionType` parameter contains `kCMMenuItemSelected`, you should look at the `outMenuID` and `outMenuItem` parameters to determine what menu item the user chose and handle it appropriately. If the `outUserSelectionType` parameter contains `kCMShowHelpSelected`, you should open the proper help sequence.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

CopyMenuItemData

Obtains multiple menu item attributes at once.

```
OSStatus CopyMenuItemData (
   MenuRef inMenu,
   MenuItemID inItem,
   Boolean inIsCommandID,
   MenuItemDataPtr ioData
);
```

Parameters

menu

The menu whose attributes you want to get. Note that if you pass `true` for the `inIsCommandID` parameter, you can pass `NULL` here, in which case the Menu Manager searches the root menu for the first menu that matches the specified command ID.

item
The menu item index or the command ID of the menu item.

isCommandID
A Boolean value indicating whether the value passed for the `inItem` parameter is a command ID or a menu item index. Pass `true` to indicate a command ID, `false` to indicate that it is a menu item index. If you pass `true`, the Menu Manager returns the data for the first menu item that matches the specified command ID.

outData
A pointer to a `MenuItemDataRec` structure. Before calling, you should set the `whichData` field to indicate what data you want to obtain. (Individual fields may also require initialization before calling.) On return, the structure contains the data you requested. For more details on the types of data you can obtain, see "Menu Item Data Flags" (page 1390).

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
You can use this function to obtain multiple menu item attributes simultaneously, which is often more efficient than making several different calls. For example, a menu definition function could use `CopyMenuItemData` to obtain all the individual attributes necessary for drawing a menu all at once.

This function returns copies of the data in the menu, so you should release any data in the `MenuItemDataRec` structure that was allocated dynamically (such as the CFString item text).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

CopyMenuItems

Copies menu items from one menu to another.

```
OSStatus CopyMenuItems (
   MenuRef inSourceMenu,
   MenuItemIndex inFirstItem,
   ItemCount inNumItems,
   MenuRef inDestMenu,
   MenuItemIndex inInsertAfter
);
```

Parameters

inSourceMenu
The menu from which to copy items.

inFirstItem
The first item to copy.

inNumItems
The number of items to copy.

inDestMenu
The menu to which to copy items.

inInsertAfter
The menu item in the destination menu after which to insert the copied items. Pass zero to insert items at the beginning of the menu. Note that you cannot specify an index value greater than the number of items in the destination menu.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

CopyMenuItemTextAsCFString

Returns a CFString containing the text of a menu item.

```
OSStatus CopyMenuItemTextAsCFString (
   MenuRef inMenu,
   MenuItemIndex inItem,
   CFStringRef *outString
);
```

Parameters
menu
The menu containing the item.
item
The item whose text you want to copy.
outString
On exit, a CFString containing the item's text. The caller must release this string when it is no longer needed.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

CopyMenuTitleAsCFString

Returns a CFString containing the title of a menu.

```
OSStatus CopyMenuTitleAsCFString (
   MenuRef inMenu,
   CFStringRef *outString
);
```

Parameters

inMenu

The menu whose title you want to obtain.

outString

On exit, a CFString containing the menu's title. This string must be released by the caller.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

CountMenuItems

Obtains the number of menu items in a menu

```
UInt16 CountMenuItems (
   MenuRef theMenu
);
```

Parameters

theMenu

The menu whose items you want to count.

Return Value
The number of menu items in the menu.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
HID Explorer

Declared In
`Menus.h`

CountMenuItemsWithCommandID

Counts the menu items with a specified command ID.

```
ItemCount CountMenuItemsWithCommandID (
   MenuRef inMenu,
   MenuCommand inCommandID
);
```

Parameters

menu

The menu in which to begin searching for items with the specified command ID. Pass `NULL` to begin searching with the root menu. The search will descend into all submenus of this menu.

commandID

The command ID for which to search.

Return Value
The number of menu items that match the specified command ID.

Version Notes
In CarbonLib 1.0.x, this function always returns zero or one; it stops after finding the first menu item with the specified command ID. In CarbonLib 1.1 and Mac OS X v10.0, it will count all menu items with the specified command ID.

In Mac OS X v10.0 and CarbonLib 1.0 through 1.4, this function searches only top-level menus (that is, those that are visible in the menu bar) and submenus of top-level menus. It does not search hierarchical menus that are in the menu bar but are not submenus of a top-level menu. For example, it does not search menus that are inserted for use in a popup menu. In Mac OS X v10.1 and later, and CarbonLib 1.5 and later, this function also searches hierarchical menus.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

CreateCustomMenu

Creates a new, untitled, empty menu using a custom menu definition function.

```
OSStatus CreateCustomMenu (
   const MenuDefSpec *inDefSpec,
   MenuID inMenuID,
   MenuAttributes inMenuAttributes,
   MenuRef *outMenuRef
);
```

Parameters

inDefSpec

A data structure that specifies a custom menu definition function.

inMenuID

The menu ID to use for the new menu.

inMenuAttributes

The menu attributes to use for the new menu. See "Menu Attribute Constants" (page 1375) for a list of possible values.

outMenuRef
On exit, contains the new menu.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
Similar to `CreateNewMenu` (page 1246), but also allows you to create a menu from a custom menu definition function This definition can be procedure pointer–based or HIView-based, which you specify in the `MenuDefSpec` structure. (Note that HIView–based menus are available only in Mac OS X v10.3 and later.)

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

CreateNewMenu

Creates a new, untitled, empty menu.

```
OSStatus CreateNewMenu (
   MenuID inMenuID,
   MenuAttributes inMenuAttributes,
   MenuRef *outMenuRef
);
```

Parameters
inMenuID
The menu ID to use for the new menu. Note that zero is a valid ID in Carbon.

inMenuAttributes
The menu attributes to use for the new menu. See "Menu Attribute Constants" (page 1375) for a list of possible values.

outMenuRef
On exit, contains the new menu.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
Unless you need to create new menus on-the-fly, you should not use functions like `CreateNewMenu` that create menus programmatically. Instead, you should define menus in Interface Builder, store them as nib files, and then call the Interface Builder Services function `CreateMenuFromNib` to create them.

`CreateNewMenu` is preferred over `NewMenu` because it allows you to specify the menu's attributes and it does not require you to specify a `Str255`-based menu title. To set the menu title, you should use `SetMenuTitleWithCFString` (page 1346).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
Menus.h

CreateStandardFontMenu

Creates a standard font menu.

```
OSStatus CreateStandardFontMenu (
   MenuRef menu,
   MenuItemIndex afterItem,
   MenuID firstHierMenuID,
   OptionBits options,
   ItemCount *outHierMenuCount
);
```

Parameters

menu
The menu to which you want to add the font items.

afterItem
The item number of the menu item after which the new menu items are to be added. If you want to insert the new items before the first menu items, specify 0. If you want to insert the items after the last item in the menu, specify a number greater than or equal to the last item in the menu. Otherwise, specify the item number for the menu item after which you want to insert new items.

firstHierMenuID
The first menu ID to use if any hierarchical menus are created. This ID is incremented for each additional hierarchical menu.

options
An option bits structure that specifies the behavior for the Font menu. Specify the `Hierarchical Font Menu Option Constant` (page 1375) if you want to construct a hierarchical font menu.

outHierMenuCount
On return, this parameter contains the number of hierarchical menus attached to the standard font menu. This value may be `NULL` if the hierarchical menus count is not useful. For example, if the only submenus in your application are those created by `CreateStandardFontMenu`, then you don't need to worry about the hierarchical menu count, as any existing submenu must be a font menu.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
You should use this function instead of the functions `AppendResMenu` (page 1232) or `InsertResMenu` (page 1297) to designate objects in the font database as menu items in the Font menu. The fonts objects will appear by name.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
Menus.h

DeleteMCEntries

Deletes a menu item entry, a menu title entry, the menu bar entry, or all menu item entries of a specific menu from your application's menu color information table. (Deprecated in Mac OS X v10.5.)

Not recommended

```
void DeleteMCEntries (
   MenuID menuID,
   short menuItem
);
```

Parameters

menuID

The menu ID that the `DeleteMCEntries` function should use to determine which entry to delete from the menu color information table. Specify 0 in the `menuID` parameter (and the `menuItem` parameter) to delete the menu bar entry. Specify the menu ID of a menu in the current menu list in the `menuID` parameter and 0 in the `menuItem` parameter to delete a specific menu title entry. Specify the menu ID of a menu in the current menu list in the `menuID` parameter and an item number in the `menuItem` parameter to delete a specific menu item entry.

menuItem

The menu item that the `DeleteMCEntries` function should use to determine which entry to delete from the menu color information table. If you specify 0 in this parameter, `DeleteMCEntries` deletes either the menu bar entry or menu title entry, depending on the value of the `menuID` parameter. If you specify the item number of a menu item in this parameter and the menu ID of a menu in the current menu list in the `menuID` parameter, `DeleteMCEntries` deletes a specific menu item entry. You can also delete all menu item entries for a specific menu from your application's menu color information table by specifying the constant `mctAllItems`.

Discussion

If the `DeleteMCEntries` function does not find the specified entry in your application's menu color information table, it does not delete the entry. Your application should not delete the last entry in your application's menu color information table.

If any of the deleted entries changes the menu bar color or a menu title color, your application should call `DrawMenuBar` (page 1255) to update the menu bar.

Carbon Porting Notes

`DeleteMCEntries` does nothing, because the Appearance Manager doesn't use color tables.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Menus.h`

DeleteMenu

Deletes an existing menu from the current menu list.

```
void DeleteMenu (
    MenuID menuID
);
```

Parameters

menuID

The menu ID of the menu to delete from the current menu list. If the menu list does not contain a menu with the specified menu ID, `DeleteMenu` does nothing.

Discussion

The `DeleteMenu` function deletes the menu identified by the specified menu ID from the current menu list. `DeleteMenu` decrements the reference count of the menu, and if the reference count reaches zero, the memory occupied by the menu is released. To explicitly release the memory occupied by the menu, use `DisposeMenu` (page 1253).

The `DeleteMenu` function first checks the submenu portion of the current menu list for a menu ID with the specified ID. If it finds such a menu, it deletes that menu and returns. If `DeleteMenu` doesn't find the menu in the submenu portion, it checks the regular portion of the current menu list.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Menus.h`

DeleteMenuItem

Deletes an item from a menu.

```
void DeleteMenuItem (
   MenuRef theMenu,
   MenuItemIndex item
);
```

Parameters

theMenu

The menu from which you want to delete the menu item.

item

The item number of the menu item to delete. If you specify 0 or a number greater than the last item in the menu, `DeleteMenuItem` does not delete any item from the menu.

Discussion

The `DeleteMenuItem` function deletes a specified menu item from a menu. You should not delete items from an existing menu unless the user expects the menu (such as a menu that lists open documents) to change.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Menus.h`

DeleteMenuItems

Deletes multiple menu items.

```
OSStatus DeleteMenuItems (
   MenuRef inMenu,
   MenuItemIndex inFirstItem,
   ItemCount inNumItems
);
```

Parameters

inMenu

The menu from which to delete items.

inFirstItem

The first item to delete.

inNumItems

The number of items to delete.

Return Value

A result code. See "Menu Manager Result Codes" (page 1399).

Discussion

You can call this function rather than calling `DeleteMenuItem` (page 1249) multiple times.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

HID Explorer

Declared In

`Menus.h`

DisableAllMenuItems

Disables all items in a menu.

```
void DisableAllMenuItems (
   MenuRef theMenu
);
```

Parameters

theMenu

The menu whose items you want to disable.

Discussion

This function is equivalent to older code that masked the `enableFlags` field of the `MenuInfo` structure (now opaque in Carbon) with `0x01`. It disables all items (including items past item 31) but does not affect the state of the menu title.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
Menus.h

DisableMenuCommand

Disables the menu item with a specified command ID.

```
void DisableMenuCommand (
   MenuRef inMenu,
   MenuCommand inCommandID
);
```

Parameters

theMenu

The menu in which to begin searching for the item. Pass NULL to begin searching with the root menu. The search will descend into all submenus of this menu.

commandID

The command ID of the menu item to be disabled. If more than one item has this command ID, only the first will be disabled.

Discussion
If you have access to the menu item index, in most cases you should use DisableMenuItem (page 1251) instead, as that function is faster and requires no searching.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
CarbonSketch

Declared In
Menus.h

DisableMenuItem

Disables a menu item or a menu.

```
void DisableMenuItem (
   MenuRef theMenu,
   MenuItemIndex item
);
```

Parameters

theMenu

The menu containing the item to be disabled.

item

The index of the menu item that you wish to disable. Pass 0 to specify the menu title (disabling the entire menu).

Discussion
The DisableMenuItem function disables a menu item (and any associated icon) so that the user cannot choose the item from the menu.

Note that `EnableMenuItem` calls the `InvalMenuBar` (page 1301) function to update the menu bar the next time through the event loop.

See also the `EnableMenuItem` (page 1257) and `IsMenuItemEnabled` (page 1303) functions.

Carbon Porting Notes

Note that the implementation of Carbon on Mac OS 8.1 only supports disabling menu items less than or equal to 31.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

QTCarbonShell

Declared In

`Menus.h`

DisableMenuItemIcon

Disables the icon associated with a menu item.

```
void DisableMenuItemIcon (
   MenuRef theMenu,
   MenuItemIndex item
);
```

Parameters

theMenu

The menu containing the icon to be disabled.

item

The index of the menu item containing the icon.

Discussion

Your application can use the `DisableMenuItemIcon` function to dim individual menu item icons. The menu item that contains the icon is unaffected by calling `DisableMenuItemIcon`. That is, if `DisableMenuItemIcon` disables an enabled menu item's icon, the menu item itself will remain enabled. Calling `DisableMenuItemIcon` on the icon of a menu item that is currently disabled will cause the icon to be disabled once the menu item is re-enabled.

See also the functions `EnableMenuItemIcon` (page 1258) and `IsMenuItemIconEnabled` (page 1304).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Menus.h`

DisposeMCInfo

Disposes of a menu color information table. (Deprecated in Mac OS X v10.5.)

Not recommended

```
void DisposeMCInfo (
   MCTableHandle menuCTbl
);
```

Parameters

menuCTbl

The handle to the menu color information table you want to remove.

Discussion

The `DisposeMCInfo` function disposes of the menu color information table referred to by the `menuCTbl` parameter.

Carbon Porting Notes

`DisposeMCInfo` does nothing, because Appearance Manager doesn't use color tables.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Menus.h`

DisposeMenu

Decrements the retain count of a menu.

```
void DisposeMenu (
   MenuRef theMenu
);
```

Parameters

theMenu

The menu whose retain count to decrement. If the retain count falls to zero, the menu is destroyed.

Discussion

The reference that you pass in the `theMenu` parameter is not valid after `DisposeMenu` returns. This function is identical to `ReleaseMenu` (page 1319).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

HID Explorer

Declared In

`Menus.h`

DisposeMenuBar

Releases a menubar handle.

```
OSStatus DisposeMenuBar (
   MenuBarHandle inMbar
);
```

Parameters

mbar

The menubar handle to release.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
Carbon applications should call this function when releasing a handle returned from `GetNewMBar` (page 1290), `GetMenuBar` (page 1268), or `DuplicateMenuBar` (page 1256). Doing so ensures that the reference counts of the menus in the menubar handle can be decremented when the handle is released.

Do not call the Memory Manager function `DisposeHandle` to release such a handle.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

DisposeMenuDefUPP

Disposes of universal procedure pointer to a custom menu definition. (Deprecated in Mac OS X v10.5.)

Not recommended

```
void DisposeMenuDefUPP (
   MenuDefUPP userUPP
);
```

Parameters

userUPP

Carbon Porting Notes

Apple discourages you from writing and using your own menu definition functions and encourages you to use the system-supplied menu definition function instead. New features that have previously been missing are now available in the system-supplied menu definition function. Since Appearance Manager 1.0 (in Mac OS 8.0), for example, the system-supplied menu definition function has supported extended menu item command key modifiers and glyphs. And in Carbon, the system-supplied menu definition function supports dynamic items, which allow the contents of a menu item to be redrawn while the menu is displayed in response to the user pressing a modifier key on the keyboard.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Menus.h`

DrawMenuBar

Draws the menu bar based on the current menu list.

```
void DrawMenuBar ();
```

Discussion
The `DrawMenuBar` function draws (or redraws) the menu bar according to the current menu list. Note that most Menu Manager calls that affect the menu bar call `InvalMenuBar` (page 1301) so that the menu bar is redrawn the next time through the event loop; however, you can call `DrawMenuBar` if you want the changes to appear immediately.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
Simple DrawSprocket

Declared In
`Menus.h`

DuplicateMenu

Creates a new menu that is a copy of another menu.

```
OSStatus DuplicateMenu (
   MenuRef inSourceMenu,
   MenuRef *outMenu
);
```

Parameters

inSourceMenu
The menu to duplicate.

outMenu
On exit, a copy of the source menu.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
Unlike `RetainMenu` (page 1321), `DuplicateMenu` creates an entirely new menu that is an exact copy of the original menu. The menu definition for the new menu will receive an initialization message/event after the menu has been fully created.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

DuplicateMenuBar

Duplicates a menubar handle.

```
OSStatus DuplicateMenuBar (
   MenuBarHandle inMbar,
   MenuBarHandle *outMbar
);
```

Parameters

inMbar

The menubar handle to duplicate.

outMBar

On exit, contains the new menubar handle.

Return Value

A result code. See "Menu Manager Result Codes" (page 1399).

Discussion

Carbon applications should use this function when duplicating a handle returned from `GetMenuBar` or `GetNewMBar`. Doing so ensures that the reference counts of the menus in the menubar handle can be incremented when the handle is duplicated.

Do not use Memory Manager APIs (`HandToHand`, `NewHandle`, and so on) to duplicate such a handle.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Menus.h`

EnableAllMenuItems

Enables all items in a menu.

```
void EnableAllMenuItems (
   MenuRef theMenu
);
```

Parameters

theMenu

The menu whose items to enable.

Discussion

This function is equivalent to older code that OR'd the `enableFlags` field of the `MenuInfo` structure (now opaque in Carbon) with 0xFFFFFFFE. It enables all items (including items past item 31) but does not affect the state of the menu title.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Menus.h`

EnableMenuCommand

Enables the menu item with a specified command ID.

```
void EnableMenuCommand (
   MenuRef inMenu,
   MenuCommand inCommandID
);
```

Parameters

inMenu

The menu in which to begin searching for the item. Pass `NULL` to begin searching with the root menu. The search will descend into all submenus of this menu.

inCommandID

The command ID of the menu item to be enabled. If more than one item has this command ID, only the first will be enabled.

Discussion

If you have access to the menu item index, in most cases you should use `EnableMenuItem` (page 1257) instead, as that function is faster and requires no searching. For example, when receiving a `kEventCommandUpdateStatus` event, the `HICommand` structure contains both the menu item's command ID and index. If you wanted to enable the menu item, you should call `EnableMenuItem` (page 1257).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

CarbonSketch

QTCarbonShell

Declared In

`Menus.h`

EnableMenuItem

Enables a menu item or a menu.

```
void EnableMenuItem (
    MenuRef theMenu,
    MenuItemIndex item
);
```

Parameters

theMenu

The menu containing the item to be enabled.

item

The item number of the menu item that you wish to enable. If you pass 0, `EnableMenuItem` enables the menu title and all items in the menu that were not previously individually disabled.

Discussion

The `EnableMenuItem` function enables a menu item so that the user can choose the item from the menu. If the menu item has an associated icon, that icon is also enabled, unless the icon was previously individually disabled with the function `DisableMenuItemIcon` (page 1252).

Note that `EnableMenuItem` calls the `InvalMenuBar` (page 1301) function to update the menu bar the next time through the event loop.

See also the `DisableMenuItem` (page 1251) and `IsMenuItemEnabled` (page 1303) functions.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
QTCarbonShell

Declared In
`Menus.h`

EnableMenuItemIcon

Enables the icon associated with a menu item.

```
void EnableMenuItemIcon (
   MenuRef theMenu,
   MenuItemIndex item
);
```

Parameters

theMenu
The menu containing the icon to be enabled.

item
The item number of the menu item containing the icon.

Discussion
Your application can use the `EnableMenuItemIcon` function to enable individual menu item icons that have been previously disabled by a call to the function `DisableMenuItemIcon` (page 1252). The menu item that contains the icon is unaffected by calling `EnableMenuItemIcon`. Note that enabling the icon of a currently disabled menu item has no visual effect; however, once the menu item is enabled, the icon is also enabled.

See also the `DisableMenuItemIcon` (page 1252) and `IsMenuItemIconEnabled` (page 1304) functions.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

EraseMenuBackground

Erases the menu background to prepare for additional drawing. (Deprecated in Mac OS X v10.5.)

```
OSStatus EraseMenuBackground (
   MenuRef inMenu,
   const Rect *inEraseRect,
   CGContextRef inContext
);
```

Parameters

inMenu

The menu whose background you want to erase.

inEraseRect

The bounds of the area to erase, in the local coordinates of the current port.

inContext

The Core Graphics context to erase. If set to `NULL`, this function creates a new context based on the current port.

Return Value

A result code. See "Menu Manager Result Codes" (page 1399).

Discussion

Typically you use this function only if you are implementing message-based custom menu definition functions. HIView-based custom menus and normal application code do not need to call `EraseMenuBackground`.

Before calling the Appearance Manager function `DrawThemeMenuBackground`, you must erase the current menu background. Themes such as Aqua draw the menu background using an alpha channel, so if the old background is not erased, portions of the old image will show through the menu background.

Availability

Available in Mac OS X v10.1 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Menus.h`

FlashMenuBar

Highlights a menu title or the entire menu bar.

```
void FlashMenuBar (
   MenuID menuID
);
```

Parameters

menuID

The menu ID of the menu whose title you want to highlight. Pass zero in this parameter to signal a visual alert (that is, flash the entire screen) If you pass a menu ID that is not in the menubar, `FlashMenuBar` returns immediately without flashing the bar or any menu title.

Discussion

Call this function twice if you want the menu or menu bar to blink.

Only one menu title can be highlighted at a time. If no menus are currently highlighted, calling `FlashMenuBar` with a specific menu ID highlights the title of that menu. If you call `FlashMenuBar` again specifying another menu ID that is different from that of the previously highlighted menu title, `FlashMenuBar` restores the previously highlighted menu to normal and then highlights the title of the specified menu.

If you pass zero for the menu ID, this function flashes the entire screen, as if an alert had occurred while the user had specified the "Flash the screen when an alert sound occurs" checkbox in the Universal Access preference pane.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

GetFontFamilyFromMenuSelection

Gets the font family reference and style from the menu identifier and menu item number returned by the function `MenuSelect`.

```
OSStatus GetFontFamilyFromMenuSelection (
   MenuRef menu,
   MenuItemIndex item,
   FMFontFamily *outFontFamily,
   FMFontStyle *outStyle
);
```

Parameters

menu
A menu handle.

item
A menu item index.

outFontFamily
A pointer to the font family reference associated with the menu item.

outStyle
A pointer to the font style associated with the menu item.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
You use this function to obtain information from a font menu created using `CreateStandardFontMenu` (page 1247), `AppendResMenu` (page 1232), `InsertFontResMenu` (page 1293), `InsertIntlResMenu` (page 1293), or `InsertResMenu` (page 1297).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

GetIndMenuItemWithCommandID

Finds a menu item with a specified command ID.

```
OSStatus GetIndMenuItemWithCommandID (
   MenuRef inMenu,
   MenuCommand inCommandID,
   UInt32 inItemIndex,
   MenuRef *outMenu,
   MenuItemIndex *outIndex
);
```

Parameters

inMenu

The menu in which to begin searching for items with the specified command ID. Pass `NULL` to begin searching with the root menu. The search will descend into all submenus of this menu.

inCommandID

The command ID for which to search.

inItemIndex

The 1-based index of the menu item to retrieve. In CarbonLib 1.0.x, this parameter must be 1. In CarbonLib 1.1 and Mac OS X 1.0, this parameter may vary from 1 to the number of menu items with the specified command ID.

outMenu

On exit, the menu containing the menu item with the specified command ID.

outIndex

On exit, the item index of the menu item with the specified command ID.

Return Value

A result code. See "Menu Manager Result Codes" (page 1399).

Discussion

This API searches the specified menu and its submenus for the n'th menu item with the specified command ID. You often want to use this function in conjunction with `CountMenuItemsWithCommandID` (page 1244).

Version Notes

In CarbonLib 1.1 and earlier, only the first menu item will be returned. In CarbonLib 1.2 and Mac OS X v10.0 and later, this API will iterate over all menu items with the specified command ID and return the `itemIndex`'th one.

In Mac OS X v10.0 and CarbonLib 1.0 through 1.4, this function searches only top-level menus (that is, those that are visible in the menu bar) and submenus of top-level menus. It does not search hierarchical menus that are in the menu bar but are not submenus of a top-level menu. For example, it does not search menus that are inserted for use in a popup menu. In Mac OS X v10.1 and later, and CarbonLib 1.5 and later, this function also searches hierarchical menus.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

CarbonSketch

Declared In

`Menus.h`

GetItemCmd

Returns the value of a menu item's keyboard equivalent field.

Not recommended

```
void GetItemCmd (
   MenuRef theMenu,
   MenuItemIndex item,
   CharParameter *cmdChar
);
```

Parameters

theMenu

The menu containing the menu item whose keyboard equivalent you want to obtain.

item

An integer representing the item number of the menu item whose keyboard equivalent you want to determine.

cmdChar

On output, an integer representing the item's keyboard equivalent field. The Menu Manager uses this value to map keyboard equivalents to menu commands or to indicate special characteristics of the menu item.

If the value referenced through the `cmdChar` parameter contains 0x1B, the menu item has a submenu; a value of 0x1C indicates that the item has a text encoding; a value of 0x1D indicates that the Menu Manager reduces the item's `'ICON'` resource; and a value of 0x1E indicates that the item has an `'SICN'` resource.

Discussion

You should call `GetMenuItemCommandKey` (page 1276) , `GetMenuItemHierarchicalID` (page 1277) , and `GetMenuItemTextEncoding` (page 1285) instead of `GetItemCmd` to obtain a menu item's keyboard equivalent and text encoding and to determine that a menu item has a submenu.

The `GetItemCmd` function returns the value in the keyboard equivalent field of the specified menu item in the value pointed to by the `cmdChar` parameter (or 0 if the item doesn't have a keyboard equivalent, submenu, text encoding, reduced icon, or small icon).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Menus.h`

GetItemIcon

Returns a menu item's icon or text encoding. (Deprecated in Mac OS X v10.5.)

```
void GetItemIcon (
   MenuRef theMenu,
   MenuItemIndex item,
   short *iconIndex
);
```

Parameters

theMenu

The menu containing the item.

item

The menu index of the item.

iconIndex

On output, an integer representing the menu item's icon or text encoding. For menu items that do not specify 0x1C in the keyboard equivalent field, a value from 1 through 255 if the menu item has an icon associated with it and is 0 otherwise representing the item's icon number.

Discussion

In most cases, you should use `GetMenuItemTextEncoding` (page 1285) rather than `GetItemIcon` to get the menu item's text encoding.

The `GetItemIcon` function returns the icon number or text encoding of the specified menu item through the `iconIndex` parameter (or 0 if the item doesn't have an icon or a text encoding). If the menu item's keyboard equivalent field contains 0x1C, the returned number represents the text encoding of the menu item. Otherwise, the returned number represents the item's icon number.

In the `iconIndex` parameter, you can add 256 to the icon number to generate the resource ID of the `'cicn'`, `'ICON '`, or `'SICN '` resource that describes the icon of the menu item. For example, if the `GetItemIcon` function returns 5 as the icon number, then the icon of the menu item is described by an icon resource with resource ID 261.

For menu items that contain 0x1C in the keyboard equivalent field, the `GetItemIcon` function returns the text encoding of the menu item. The Menu Manager displays the menu item using this text encoding if the corresponding script system is installed.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Menus.h`

GetItemMark

Returns a menu item's mark.

```
void GetItemMark (
   MenuRef theMenu,
   MenuItemIndex item,
   CharParameter *markChar
);
```

Parameters

theMenu

The menu containing the item.

item

The menu index of the item.

markChar

On output, an integer representing the mark of the menu item or its submenu (item has a submenu). See the Font Manager for a list of character marking constants that this function can obtain. This parameter is set to 0 if the menu item has neither mark nor submenu.

Discussion

You should call `GetMenuItemHierarchicalID` (page 1277) instead of `GetItemMark` to obtain the menu ID of a menu item's submenu. However, you can still use `GetItemMark` to obtain the mark of a menu item.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

QTCarbonShell

Declared In

`Menus.h`

GetItemStyle

Returns a menu item's text style.

```
void GetItemStyle (
   MenuRef theMenu,
   MenuItemIndex item,
   Style *chStyle
);
```

Parameters

theMenu

The menu containing the item.

item

The menu index of the item.

chStyle

On output, an integer representing the menu item's text style. The functions returns one of the following constants: `normal`, `bold`, `italic`, `underline`, `outline`, `shadow`, `condense`, and `extend`.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
`Menus.h`

GetMBarHeight

Determines the current height of the menu bar.

```
short GetMBarHeight (
   void
);
```

Return Value
The current height, in pixels, of the menu bar.

Discussion
The `GetMBarHeight` function determines the menu bar height based on factors such as the current script system and theme.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

GetMCEntry

Gets information about an entry in an application's menu color information table. (Deprecated in Mac OS X v10.5.)

Not recommended

```
MCEntryPtr GetMCEntry (
   MenuID menuID,
   short menuItem
);
```

Parameters

`menuID`

The menu ID that the `GetMCEntry` function should use to return information about the menu color information table. Specify 0 in the `menuID` parameter (and the `menuItem` parameter) to get the menu bar entry. Specify the menu ID of a menu in the current menu list in the `menuID` parameter and 0 in the `menuItem` parameter to get a specific menu title entry. Specify the menu ID of a menu in the current menu list in the `menuID` parameter and an item number in the `menuItem` parameter to get a specific menu item entry.

`menuItem`

The menu item that the `GetMCEntry` function should use to return information about the menu color information table. If you specify 0 in this parameter, `GetMCEntry` returns either the menu bar entry or the menu title entry, depending on the value of the `menuID` parameter. If you specify the item number of a menu item in this parameter and the menu ID of a menu in the current menu list in the `menuID` parameter, `GetMCEntry` returns a specific menu item entry.

Return Value
A menu bar entry, a menu title entry, or a menu item entry according to the values specified in the `menuID` and `menuItem` parameters. If the `GetMCEntry` function finds the specified entry in your application's menu color information table, it returns a pointer to a structure of data type `MCEntry`. If the specified entry is not found, `GetMCEntry` returns `null`. The menu color information table is relocatable, so the pointer returned by the `GetMCEntry` function may not be valid across functions that may move or purge memory. Your application should make a copy of the menu color entry structure if necessary. See page for a description of the `MCEntry` data structure.

Carbon Porting Notes

`GetMCEntry` does nothing, because Appearance Manager doesn't use color tables.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Menus.h`

GetMCInfo

Returns a handle to a copy of your application's menu color information table. (Deprecated in Mac OS X v10.5.)

Not recommended

```
MCTableHandle GetMCInfo (
   void
);
```

Return Value
A handle to a copy of your application's menu color information table. If the copy fails, `GetMCInfo` returns `null`.

Carbon Porting Notes

The Menu Manager ignores color tables in Mac OS X, so there is no reason to call `GetMCInfo`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Menus.h`

GetMenu

Creates a menu from the specified menu and extended menu resources.

Not recommended

```
MenuRef GetMenu (
    short resourceID
);
```

Parameters

resourceID

The resource ID of the menu and extended menu that defines the characteristics of the menu. You typically use the same number for a menu's resource ID as the number that you specify for the menu ID in the menu resource.

Return Value

The new menu. You can use the returned menu handle to refer to this menu in most Menu Manager functions. If `GetMenu` is unable to read the menu or menu definition function from the resource file, `GetMenu` returns `null`. See the description of the `MenuRef` data type.

Discussion

Unless you must support legacy code, you should not use functions like `GetMenu` that rely on menus stored as resources. Instead, you should define menus in Interface Builder, store them as nib files, and then call the Interface Builder Services function `CreateMenuFromNib` to create them.

`GetMenu` reads the menu definition function into memory (if not already present) and stores a handle to the menu definition function in the menu structure. `GetMenu` does not insert the newly created menu into the current menu list.

You typically use the `GetMenu` function only when you create submenus; you can create all your pull-down menus at once using the function `GetNewMBar`, and you can create pop-up menus using the standard pop-up menu button control definition function.

After reading the `'MENU'` resource, `GetMenu` searches for an extended menu resource and an `'mctb'` resource with the same resource ID as the `'MENU'` resource. If the specified `'mctb'` resource exists, `GetMenu` uses `SetMCEntries` to add the entries defined by the resource to the application's menu color information table. If the `'mctb'` resource does not exist, `GetMenu` uses the default colors specified in the menu bar entry of the application's menu color information. If neither a menu bar entry nor a `'mctb'` resource exists, `GetMenu` uses the standard colors for the menu.

Storing the definitions of your menus in resources (especially menu titles and menu items) makes your application easier to localize.

After creating a menu with `GetMenu` or `NewMenu` (page 1314) , use `InsertMenuItem` (page 1295) , `AppendMenu` (page 1229) , or `InsertResMenu` (page 1297) to add menu items to the menu. To add a menu created by `GetMenu` to the current menu list, use `InsertMenu` (page 1294). To update the menu bar with any new menu titles, use `DrawMenuBar` (page 1255).

Menus in a resource must not be purgeable nor should the resource lock bit be set. Do not define a "circular" hierarchical menu—that is, a hierarchical menu in which a submenu has a submenu whose submenu is a hierarchical menu higher in the chain.

Special Considerations

To release the memory associated with a menu that you created using `GetMenu`, first call `DeleteMenu` to remove the menu from the current menu list and to remove any entries for this menu in your application's menu color information table then call `DisposeMenu` to dispose of the menu structure.

Carbon Porting Notes

In Carbon, the `GetMenu` function always returns a newly created `MenuRef`. Prior to Carbon, `GetMenu` would first check if the menu was already in memory. If so, `GetMenu` would return the in-memory copy. This behavior is no longer supported.

Carbon does not support custom menu definitions stored in 'MDEF' resources. If you want to specify a custom menu definition for `GetMenu`, you must compile your definition function directly in your application and then register the function by calling `RegisterMenuDefinition` (page 1318). When `GetMenu` gets a `resourceID` value that doesn't recognize, it checks a special mapping table to find the pointer that's registered for the `resourceID` parameter. It then calls that function to implement your menu.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

GetMenuAttributes

Gets the attributes of a menu.

```
OSStatus GetMenuAttributes (
   MenuRef menu,
   MenuAttributes *outAttributes
);
```

Parameters

menu
The menu.

outAttributes
On exit, contains the attributes of the menu. See "Menu Attribute Constants" (page 1375) for a list of possible values.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

GetMenuBar

Gets a handle to a copy of the current menu list.

```
MenuBarHandle GetMenuBar (
   void
);
```

Return Value
A handle to a copy of the current menu list. See the description of the `MenuBarHandle` data type.

Discussion
The `GetMenuBar` function creates a copy of the current menu list and returns a handle to the copy. You can save the returned menu list and then add menus to or remove menus from the current menu list using `InsertMenu` (page 1294) , `DeleteMenu` (page 1248) , or `ClearMenuBar` (page 1237). You can later restore the saved menu list using `SetMenuBar` (page 1327).

To release the memory occupied by a saved menu list, use the `DisposeMenuBar` (page 1253) function.

`GetMenuBar` doesn't copy the menu structures, just the menu list (which contains handles to the menu structures). Do not dispose of any menus in a saved menu list if you wish to restore the menu list later.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

GetMenuCommandMark

Locates the menu item with a specified command ID and returns its mark character.

```
OSStatus GetMenuCommandMark (
   MenuRef inMenu,
   MenuCommand inCommandID,
   UniChar *outMark
);
```

Parameters

theMenu
The menu in which to begin searching for the item. Pass `NULL` to begin searching with the root menu. The search will descend into all submenus of this menu.

commandID
The command ID of the menu item to be examined. If more than one item has this command ID, only the first will be examined.

outMark
On exit, the menu item's mark character.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
If you have access to the menu item index, in most cases you should use `GetItemMark` (page 1263) instead, as that function is faster and requires no searching.

Availability
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
`Menus.h`

GetMenuCommandProperty

Retrieves property data for a menu item with the specified command ID.

```
OSStatus GetMenuCommandProperty (
   MenuRef inMenu,
   MenuCommand inCommandID,
   OSType inPropertyCreator,
   OSType inPropertyTag,
   ByteCount inBufferSize,
   ByteCount *outActualSize,
   void *inPropertyBuffer
);
```

Parameters

inMenu
The menu in which to search for the command ID. Pass `NULL` to begin searching with the root menu. The search will descend into all submenus of this menu.

inCommandID
The command ID of the menu item containing the property. Note that if more than one item has the same command ID, only the first match is used.

inPropertyCreator
The four-character creator code of the application.

inPropertyTag
The four-character tag of the property to obtain.

inBufferSize
The size of the buffer to hold the retrieved data, in bytes.

outActualSize
The actual size of the property data. If you do not need this information, pass `NULL`.

inPropertyBuffer
A pointer to the buffer in which to place the data. On return, the buffer contains the property data.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
If you have access to the menu item index, in most cases you should use `GetMenuItemProperty` (page 1281) instead, as that function is faster and requires no searching.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

GetMenuCommandPropertySize

Retrieves the size of the property data for a menu item with the specified command ID.

```
OSStatus GetMenuCommandPropertySize (
   MenuRef inMenu,
   MenuCommand inCommandID,
   OSType inPropertyCreator,
   OSType inPropertyTag,
   ByteCount *outSize
);
```

Parameters

menu

The menu in which to search for the command ID. Pass `NULL` to begin searching with the root menu. The search will descend into all submenus of this menu.

commandID

The command ID of the menu item containing the property. If more than one menu item has the same command ID, only the first match is used.

propertyCreator

The four-character creator code of the application.

propertyTag

The four-character tag of the property data whose size you want to obtain.

size

On return, contains the size of the property data.

Return Value

A result code. See "Menu Manager Result Codes" (page 1399).

Discussion

If you have access to the menu item index, in most cases you should use `GetMenuItemPropertySize` (page 1283) instead, as that function is faster and requires no searching.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Menus.h`

GetMenuDefinition

Obtains the menu definition structure for a menu.

```
OSStatus GetMenuDefinition (
   MenuRef menu,
   MenuDefSpecPtr outDefSpec
);
```

Parameters

menu

The menu whose menu definition structure you want to obtain.

outDefSpec
On return, a pointer to the menu's `MenuDefSpec` structure.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
Note that you cannot use this function to obtain the standard system menu definition. If you call `GetMenuDefinition` on a menu that uses the standard system MDEF or menu content HIView, the function returns `menuUsesSystemDefErr`.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

GetMenuExcludesMarkColumn

Returns whether a menu contains space for mark characters.

Not recommended

```
Boolean GetMenuExcludesMarkColumn (
   MenuRef menu
);
```

Parameters

menu
The menu whose width is to be examined.

Return Value
Returns `true` if the menu currently contains no space for mark characters; `false` if the menu is currently drawn in its full width, with space for mark characters.

Discussion
See also the `SetMenuExcludesMarkColumn` (page 1330) function.

Carbon Porting Notes

You should instead inspect the `kMenuExcludesMarkColumn` menu attribute using the `GetMenuAttributes` function.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

GetMenuFont

Obtains the font used in a menu.

```
OSStatus GetMenuFont (
   MenuRef menu,
   SInt16 *outFontID,
   UInt16 *outFontSize
);
```

Parameters

menu

The menu whose font is to be obtained.

outFontID

On input, a pointer to a signed 16-bit integer. On return, this value identifies the font family ID for the menu font. Note that this is the menu item font, not the menu title font.

outFontSize

On input, a pointer to an unsigned 16-bit integer. On return, this value identifies the size of the font, in points.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
Your application may use the `GetMenuFont` function to retrieve the font used for an individual menu, such as a pop-up menu.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

GetMenuHandle

Obtains a menu reference corresponding to a menu ID.

```
MenuRef GetMenuHandle (
   MenuID menuID
);
```

Parameters

menuID

The menu ID of the menu whose reference you want to obtain. (Note that this is not the resource ID, although you often assign the menu ID so that it matches the resource ID.) You assign a menu ID in a nib file or in the 'MENU' resource of a menu. If you do not define your menus in nib files or in 'MENU' resources, you can assign a menu ID using `NewMenu` (page 1314) or `SetMenuID` (page 1333).

Return Value
The menu corresponding to the specified ID. If the specified menu is not in the current menu list, `GetMenuHandle` returns `NULL`. See the description of the `MenuRef` data type.

Discussion
You can also call this function as `GetMenuRef (menuID);`.

Use the `GetMenuHandle` function to obtain the menu reference for any of your application's pull-down menus or submenus in the current menu list, other than the Help menu. You can also use the Help Manager function `HMGetHelpMenuHandle` to get a handle for your application's Help menu.

Special Considerations

To get a menu reference for a pop-up menu that you create using the pop-up control definition function, call the Control Manager functions `GetControlData` and `GetControlDataSize`, passing the tag constant `kControlPopupButtonMenuRefTag` in the `tagName` parameter to specify the menu reference.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
Simple DrawSprocket

Declared In
`Menus.h`

GetMenuHeight

Obtains the height of a menu, in pixels.

```
SInt16 GetMenuHeight (
   MenuRef menu
);
```

Parameters

menu
The menu whose height you want to obtain.

Return Value
The height of the menu, in pixels.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

GetMenuID

Obtains the ID of a menu.

```
MenuID GetMenuID (
   MenuRef menu
);
```

Parameters

menu

Return Value
The menu ID of the menu. See the description of the `MenuID` data type.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

GetMenuItemAttributes

Gets the attributes of a menu item.

```
OSStatus GetMenuItemAttributes (
   MenuRef menu,
   MenuItemIndex item,
   MenuItemAttributes *outAttributes
);
```

Parameters

menu
The menu.

item
The index of the menu item.

outAttributes
On exit, contains the attributes of the menu item. See "Menu Item Attribute Constants" (page 1377) for a list of possible values.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

GetMenuItemCommandID

Obtains a menu item's command ID.

```
OSErr GetMenuItemCommandID (
   MenuRef inMenu,
   MenuItemIndex inItem,
   MenuCommand *outCommandID
);
```

Parameters

inMenu
The menu that contains the menu item whose command ID you want to obtain.

inItem
The menu index of the item.

outCommandID

Pass a pointer to an unsigned 32-bit integer value. On return, the value is set to the item's command ID.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
After a successful call to `MenuSelect`, `MenuEvent` (page 1309) , or `MenuKey` (page 1311), call the `GetMenuItemCommandID` function to get a menu item's command ID. You can use a menu item's command ID as a position-independent method of signalling a specific action in an application.

See also the function `SetMenuItemCommandID` (page 1333).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

GetMenuItemCommandKey

Gets the keyboard equivalent of a menu item.

```
OSStatus GetMenuItemCommandKey (
   MenuRef inMenu,
   MenuItemIndex inItem,
   Boolean inGetVirtualKey,
   UInt16 *outKey
);
```

Parameters

inMenu

The menu containing the item.

inItem

The item whose keyboard equivalent you want to retrieve.

inGetVirtualKey

Indicates whether to retrieve the item's character code (`false`) or virtual keycode equivalent (`true`).

outKey

On exit, the keyboard equivalent of the item.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
A menu item's keyboard equivalent may be either a character code or a virtual keycode. An item's character code and virtual keycode are stored separately and may contain different values, but only one is used by the Menu Manager at any given time. When requesting a menu item's virtual keycode equivalent, you should first check that the item is using a virtual keycode by testing the `kMenuItemAttrUseVirtualKey` attribute for that item. If this attribute is not set, the item's virtual keycode is ignored by the Menu Manager. Note that zero is a valid virtual keycode, so you cannot test the returned keycode against zero to determine if the item is using a virtual keycode equivalent. You must test the `kMenuItemAttrUseVirtualKey` attribute.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

GetMenuItemFontID

Obtains a menu item's font ID.

```
OSErr GetMenuItemFontID (
   MenuRef inMenu,
   MenuItemIndex inItem,
   SInt16 *outFontID
);
```

Parameters

inMenu
The menu that contains the menu item for which you wish to get a font ID.

inItem
The menu index of the item.

outFontID
Pass a pointer to a signed 16-bit integer value. On return, the value is set to the font ID for the menu item.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
See also the function `SetMenuItemFontID` (page 1335).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

GetMenuItemHierarchicalID

Obtains the menu ID of a specified submenu. (Deprecated in Mac OS X v10.5.)

```
OSErr GetMenuItemHierarchicalID (
   MenuRef inMenu,
   MenuItemIndex inItem,
   MenuID *outHierID
);
```

Parameters

inMenu
The menu that contains the menu item for which you wish to get the submenu's menu ID.

inItem
The menu index of the item.

outHierID
Pass a pointer to a signed 16-bit integer value. On return, the value is set to the menu ID of the submenu.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Menus.h`

GetMenuItemHierarchicalMenu

Returns the submenu attached to a menu item.

```
OSStatus GetMenuItemHierarchicalMenu (
   MenuRef inMenu,
   MenuItemIndex inItem,
   MenuRef *outHierMenu
);
```

Parameters
inMenu
The parent menu.

inItem
The parent item.

outHierMenu
On exit, the item's submenu, or `NULL` if it does not have one.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
`GetMenuItemHierarchicalMenu` will return the submenu attached to a menu item regardless of how the submenu was specified. If the submenu was specified by menu ID (using `SetItemCmd` or `SetMenuItemHierarchicalID`), `GetMenuItemHierarchicalMenu` will return the currently installed menu with that ID, if any. The only case where `GetMenuItemHierarchicalMenu` will fail to return the item's submenu is when the submenu is specified by menu ID, but the submenu is not currently inserted in the menu bar.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

GetMenuItemIconHandle

Obtains a handle to a menu item's icon.

```
OSErr GetMenuItemIconHandle (
   MenuRef inMenu,
   MenuItemIndex inItem,
   UInt8 *outIconType,
   Handle *outIconHandle
);
```

Parameters

inMenu

The menu that contains the menu item for which you wish to obtain the handle.

inItem

The menu index of the item.

outIconType

Pass a pointer to an unsigned 8-bit value. On return, the value specifies the type of icon (`'ICON'`, `'cicn'`, `'SICN'`, icon suite, `CGImageRef`, or `IconRef`) for which you are obtaining a handle. If the menu item has no icon attached, this parameter will contain `kMenuNoIcon`. See "Menu Item Icon Type Constants" (page 1394) for descriptions of possible values.

outIconHandle

Pass a pointer to a handle. On return, `outIconHandle` contains a handle to the icon that is attached to the menu item. If the menu item has no icon attached, this parameter contains `NULL`.

Return Value

A result code. See "Menu Manager Result Codes" (page 1399).

Discussion

The `GetMenuItemIconHandle` function gets the icon handle and type of icon of the specified menu item. If you wish to get a resource-based menu item icon, call `GetItemIcon`.

See also the function `SetMenuItemIconHandle` (page 1338).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Menus.h`

GetMenuItemIndent

Gets the indent level of a menu item.

```
OSStatus GetMenuItemIndent (
   MenuRef inMenu,
   MenuItemIndex inItem,
   UInt32 *outIndent
);
```

Parameters

inMenu

The menu containing the item.

inItem
The item whose indent level you want to retrieve.

outIndent
On exit, the indent level of the item.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
The indent level of an item is an amount of extra space added to the left of the item's icon or checkmark. The level is simply a number, starting at zero, which the Menu Manager multiplies by a constant to get the indent in pixels. The default indent level is zero.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

GetMenuItemKeyGlyph

Obtains the keyboard glyph for a menu item's keyboard equivalent.

```
OSErr GetMenuItemKeyGlyph (
   MenuRef inMenu,
   MenuItemIndex inItem,
   SInt16 *outGlyph
);
```

Parameters
inMenu
The menu that contains the menu item for which you wish to get the keyboard glyph.

inItem
The menu index of the item.

outGlyph
A pointer to a signed 16-bit integer value. On return the value is set to the modifier key glyph. For a description of available keyboard glyphs, see "Menu Glyph Constants" (page 1382).

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
See also the function `SetMenuItemKeyGlyph` (page 1339).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

GetMenuItemModifiers

Obtains the modifier keys that must be pressed with a character key to select a particular menu item.

```
OSErr GetMenuItemModifiers (
   MenuRef inMenu,
   MenuItemIndex inItem,
   UInt8 *outModifiers
);
```

Parameters

inMenu

The menu that contains the menu item for which you wish to get the modifier key(s).

inItem

The menu index of the item.

outModifiers

A pointer to an unsigned 8-bit value. On return, the bits of the value are set to indicate the modifier keys that can be used in selecting the menu item; see "Modifier Key Mask Constants" (page 1396).

Return Value

A result code. See "Menu Manager Result Codes" (page 1399).

Discussion

See also the function `SetMenuItemModifiers` (page 1340).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Menus.h`

GetMenuItemProperty

Obtains a piece of data that has been previously associated with a menu item.

```
OSStatus GetMenuItemProperty (
   MenuRef menu,
   MenuItemIndex item,
   OSType propertyCreator,
   OSType propertyTag,
   ByteCount bufferSize,
   ByteCount *actualSize,
   void *propertyBuffer
);
```

Parameters

menu

The menu containing the item to be examined for associated data.

item

The index number of the menu item or 0 if the data is associated with the menu as a whole.

propertyCreator

A four-character code. Pass your program's signature (also called a creator), as registered through Apple Developer Technical Support. If your program is of a type that would not normally have a signature (for example, a plug-in), you should still register and use a signature in this case, even though your program's file may not have the same creator code as the signature that you register. The `'macs'` property signature is reserved for the system and may not be used.

propertyTag

A four-character code. Pass the application-defined code identifying the data.

bufferSize

The size of the data to be obtained. If this is unknown, use the function `GetMenuItemPropertySize` (page 1283) to get the data's size. If the size specified in the `bufferSize` parameter does not match the actual size of the property, `GetMenuItemProperty` only retrieves data up to the size specified or up to the actual size of the property, whichever is smaller, and an error is returned.

actualSize

On input, a pointer to an unsigned 32-bit integer. On return, this value is set to the actual size of the associated data. You may pass `NULL` for the `actualSize` parameter if you are not interested in this information.

propertyBuffer

On input, a pointer to a buffer. On return, this buffer contains a copy of the data that is associated with the specified menu item.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
You may use the function `GetMenuItemProperty` to obtain a copy of data previously set with the function `SetMenuItemProperty` (page 1341).

See also the `RemoveMenuItemProperty` (page 1320) function.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
HID Explorer
QTCarbonShell

Declared In
`Menus.h`

GetMenuItemPropertyAttributes

Gets the attributes of a menu item property.

```
OSStatus GetMenuItemPropertyAttributes (
   MenuRef menu,
   MenuItemIndex item,
   OSType propertyCreator,
   OSType propertyTag,
   OptionBits *attributes
);
```

Parameters

menu

The menu containing the item whose properties you want to obtain.

item

The menu index of the item.

propertyCreator

The creator code of the property.

propertyTag

The property tag.

attributes

On exit, contains the attributes of the property.

Return Value

A result code. See "Menu Manager Result Codes" (page 1399).

Discussion

Currently the only attribute you can receive from this function is the `kMenuPropertyPersistent` attribute (See "Menu Item Property Attribute Constant" (page 1395)), which currently has no effect on Mac OS X. Therefore, `GetMenuItemPropertyAttributes` is not useful at this time.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Menus.h`

GetMenuItemPropertySize

Obtains the size of a piece of data that has been previously associated with a menu item.

```
OSStatus GetMenuItemPropertySize (
   MenuRef menu,
   MenuItemIndex item,
   OSType propertyCreator,
   OSType propertyTag,
   ByteCount *size
);
```

Parameters

menu

The menu containing the item to be examined for associated data.

item

The index number of the menu item or 0 if the data is associated with the menu as a whole.

propertyCreator

A four-character code. Pass your program's signature, as registered through Apple Developer Technical Support. If your program is of a type that would not normally have a signature (for example, a plug-in), you should still register and use a signature in this case, even though your program's file may not have the same creator code as the signature that you register. The `'macs'` property signature is reserved for the system and may not be used.

propertyTag

A four-character code. Pass the application-defined code identifying the data.

size

On input, a pointer to an unsigned 32-bit integer. On return, this value is set to the actual size of the data.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
If you want to retrieve a piece of associated data with the function `GetMenuItemProperty` (page 1281) , you will typically need to use the `GetMenuItemPropertySize` function beforehand to determine the size of the associated data.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

GetMenuItemRefCon

Obtains application-specific information for a menu item.

```
OSErr GetMenuItemRefCon (
   MenuRef inMenu,
   MenuItemIndex inItem,
   URefCon *outRefCon
);
```

Parameters

inMenu

The menu that contains the menu item for which you wish to get information.

inItem

The menu index of the item. In CarbonLib 1.6 and later and Mac OS X v10.2 and later, you may pass zero to obtain the reference constant for the menu itself.

outRefCon

A pointer to an unsigned 32-bit integer value. On return, the value is set to the reference constant associated with the menu item.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
If you have assigned any data to a menu item using `SetMenuItemRefCon` (page 1342) function, you can read it using the `GetMenuItemRefCon` function.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

GetMenuItemText

Obtains the text of a menu item. (Deprecated in Mac OS X v10.5.)

Not recommended

```
void GetMenuItemText (
   MenuRef theMenu,
   MenuItemIndex item,
   Str255 itemString
);
```

Parameters

theMenu
The menu containing the item.

item
The menu index of the item.

itemString
On output, the menu item's text string.

Discussion
Unless you need to support legacy code, you should use the `CopyMenuItemTextAsCFString` (page 1243) instead.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Related Sample Code
CarbonSketch

Declared In
`Menus.h`

GetMenuItemTextEncoding

Obtains the text encoding used for a menu item's text. (Deprecated in Mac OS X v10.5.)

```
OSErr GetMenuItemTextEncoding (
   MenuRef inMenu,
   MenuItemIndex inItem,
   TextEncoding *outScriptID
);
```

Parameters

inMenu

The menu containing the menu item whose text encoding you wish to obtain.

inItem

The menu index of the item.

outScriptID

A pointer to a `TextEncoding` value. On return, the value is set to the script code of the text encoding used in the menu item's text.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
If a menu item has a command code of 0x1C when `GetMenuItemTextEncoding` is called, `GetMenuItemTextEncoding` gets the value in the icon field of the menu item.

See also the function `SetMenuItemTextEncoding` (page 1343).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Menus.h`

GetMenuRef

Obtains a menu reference corresponding to a menu ID.

```
MenuRef GetMenuHandle (
    MenuID menuID
);
#define GetMenuRef GetMenuHandle
```

Discussion
This is simply a redefinition of the `GetMenuHandle` (page 1273) function.

Declared In
`Menus.h`

GetMenuRetainCount

Returns the retain count of this menu. (Deprecated in Mac OS X v10.5.)

```
ItemCount GetMenuRetainCount (
   MenuRef inMenu
);
```

Parameters

inMenu

The menu whose retain count you want to obtain.

Return Value
The retain count for the menu.

Version Notes
In Mac OS X v10.2 and later, all menus are Core Foundation CFTypes, so you can also call `CFGetRetainCount` instead of `GetMenuRetainCount`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Menus.h`

GetMenuTitle

Obtains the title of the menu (Deprecated in Mac OS X v10.5.)

Not recommended

```
StringPtr GetMenuTitle (
   MenuRef menu,
   Str255 title
);
```

Parameters

menu

title

The menu title, as a Str255 string.

Return Value
A pointer to the menu title.

Discussion
Unless you need to support legacy code, you should use the `CopyMenuTitleAsCFString` (page 1243) function instead.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Menus.h`

GetMenuTitleIcon

Retrieves the icon, if any, being used as the title of a menu.

```
OSStatus GetMenuTitleIcon (
   MenuRef inMenu,
   UInt32 *outType,
   void **outIcon
);
```

Parameters

inMenu

The menu whose icon title to retrieve.

outType

On exit, contains the type of icon being used as the title of the menu. Contains `kMenuNoIcon` if the menu does not have an icon title.

outIcon

On exit, contains the icon reference, icon suite reference, or Core Graphics image reference of the icon being used as the title of the menu. If the menu does not have an icon title, this parameter is set to `NULL`.

Return Value

A result code. See "Menu Manager Result Codes" (page 1399).

Discussion

This function does not increment the reference count of the returned icon, so the caller should not attempt to release it.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Menus.h`

GetMenuTrackingData

Gets information about the menu currently selected by the user.

```
OSStatus GetMenuTrackingData (
   MenuRef theMenu,
   MenuTrackingData *outData
);
```

Parameters

menu

The menu about which to get tracking information. Pass `NULL` to get information about the most recently opened menu; for example, if the user has selected a menu that contains a submenu, and the submenu is open, then `GetMenuTrackingData` returns the submenu not its parent menu.

outData

On exit, contains tracking data about the menu.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399). If the menu is not currently open, `menuNotFoundErr` is returned.

Discussion
You can call this function only during menu tracking. As the standard menu definition automatically handles tracking, you would probably need this function only when writing a custom menu definition.

This function replaces direct access to the pre-Carbon low-memory globals `TopMenuItem`, `AtMenuBottom`, `MenuDisable`, and `mbSaveLoc`. See the Carbon Porting Notes for `MenuDefProcPtr` (page 1349) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Menus.h`

GetMenuType

Gets the display type (pulldown, hierarchical, or popup) of a menu.

```
OSStatus GetMenuType (
   MenuRef theMenu,
   UInt16 *outType
);
```

Parameters

theMenu
The menu whose type to get.

outType
On exit, the type of the menu. The returned value will be one of the Appearance Manager `ThemeMenuType` constants: `kThemeMenuTypePullDown`, `kThemeMenuTypePopUp`, or `kThemeMenuTypeHierarchical`. The `kThemeMenuTypeInactive` bit is never set.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
You can call this function only when the menu is displayed. If the menu is not currently open, an error is returned. The display type of a menu may vary from one menu tracking session to another; for example, the same menu might be displayed as a pulldown menu and as a popup menu.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

GetMenuWidth

Obtains the width of the menu, in pixels.

```
SInt16 GetMenuWidth (
   MenuRef menu
);
```

Parameters

menu

The menu whose width you want to obtain.

Return Value
The width of the menu, in pixels.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
Menus.h

GetNewMBar

Reads in the definition of a menu bar from an 'MBAR' resource.

Not recommended

```
MenuBarHandle GetNewMBar (
   short menuBarID
);
```

Parameters

menuBarID

The resource ID of an 'MBAR' resource that specifies the menus for a menu bar.

Return Value
A handle to the menu list. (If the resource isn't already in memory, GetNewMBar reads it into memory.) If GetNewMBar can't read the resource, GetNewMBar returns NULL. See the description of the MenuBarHandle data type.

Discussion
Unless you must support legacy code, you should not use functions like GetNewMBar that rely on menus and menu bars stored as resources. Instead, you should define menus in Interface Builder, store them as nib files, and then call the Interface Builder Services functions CreateMenuFromNib, CreateMenuBarFromNib, or SetMenuBarFromNibto create them.

The GetNewMBar function reads in the definition of a menu bar and its associated menus from an 'MBAR ' resource. The 'MBAR ' resource identifies the order of menus contained in its menu bar. For each menu, it also specifies the menu's resource ID. The GetNewMBar function reads in each menu from the 'MENU ' resource with the resource ID specified in the 'MBAR ' resource.

The GetNewMBar function creates a menu list for the menu bar defined by the 'MBAR ' resource and returns a handle to the menu list. GetNewMBar uses GetMenu (page 1266) to read in each individual menu.

After reading in menus from an 'MBAR ' resource, use SetMenuBar to make the menu list created by GetNewMBar the current menu list. Then use DrawMenuBar (page 1255) to update the menu bar.

To release the memory occupied by a menu list, use the function DisposeMenuBar (page 1253)

Special Considerations

The `GetNewMBar` function first saves the current menu list and then clears the current menu list and your application's menu color information table. It then creates a new menu list. Before returning a handle to the new menu list, the `GetNewMBar` function restores the current menu list to the previously saved menu list, but `GetNewMBar` does not restore the previous menu color information table. To save and then restore your application's current menu color information table, call the function `GetMCInfo` (page 1266) before `GetNewMBar` and call `SetMCInfo` (page 1326) afterward.

While you supply only the resource ID of an 'MBAR' resource to the `GetNewMBar` function, your application often needs to use the menu IDs defined in each of your menus' 'MENU' resources. Most Menu Manager functions require either a menu ID or a handle to a menu structure to perform operations on a specific menu. For menus in the current menu list, you can use the `GetMenuHandle` (page 1273) function to get the handle to a menu structure for a menu with a given menu ID.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Menus.h`

HideMenuBar

Conceals the menu bar.

```
void HideMenuBar (
   void
);
```

Discussion

The `HideMenuBar` function makes the menu bar invisible and unselectable by the user. You can use this function to enable full-screen display; however, in Mac OS X v10.2 and later, you should call the `SetSystemUIMode` function (available in `MacApplication.h`) instead.

Note that calling this function causes the `kEventMenuBarHidden` event to be sent to the application target (if your application has registered for the event).

Also see the `ShowMenuBar` (page 1348) and `IsMenuBarVisible` (page 1303) functions.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

HideMenuBar

Declared In

`Menus.h`

HiliteMenu

Highlights or unhighlights menu titles.

```
void HiliteMenu (
   MenuID menuID
);
```

Parameters

menuID

The menu ID of the menu whose title should be highlighted. If the menu title of the specified menu is already highlighted, `HiliteMenu` does nothing. If the menu ID is 0 or the specified menu ID isn't in the current menu list, `HiliteMenu` unhighlights whichever menu title is currently highlighted (if any).

Discussion

The `IsMenuKeyEvent` (page 1305), `MenuSelect` (page 1312), `MenuEvent` (page 1309), and `MenuKey` (page 1311) functions highlight the title of the menu containing the item chosen by the user. After performing the chosen task, your application should unhighlight the menu title by calling `HiliteMenu` and passing 0 in the `menuID` parameter.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

Simple DrawSprocket

Declared In

`Menus.h`

InitContextualMenus

Adds a program to the system registry of contextual menu clients. (Deprecated in Mac OS X v10.5.)

Not recommended

```
OSStatus InitContextualMenus (
   void
);
```

Return Value

A result code. See "Menu Manager Result Codes" (page 1399).

Discussion

Your program should call the `InitContextualMenus` function early in your startup code to register your application as a contextual menu client. If you do not register your program, some system-level functions may respond as though your program does not use contextual menus. Not registering your program may also cause `ProcessIsContextualMenuClient` (page 1317) to return an incorrect value.

Carbon Porting Notes

You do not need to call this function before using contextual menus in Mac OS X.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
Menus.h

InsertFontResMenu

Inserts menu items from a font resource. (Deprecated in Mac OS X v10.5.)

Not recommended

```
void InsertFontResMenu (
   MenuRef theMenu,
   MenuItemIndex afterItem,
   short scriptFilter
);
```

Parameters

theMenu
The menu to add the fonts to.

afterItem
The menu item after which you want to add the fonts.

scriptFilter
The script filter you want to use.

Discussion
Unless you need to do script filtering, you should use `CreateStandardFontMenu` (page 1247)instead.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
Menus.h

InsertIntlResMenu

Inserts menu items from an internationalized resource. (Deprecated in Mac OS X v10.5.)

Not recommended

```
void InsertIntlResMenu (
   MenuRef theMenu,
   ResType theType,
   MenuItemIndex afterItem,
   short scriptFilter
);
```

Parameters

theMenu
The menu to add the menu items to.

theType
The resource to retrieve the menu items from.

afterItem
The menu item after which you want to add the new items.

scriptFilter
The script filter you want to use.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Menus.h`

InsertMenu

Inserts an existing menu into the current menu list.

```
void InsertMenu (
    MenuRef theMenu,
    MenuID beforeID
);
```

Parameters

theMenu
The menu to insert.

beforeID
An integer that indicates where in the current menu list the menu should be inserted. `InsertMenu` inserts the menu into the current menu list before the menu whose menu ID is specified in the `beforeID` parameter. If the number in the `beforeID` parameter is 0 (or it isn't the ID of any menu in the menu list), `InsertMenu` adds the new menu after all others (except before the Help menu). If the menu is already in the current menu list or the menu list is already full, `InsertMenu` does nothing.

To insert a submenu into the current menu list, specify –1 or the equivalent constant `kInsertHierarchicalMenu` for the `beforeID` parameter. The submenus in the submenu portion of the menu list do not have to be currently associated with a hierarchical menu item; you can store submenus in the menu list and later specify that a menu item has a submenu if needed. However, note that during command key matching the Menu Manager scans all menus in the menu list for modifiers, including submenus that are not associated with any menu item.

You can also specify –1 for the `beforeID` parameter to insert a pop-up menu into the current menu list. However, if you use the standard pop-up control definition function, the pop-up control automatically inserts the menu into the current menu list according to the needs of the pop-up control.

Discussion
Menus inserted using this function are added to a menu list attached to the current root menu. To obtain or set this root menu, call `AcquireRootMenu` (page 1229) and `SetRootMenu` (page 1347) respectively.

Inserting a menu in the root menu (menu bar) increments its reference count; removing the menu decrements its reference count.

To change a menu title, call `SetMenuItemTextWithCFString` (page 1344).

Availability
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code
Simple DrawSprocket

Declared In
`Menus.h`

InsertMenuItem

Inserts one or more items into a menu previously created. (Deprecated in Mac OS X v10.5.)

Not recommended

```
void InsertMenuItem (
    MenuRef theMenu,
    ConstStr255Param itemString,
    short afterItem
);
```

Parameters

theMenu

The menu to which you wish to add the menu item or items.

itemString

A string that defines the characteristics of the new menu items. Note that in most cases you should store the text of a menu item in a nib file or resource, so that your menu items can be more easily localized. You can specify the contents of the `itemString` parameter using metacharacters; the function `InsertMenuItem` accepts the same metacharacters as the `AppendMenu` (page 1229) function. However, if you specify multiple items, the `InsertMenuItem` function inserts the items in the reverse of their order in the `itemString` parameter.

afterItem

The item number of the menu item after which the new menu items are to be added. Specify 0 in the `afterItem` parameter to insert the new items before the first menu item; specify the item number of a current menu item to insert the new menu items after it; specify a number greater than or equal to the last item in the menu to append the new items to the end of the menu.

Discussion
Note that unless you are supporting legacy code, you should use the `InsertMenuItemTextWithCFString` (page 1296) function instead.

If you do not define a specific characteristic of a menu item, the `InsertMenuItem` function assigns the default characteristic to the menu item. If you do not define any characteristic other than the text for a menu item, the `InsertMenuItem` function inserts the menu item so that it appears in the menu as an enabled item, without an icon or a mark, in the plain character style, and without a keyboard equivalent.

You can use `InsertMenuItem` to insert items into a menu regardless of whether the menu is in the current menu list.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Menus.h`

InsertMenuItemText

Inserts a menu item into a menu. (Deprecated in Mac OS X v10.5.)

Not recommended

```
OSStatus InsertMenuItemText (
   MenuRef menu,
   ConstStr255Param inString,
   MenuItemIndex afterItem
);
```

Parameters

menu
The menu into which the item is to be inserted.

inString
A Pascal string containing the text of the menu item to insert. You can pass a string containing any characters, and these characters will be presented verbatim in the menu item.

afterItem
The item number of the menu item after which the new menu item is to be inserted. Specify 0 to insert the new menu item at the top of the menu, before the first menu item. Specify a value greater than or equal to the last menu item to append the new item to the end of the menu.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion

Note that unless you are supporting legacy code, you should use the `InsertMenuItemTextWithCFString` (page 1296) function instead.

The `InsertMenuItemText` function inserts an enabled menu item containing the specified string into a menu, without evaluating the string for metacharacters, as the pre–Mac OS 8.5 Menu Manager function `InsertMenuItem` does. You may wish to use `InsertMenuItemText` if you have a need to present non-alphanumeric characters in a menu item.

See also the `AppendMenuItemText` (page 1231) function.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Menus.h`

InsertMenuItemTextWithCFString

Inserts a new menu item with text from a CFString.

```
OSStatus InsertMenuItemTextWithCFString (
   MenuRef inMenu,
   CFStringRef inString,
   MenuItemIndex inAfterItem,
   MenuItemAttributes inAttributes,
   MenuCommand inCommandID
);
```

Parameters

menu

The menu in which to insert the new item.

inString

The text of the new item.

inAfterItem

The item after which to insert the new item. Pass zero to insert the item at the beginning of the menu. If the index value is greater than the number of items in the menu, the item is inserted at the end of the menu.

inAttributes

The attributes of the new item.

inCommandID

The command ID of the new item.

Return Value

A result code. See "Menu Manager Result Codes" (page 1399).

Discussion

If the CFString is mutable, the Menu Manager will make its own copy of the CFString before returning from `InsertMenuItemWithCFString`. Modifying the string after calling `InsertMenuItemTextWithCFString` will have no effect on the menu item's actual text.

If the CFString is immutable, the Menu Manager increments the reference count of the string before returning.

The caller may release the string after calling `InsertMenuItemTextWithCFString`.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Menus.h`

InsertResMenu

Searches all resource files open to your application for a given resource type and inserts the names of any resources it finds in the specified menu. (Deprecated in Mac OS X v10.5.)

Not recommended

```
void InsertResMenu (
   MenuRef theMenu,
   ResType theType,
   MenuItemIndex afterItem
);
```

Parameters

theMenu

The menu to which to add the names of any resources of a given type that `InsertResMenu` finds.

theType

A four-character code that identifies the resource type for which to search.

afterItem

A number that indicates where in the menu to insert the names of any resources of the given type that `InsertResMenu` finds. Specify 0 in the `afterItem` parameter to insert the items before the first menu item; specify the item number of a menu item already in the menu to insert the items after the specified item number. If you specify a number greater than or equal to the last item in the menu, the items are inserted at the end of the menu.

Discussion

Unless you must support legacy code, you should not use functions like `InsertResMenu` that rely on searching the resource chain. Prior to Carbon, you used `InsertResMenu` primarily to create an Apple menu (by passing `'DRVR'` for the resource type or to create a font menu (by passing `'FONT'`). In Carbon, the Apple menu is created for you automatically, and you should call `CreateStandardFontMenu` (page 1247) to create a font menu.

The `InsertResMenu` function searches all resource files open to your application for resources of the type defined by the parameter `theType`. The specified menu must have been previously created using `NewMenu` (page 1314) , `GetMenu` (page 1266) , or `GetNewMBar` (page 1290). `InsertResMenu` adds the names of found resources after the specified menu item in alphabetical order; it does not alphabetize items already in the menu.

The `InsertResMenu` function does not add resources with names that begin with a period (.) or a percent sign (%) to the menu.

The `InsertResMenu` function assigns default characteristics to each menu item. Each appended menu item appears in the menu as an enabled item, without an icon or a mark, in the plain character style, and without a keyboard equivalent.

If you specify that `InsertResMenu` add resources of type `'DRVR '` to your Apple menu, `InsertResMenu` adds the name (and icon) of each item in the Apple Menu Items folder to the menu.

If you specify that `InsertResMenu` add resources of type `'FONT '` or `'FOND '`, the Menu Manager performs special processing for any resources it finds that have font numbers greater than 0x4000. If the script associated with the font name is currently active, `InsertResMenu` stores information in the `itemDefinitions` array (in the `itemIcon` and `itemCmd` fields for that item) in the menu structure that allows the Menu Manager to display the font name in the correct script.

Special Considerations

The `InsertResMenu` function calls the Resource Manager function `SetResLoad` (specifying `true` in the `load` parameter) before returning. The `InsertResMenu` function reads the resource data of the resources it finds into memory. If your application does not want the Resource Manager to read resource data into memory when your application calls other functions that read resources, you need to call the Resource Manager function `SetResLoad` and specify `false` in the `load` parameter after `InsertResMenu` returns.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Menus.h`

InvalidateMenuEnabling

Requests that the menu's enable state be recalculated.

```
OSStatus InvalidateMenuEnabling (
   MenuRef inMenu
);
```

Parameters

inMenu

The menu whose enable states should be recalculated. Pass `NULL` to recalculate enable states for all menus in the root menu.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
State changes in an application can cause changes in enable state of a menu's menu items. For example, selecting a block of text would usually require the Copy menu item to become enabled. Menu items are typically updated in response to a `kEventCommandUpdateStatus` Carbon event. However, the update event usually occurs only before an command key press or a click in the menu bar. To explicitly update the enable states for a menu, you can call `InvalidateMenuEnabling`.

Note that the Carbon Event Manager automatically requests recalculation of enable states for all top-level menus when

- it dispatches a user event
- the user focus changes
- the active window changes

so in many cases you may not need to explicitly call this function.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

InvalidateMenuItems

Invalidates a group of menu items so that they will be redrawn when `UpdateInvalidMenuItems` is next called.

```
OSStatus InvalidateMenuItems (
   MenuRef inMenu,
   MenuItemIndex inFirstItem,
   ItemCount inNumItems
);
```

Parameters

inMenu

The menu whose items to invalidate.

inFirstItem

The first item to invalidate.

inNumItems

The number of items to invalidate.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
Menu items are automatically invalidated when their contents are changed using Menu Manager APIs while the menu is open. However, you might need to use this function if you have a custom menu definition that draws based on information not contained in the menu. For example, say you have a custom menu definition that draws a menu item using a color stored elsewhere in an application-defined data structure. If you change the color in the data structure, there is no way for the menu definition to know this has occurred. Calling `InvalidateMenuItems` after changing the color in the structure indicates that the menu should be redrawn.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

InvalidateMenuSize

Invalidates the menu size so that it will be recalculated when next displayed.

```
OSStatus InvalidateMenuSize (
   MenuRef inMenu
);
```

Parameters

inMenu

The menu whose size you want to invalidate.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
Prior to Carbon, the technique for invalidating the menu size was to set the width and height to -1. Although this technique still works, for best compatibility you should call `InvalidateMenuSize` so that the Menu Manager has explicit notification that the menu is invalid.

Availability
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
`Menus.h`

InvalMenuBar

Invalidates the menu bar.

```
void InvalMenuBar (
   void
);
```

Discussion
The `InvalMenuBar` function marks the menu bar as changed and in need of updating. When the Carbon Event Manager checks incoming events for regions that require updating, the Carbon Event Manager also checks to determine whether the menu bar requires updating (because of a call to `InvalMenuBar`). If the menu bar needs updating, the Carbon Event Manager calls the `DrawMenuBar` (page 1255) function to draw the menu bar.

You can use `InvalMenuBar` instead of `DrawMenuBar` to minimize blinking in the menu bar. For example, if you have several application-defined functions that can change the enabled state of a menu and each calls `DrawMenuBar`, you can replace the calls to `DrawMenuBar` with calls to `InvalMenuBar`. In this way the menu bar is redrawn only once instead of multiple times in quick succession. If you need to make immediate changes to the menu bar, use `DrawMenuBar`. If you want to redraw the menu bar at most once each time through your event loop, use `InvalMenuBar`. Note. however, that most Menu Manager calls that affect the menu bar call `InvalMenuBar` (page 1301) before returning. so you probably won't need to call it yourself.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

InvokeMenuDefUPP

Calls your custom menu definition through a universal procedure pointer. (Deprecated in Mac OS X v10.5.)

Not recommended

```
void InvokeMenuDefUPP (
   short message,
   MenuRef theMenu,
   Rect *menuRect,
   Point hitPt,
   short *whichItem,
   MenuDefUPP userUPP
);
```

Parameters

message

theMenu

menuRect

hitPt

whichItem

userUPP

Carbon Porting Notes

Apple discourages you from writing and using your own menu definition functions and encourages you to use the system-supplied menu definition function instead. New features that have previously been missing are now available in the system-supplied menu definition function. Since Appearance Manager 1.0 (in Mac OS 8.0), for example, the system-supplied menu definition function has supported extended menu item command key modifiers and glyphs. And in Carbon, the system-supplied menu definition function supports dynamic items, which allow the contents of a menu item to be redrawn while the menu is displayed in response to the user pressing a modifier key on the keyboard.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Menus.h`

IsMenuBarInvalid

Determines if the menubar is invalid and should be redrawn.

```
Boolean IsMenuBarInvalid (
   MenuRef rootMenu
);
```

Parameters

rootMenu

The root menu for the menubar to be examined. Pass `NULL` to check the state of the current menubar.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Menus.h`

IsMenuBarVisible

Reports whether the menu bar is currently visible.

```
Boolean IsMenuBarVisible (
   void
);
```

Return Value
Returns `true` if the menu bar is currently visible; otherwise, `false`.

Discussion
Also see the `HideMenuBar` (page 1291) and `ShowMenuBar` (page 1348) functions.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

IsMenuCommandEnabled

Determines if the menu item with a specified command ID is enabled.

```
Boolean IsMenuCommandEnabled (
   MenuRef inMenu,
   MenuCommand inCommandID
);
```

Parameters

inMenu
The menu in which to begin searching for the item. Pass `NULL` to begin searching with the root menu. The search will descend into all submenus of this menu.

inCommandID
The command ID of the menu item to examine. If more than one item has this command ID, only the first will be examined.

Discussion
If you have access to the menu item index, in most cases you should use `IsMenuItemEnabled` (page 1303) instead, as that function is faster and requires no searching.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

IsMenuItemEnabled

Reports whether a given menu or menu item is enabled.

```
Boolean IsMenuItemEnabled (
   MenuRef menu,
   MenuItemIndex item
);
```

Parameters

menu

The menu containing the item to be examined.

item

The item number of the menu item. Pass 0 to specify the menu title and determine whether the menu as a whole is enabled.

Return Value
Returns `true` if the menu item is currently enabled; otherwise, `false`.

Discussion
Your application can use the `IsMenuItemEnabled` function to determine whether specific menu items, even those with item numbers greater than 31, are currently enabled and can therefore be selected by the user.

Note that this function ignores the enable state of the menu when returning the enable state of a menu item. For example, if you call `IsMenuItemEnabled` on an enabled item while its parent menu is disabled, the function still returns `true`.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

IsMenuItemIconEnabled

Reports whether a given menu item icon is enabled.

```
Boolean IsMenuItemIconEnabled (
   MenuRef menu,
   MenuItemIndex item
);
```

Parameters

menu

The menu containing the icon to be examined.

item

The item number of the menu item containing the icon.

Return Value
Returns `true` if the menu item icon is currently enabled; otherwise, `false`.

Discussion
Your application can use the `IsMenuItemIconEnabled` function to determine whether a specific menu item's icon is currently enabled or dimmed.

See also the functions `DisableMenuItemIcon` (page 1252) and `EnableMenuItemIcon` (page 1258).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

IsMenuItemInvalid

Determines if a menu item is invalid and should be redrawn.

```
Boolean IsMenuItemInvalid (
   MenuRef inMenu,
   MenuItemIndex inItem
);
```

Parameters

inMenu
The menu whose item to examine.

inItem
The item to examine.

Return Value
Returns `true` if the menu is invalid, `false` otherwise.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

IsMenuKeyEvent

Determines if an event corresponds to a menu command key.

```
Boolean IsMenuKeyEvent (
   MenuRef inStartMenu,
   EventRef inEvent,
   MenuEventOptions inOptions,
   MenuRef *outMenu,
   MenuItemIndex *outMenuItem
);
```

Parameters

inStartMenu
The menu to search. `IsMenuKeyEvent` searches for matching menu items in this menu and all of its submenus. Pass `NULL` to search the current menu bar contents.

inEvent
The event to match against. Non-keyboard events are ignored.

inOptions
Options controlling how to search. See "Menu Event Option Constants" (page 1381) for possible options. Pass `kNilOptions` for the default behavior.

outMenu
On exit, the menu containing the matching item. Set to `NULL` if no match was made.

outMenuItem
On exit, the menu item that matched. Set to `NULL` if no match was made.

Discussion
By default, `IsMenuKeyEvent` searches the menus in the current menu bar and highlights the menu title of the menu containing the selected item.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

IsMenuSizeInvalid

Determines if a menu's size is invalid and should be recalculated.

```
Boolean IsMenuSizeInvalid (
   MenuRef inMenu
);
```

Parameters

inMenu
The menu whose size you want to examine.

Return Value
Set to `true` if the menu size is invalid, `false` otherwise.

Discussion
Prior to Carbon, the technique for determining if a menu's size is invalid was to check if the width or height was -1. This technique is not always reliable in Carbon due to implementation changes in the Menu Manager, so you should use `IsMenuSizeInvalid` instead.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

IsShowContextualMenuClick

Determines whether a particular event could invoke a contextual menu.

Not recommended

```
Boolean IsShowContextualMenuClick (
   const EventRecord *inEvent
);
```

Parameters

inEvent

A pointer to the event structure that describes the event to examine.

Return Value

If `true`, the contextual menu should be displayed; if `false`, not.

Discussion

Before calling the `IsShowContextualMenuClick` function, you should call `InitContextualMenus` (page 1292). If no error is returned, you can then call `IsShowContextualMenuClick`.

Unless you must support the legacy `WaitNextEvent` event model, you should use the Carbon event–based `IsShowContextualMenuEvent` (page 1307) function instead. In addition, some users may choose to use a two-button mouse with their Macintosh computers, in which case a right-click does not return `true` with `IsShowContextualMenuClick` unless the mouse manufacturer's driver software deliberately returns Control-left-click in place of the right-click. If you want to properly recognize the right-click to invoke a contextual menu, you should use the `IsShowContextualMenuEvent` (page 1307) function.

Applications should call `IsShowContextualMenuClick` when they receive non-null events. If `IsShowContextualMenuClick` returns `true`, your application should generate its own menu and Apple Event descriptor (`AEDesc`), and then call `ContextualMenuSelect` (page 1239) to display and track the contextual menu, and then handle the user's choice.

If the mouse-down event did not invoke a contextual menu, then the application should check to see if the event occurred in the menu bar (using the `FindWindow` function) and, if so, call `MenuSelect` to allow the user to choose a command from the menu bar.

See also "Contextual Menu Gestalt Selector Constants" (page 1368).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Menus.h`

IsShowContextualMenuEvent

Determines whether a particular Carbon event could invoke a contextual menu.

```
Boolean IsShowContextualMenuEvent (
   EventRef inEvent
);
```

Parameters

inEvent

The event to examine.

Return Value

Returns `true` if the application should display a contextual menu, `false` otherwise.

Discussion
If your application supports Carbon events, you should use this function in place of the older `IsShowContextualMenuClick` (page 1306). This function also properly supports right-click activation of contextual menus, while `IsShowContextualMenuClick` (page 1306) does not.

However, if your application uses the standard window handler, you probably don't need to use `IsShowContextualMenuEvent`, because the standard window handler automatically detects contextual menu clicks and sends the `kEventWindowContextualMenuSelect` and `kEventControlContextualMenuClick` events.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

IsValidMenu

Determines if a menu is valid.

```
Boolean IsValidMenu (
   MenuRef inMenu
);
```

Parameters

inMenu
The menu to check for validity.

Return Value
Indicates whether the menu is valid.

Discussion
The Menu Manager keeps a table that maps menu references to their corresponding (opaque) `MenuData` structures. A menu is considered valid if its menu reference appears in that table.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

LMGetTheMenu

Returns the menu ID of the currently highlighted menu in the menu bar.

```
MenuID LMGetTheMenu (
   void
);
```

Return Value
A value which contains the menu ID of the currently highlighted menu in the menu bar.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

MenuChoice

Returns the menu ID and index of the menu item under the cursor.

```
SInt32 MenuChoice (
   void
);
```

Return Value
The high-order word of the function result contains the menu ID of the menu, and the low-order word contains the item number of the menu item chosen by the user. The `MenuChoice` function returns 0 as the low-order word of its function result if the mouse button was released while the cursor was in the menu bar or outside the menu.

Discussion
You can use this function in the rare cases that you want to take action based upon the selection of a disabled menu item. For example, say the user is in contextual help mode (activated by pressing the Help key) and wants to get help information about a disabled menu item. You can use `MenuChoice` to determine which item the user selected and provide help for that item.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

MenuEvent

Maps a keyboard combination from the event structure to the keyboard equivalent of a menu item in a menu in the current menu list.

Not recommended

```
UInt32 MenuEvent (
   const EventRecord *inEvent
);
```

Parameters
inEvent
A pointer to the event structure containing the keyboard equivalent.

Return Value
A value that indicates the menu ID (in the high 16 bits) and menu item (in the low 16 bits) that the user chose. If the given character does not map to an enabled menu item in the current menu list, `MenuEvent` returns 0 in its high-order word and the low-order word is undefined.

Discussion
Note that unless your application uses the `WaitNextEvent` event model, you should use the Carbon event–based `IsMenuKeyEvent` (page 1305) instead.

The `MenuEvent` function does not distinguish between uppercase and lowercase letters. This allows a user to invoke a keyboard equivalent command, such as the Copy command, by pressing the Command key and "c" or "C". For consistency between applications, you should define the keyboard equivalents of your commands so that they appear in uppercase in your menus.

If the given character maps to an enabled menu item in the current menu list, `MenuEvent` highlights the menu title of the chosen menu, returns the menu ID in the high-order word of its function result, and returns the chosen menu item in the low-order word of its function result. After performing the chosen task, your application should unhighlight the menu title using the `HiliteMenu` function.

You should not define menu items with identical keyboard equivalents. The `MenuEvent` function scans the menus from right to left and the items from top to bottom. If you have defined more than one menu item with identical keyboard equivalents, `MenuEvent` returns the first one it finds.

The `MenuEvent` function first searches the regular portion of the current menu list for a menu item with a keyboard equivalent matching the given key. If it doesn't find one there, it searches the submenu portion of the current menu list. If the given key maps to a menu item in a submenu, `MenuEvent` highlights the menu title in the menu bar that the user would normally pull down to begin traversing to the submenu. Your application should perform the desired command and then unhighlight the menu title.

Note that some keyboard equivalents are reserved for use by the system, such as Command–Shift–3 annd Command-Shift-4 (for taking screen shots).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

MenuHasEnabledItems

Determines if any items in a menu are enabled.

```
Boolean MenuHasEnabledItems (
   MenuRef theMenu
);
```

Parameters

theMenu
The menu whose items to examine.

Return Value
Returns `true` if one or more items in the menu are enabled, `false` otherwise.

Discussion
This function is equivalent to pre-Carbon code that compared the `enableFlags` field of the `MenuInfo` structure (now opaque in Carbon) with 0. It checks the enable state of all items to see if any are enabled, but ignores the state of the menu title. It will return `true` even if the menu title is disabled.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

MenuKey

Maps a character key with the command key to determine the keyboard equivalent of a menu item in a menu in the current menu list. (Deprecated in Mac OS X v10.5.)

Not recommended

```
SInt32 MenuKey (
   CharParameter ch
);
```

Parameters

ch

The 1-byte character that represents the key pressed by the user in combination with the Command key.

Return Value
A value containing the menu ID and menu item that corresponds to the given character.If the given character does not map to an enabled menu item in the current menu list, `MenuKey` returns 0 in its high-order word and the low-order word is undefined.

Discussion
Note that unless your application uses the `WaitNextEvent` event model, you should use the Carbon event–based `IsMenuKeyEvent` (page 1305) instead. Even when using the `WaitNextEvent` model, you should probably use `MenuEvent` (page 1309), as that function supports the Shift, Option, and Control modifier keys in addition to the Command key

The `MenuKey` function determines whether the key combination maps to a current menu item when the user presses another key while holding down the Command key.

`MenuKey` does not distinguish between uppercase and lowercase letters. This allows a user to invoke a keyboard equivalent command, such as the Copy command, by pressing the Command key and “c” or “C”. For consistency between applications, you should define the keyboard equivalents of your commands so that they appear in uppercase in your menus.

You should not define menu items with identical keyboard equivalents. The `MenuKey` function scans the menus from right to left and the items from top to bottom. If you have defined more than one menu item with identical keyboard equivalents, `MenuKey` returns the first one it finds.

If the given character maps to an enabled menu item in the current menu list, `MenuKey` highlights the menu title of the chosen menu, returns the menu ID in the high-order word of its function result, and returns the chosen menu item in the low-order word of its function result. After performing the chosen task, your application should unhighlight the menu title using the `HiliteMenu` function.

The `MenuKey` function first searches the regular portion of the current menu list for a menu item with a keyboard equivalent matching the given key. If it doesn't find one there, it searches the submenu portion of the current menu list. If the given key maps to a menu item in a submenu, `MenuKey` highlights the menu title in the menu bar that the user would normally pull down to begin traversing to the submenu. Your application should perform the desired command and then unhighlight the menu title.

You shouldn't assign a Command–Shift–number key sequence to a menu item as its keyboard equivalent Command–Shift–number key sequences are reserved for use as `'FKEY '` resources. Command–Shift–number key sequences are not returned to your application, but instead are processed by the Event Manager. The Event Manager invokes the `'FKEY '` resource with a resource ID that corresponds to the number that activates it.

Apple reserves the Command-key codes 0x1B through 0x20 to indicate meanings other than keyboard equivalents. `MenuKey` ignores these character codes and returns a function result of 0 if you specify any of these values in the `ch` parameter. Your application should not use these character codes for its own use

Carbon Porting Notes

Carbon does not support desk accessories, so `MenuKey` cannot be used in OS X to pass keyboard equivalents to desk accessories.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Related Sample Code

Simple DrawSprocket

Declared In

`Menus.h`

MenuSelect

Allows the user to choose a menu item from a menu in the menu bar.

```
SInt32 MenuSelect (
   Point startPt
);
```

Parameters

`startPt`

The location of the cursor at the time the mouse button was first pressed, in global coordinates. Your application retrieves this point from the `kEventParamMouseLocation` parameter of a Carbon event, or the `where` field of the `EventRecord` structure.

Return Value

If the user chooses an enabled menu item (including any item from a submenu), the `MenuSelect` function returns a value as its function result that indicates which menu and menu item the user chose. The high-order word of the function result contains the menu ID of the menu, and the low-order word contains the item number of the menu item chosen by the user.

Discussion
If your application uses Carbon events, you probably don't need to use this function. The standard event handler installed by the Carbon Event Manager function `RunApplicationEventLoop` calls `MenuSelect` for you to begin tracking when the user clicks in a menu. If you do not call `RunApplicationEventLoop`, however, you will need to use this function to initiate menu tracking.

When the user presses the mouse button while the cursor is in the menu bar, your application receives a mouse-down event. To handle mouse-down events in the menu bar, pass the location of the cursor at the time of the mouse-down event as the `startPt` parameter to `MenuSelect`. The `MenuSelect` function displays and removes menus as the user moves the cursor over menu titles in the menu bar, and it handles all user interaction until the user dismisses the menu.

As the user drags the cursor through the menu bar, the `MenuSelect` function highlights the title of the menu the cursor is currently over and displays all items in that menu. If the user moves the cursor so that it is over a different menu, the `MenuSelect` function removes the previous menu and unhighlights its menu title.

The `MenuSelect` function highlights and unhighlights menu items as the user drags the cursor over the items in a menu. The `MenuSelect` function highlights a menu item if the item is enabled and the cursor is currently over it; it removes such highlighting when the user moves the cursor to another menu item. The `MenuSelect` function does not highlight disabled menu items.

If the user chooses an enabled menu item (including any item from a submenu), the `MenuSelect` function returns a value as its function result that indicates which menu and menu item the user chose. The high-order word of the function result contains the menu ID of the menu, and the low-order word contains the item number of the menu item chosen by the user.The `MenuSelect` function leaves the menu title highlighted; after performing the chosen task your application should unhighlight the menu title using the `HiliteMenu` (page 1291) function.

If the user chooses an item from a submenu, `MenuSelect` returns the menu ID of the submenu in the high-order word and the item chosen by the user in the low-order word of its function result. The `MenuSelect` function also highlights the title of the menu in the menu bar that the user originally displayed in order to begin traversing to the submenu. After performing the chosen task, your application should unhighlight the menu title.

If the user releases the mouse button while the cursor is over a disabled item, in the menu bar, or outside of any menu, the `MenuSelect` function returns 0 in the high-order word of its function result and the low-order word is undefined. If it is necessary for your application to find the item number of the disabled item, your application can call `MenuChoice` (page 1309) to return the menu ID and menu item.

Note that `MenuSelect` sends a number of Carbon events during menu tracking:

- `kEventMenuBeginTracking` when tracking begins, before displaying any menus. An event handler for this event may return `userCanceledErr` to prevent menu tracking from occurring.
- `kEventMenuEndTracking` after tracking was completed.
- `kEventMenuChangeTrackingMode` when the user switches from the mouse to the keyboard to navigate through menus, or vice versa.
- `kEventMenuOpening` is sent to each menu that the user opens, just before it opens. An event handler for this event may return `userCanceledErr` to prevent menu tracking from occurring.
- `kEventMenuClosed` is sent to each menu that the user closes.
- `kEventMenuPopulate` is sent to a menu just before it is opened in a menu tracking session. This event is sent only once per menu tracking session.

- `kEventMenuEnableItems` is sent to a menu just before it is opened in a menu tracking session. This event is sent only once per menu tracking session.
- `kEventCommandUpdateStatus` is sent to a menu just before it is opened in a menu tracking session. This event is sent only once per menu tracking session.
- `kEventMenuTargetItem` is sent to a menu when the mouse passes over a menu item or if the keyboard is used to select a menu item. This event is sent for both enabled and disabled menu items, and is also sent when the mouse is over a menu title.
- `kEventCommandProcess` is sent to a menu when the user chooses one of its menu items.

For more details about these Carbon events, see the *Carbon Event Manager Reference*.

Carbon Porting Notes

Carbon does not support desk accessories, so `MenuSelect` cannot be used in Mac OS X to pass keyboard equivalents to desk accessories.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
Simple DrawSprocket

Declared In
`Menus.h`

NewMenu

Creates an empty menu with a specified title and menu ID. (Deprecated in Mac OS X v10.5.)

Not recommended

```
MenuRef NewMenu (
   MenuID menuID,
   ConstStr255Param menuTitle
);
```

Parameters

`menuID`

The ID for the menu. The menu ID is a number that identifies the menu. Menu IDs in Carbon can be any value, but Apple recommends that the ID be either zero or positive. A menu ID of zero is a valid ID. IDs of submenus should similarly be zero or a positive value.

`menuTitle`

The title of the new menu. Note that in most cases you should store the titles of menus in resources, so that your menu titles can be more easily localized.

Return Value
A menu reference. If the `NewMenu` function is unable to create the menu structure, it returns `NULL`. See the description of the `MenuRef` data type.

Discussion
Unless you must support legacy code, you should not use functions like `NewMenu`. If you need to create menus programmatically, you can call `CreateNewMenu` (page 1246); otherwise you should define menus in Interface Builder, store them as nib files, and then call the Interface Builder Services function `CreateMenuFromNib` to create them.

The `NewMenu` function creates a menu with the specified title, assigns it the specified menu ID, and returns a handle to the menu structure. It sets up the menu structure to use the standard menu definition function (and it reads the standard menu definition function into memory if it isn't already there). `NewMenu` does not insert the newly created menu into the current menu list.

After creating a menu with `NewMenu`, use `AppendMenu` (page 1229) , `InsertMenuItem` (page 1295) , `AppendResMenu` (page 1232) , or `InsertResMenu` (page 1297) to add menu items to the menu. To add a menu created by `NewMenu` to the current menu list, use `InsertMenu` (page 1294). In Carbon, you do not need to call `DrawMenuBar` (page 1255) to update the menu bar, as the Menu Manager automatically invalidates andredraws the menu bar.

Menus in a resource must not be purgeable nor should the resource lock bit be set. Do not define a "circular" hierarchical menu—that is, a hierarchical menu in which a submenu has a submenu whose submenu is a hierarchical menu higher in the chain.

Special Considerations

To release the memory associated with a menu that you created using `NewMenu`, first call `DeleteMenu` (page 1248) to remove the menu from the current menu list and to remove any entries for this menu in your application's menu color information table then call `DisposeMenu` (page 1253) to dispose of the menu structure. Note that in Carbon, the Menu Manager automatically invalidates and redraws the menu bar after disposing of a menu.

Version Notes
Note that if you are running on Mac OS 8.1 and earlier, the menu ID of a submenu must be within the range 0 to 255.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Related Sample Code
Simple DrawSprocket

Declared In
`Menus.h`

NewMenuDefUPP

Creates a new univeral procedure pointer to your custom menu definition. (Deprecated in Mac OS X v10.5.)

Not recommended

```
MenuDefUPP NewMenuDefUPP (
   MenuDefProcPtr userRoutine
);
```

Parameters

userRoutine

Return Value
See the description of the `MenuDefUPP` data type.

Carbon Porting Notes

Apple discourages you from writing and using your own menu definition functions and encourages you to use the system-supplied menu definition function instead. New features that have previously been missing are now available in the system-supplied menu definition function. Since Appearance Manager 1.0 (in Mac OS 8.0), for example, the system-supplied menu definition function has supported extended menu item command key modifiers and glyphs. And in Carbon, the system-supplied menu definition function supports dynamic items, which allow the contents of a menu item to be redrawn while the menu is displayed in response to the user pressing a modifier key on the keyboard.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`Menus.h`

PopUpMenuSelect

Displays a pop-up menu without using the standard pop-up control definition function.

```
SInt32 PopUpMenuSelect (
   MenuRef menu,
   short top,
   short left,
   MenuItemIndex popUpItem
);
```

Parameters

menu

The pop-up menu to be displayed.

top

The y-coordinate of the top-left corner of the selected menu item when the menu opens. This value should be in global coordinates.

left

The x-coordinate of the top-left corner of the selected menu item when the menu opens. This value should be in global coordinates.

popUpItem

The index of the menu item to display at the global point (`top`, `left`). If you pass zero, the first menu item is positioned at the indicated point.

Return Value
The menu ID of the chosen menu in the high 16-bit word of the function result and the chosen menu item index value in the low 16-bit word. If no item was selected, the result is zero.

Discussion
If your application uses the standard pop-up control definition function, your application does not need to use the `PopUpMenuSelect` function. `PopUpMenuSelect` uses the location specified by the `top` and `left` parameters to determine where to display the specified item of the pop-up menu. `PopUpMenuSelect` displays the pop-up menu so that the menu item specified in the `popUpItem` parameter appears highlighted at the specified location.

The `PopUpMenuSelect` function highlights and unhighlights menu items and handles all user interaction until the user releases the mouse button.

Note that `PopUpMenuSelect` sends a number of Carbon events during menu tracking (these are the same as those sent for `MenuSelect` (page 1312)):

- `kEventMenuBeginTracking` when tracking begins, before displaying any menus. An event handler for this event may return `userCanceledErr` to prevent menu tracking from occurring.
- `kEventMenuEndTracking` after tracking was completed.
- `kEventMenuChangeTrackingMode` when the user switches from the mouse to the keyboard to navigate through menus, or vice versa.
- `kEventMenuOpening` is sent to each menu that the user opens, just before it opens. An event handler for this event may return `userCanceledErr` to prevent menu tracking from occurring.
- `kEventMenuClosed` is sent to each menu that the user closes.
- `kEventMenuPopulate` is sent to a menu just before it is opened in a menu tracking session. This event is sent only once per menu tracking session.
- `kEventMenuEnableItems` is sent to a menu just before it is opened in a menu tracking session. This event is sent only once per menu tracking session.
- `kEventCommandUpdateStatus` is sent to a menu just before it is opened in a menu tracking session. This event is sent only once per menu tracking session.
- `kEventMenuTargetItem` is sent to a menu when the mouse passes over a menu item or if the keyboard is used to select a menu item. This event is sent for both enabled and disabled menu items, and is also sent when the mouse is over a menu title.
- `kEventCommandProcess` is sent to a menu when the user chooses one of its menu items.

For more details about these Carbon events, see the *Carbon Event Manager Programming Guide*.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

ProcessIsContextualMenuClient

Determines whether a given program is a contextual menu client. (Deprecated in Mac OS X v10.5.)

Not recommended

```
Boolean ProcessIsContextualMenuClient (
   ProcessSerialNumber *inPSN
);
```

Parameters

inPSN

A pointer to the ID of the process containing the program.

Return Value

`true` if the program in the process uses contextual menus; otherwise, `false`.

Discussion

In Mac OS X, you do not need to call `InitContextualMenus` (page 1292) to register your application, so this function call is unnecessary.

The `ProcessIsContextualMenuClient` function checks the system registry of contextual menu clients and returns `true` if the program in the given process supports contextual menus. However, the program must have been registered as a client using `InitContextualMenus` (page 1292).

See also "Contextual Menu Gestalt Selector Constants" (page 1368).

Version Notes

This function is available with Appearance Manager 1.0 and later.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Menus.h`

RegisterMenuDefinition

Registers a binding between a resource ID and a menu definition function.

```
OSStatus RegisterMenuDefinition (
   SInt16 inResID,
   MenuDefSpecPtr inDefSpec
);
```

Parameters

inResID

An MDEF resource ID, as used in a `'MENU'` resource.

inDefSpec

Specifies the `MenuDefUPP` that should be used for menus with the given MDEF proc ID. Pass `NULL` if you want to unregister the menu definition.

Return Value

A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
Mac OS X does not allow you to store custom menu definitions in resources. However, some older functions such as `GetMenu` expect an `'MDEF'` resource ID to be in the `'MENU'` resource when creating menus . To work around this, you can use `RegisterMenulDefinition` to register "virtual" resource IDs for your menu definition functions. When `GetMenu` (or a similar function) attempts to access an `'MDEF'` resource, the Menu Manager uses the menu definition specified by the `MenuDefSpec` structure instead.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

ReleaseMenu

Decrements the retain count of a menu. (Deprecated in Mac OS X v10.5.)

```
OSStatus ReleaseMenu (
   MenuRef inMenu
);
```

Parameters

inMenu
The menu whose retain count to decrement. If the retain count falls to zero, the menu is destroyed.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
The reference that you pass in the `theMenu` parameter is not valid after `DisposeMenu` returns. This function is identical to `DisposeMenu` (page 1253).

Version Notes
In Mac OS X v10.2 and later, all menus are Core Foundation CFTypes, so you can optionally call `CFRelease` instead of `ReleaseMenu`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Related Sample Code
QTCarbonShell

Declared In
`Menus.h`

RemoveMenuCommandProperty

Removes a property from a menu item with the specified command ID.

```
OSStatus RemoveMenuCommandProperty (
   MenuRef inMenu,
   MenuCommand inCommandID,
   OSType inPropertyCreator,
   OSType inPropertyTag
);
```

Parameters

inMenu

The menu in which to begin searching for the item. Pass `NULL` to begin searching with the root menu. The search will descend into all the submenus of this menu.

inCommandID

The command ID of the menu item whose property you want to remove. If more than one item has the same command ID, only the first will be modified.

inPropertyCreator

The four-character creator code for the application.

inPropertyTag

The four-character tag identifying the property to remove.

Return Value

A result code. See "Menu Manager Result Codes" (page 1399).

Discussion

If you have access to the menu item index, in most cases you should use `RemoveMenuItemProperty` (page 1320) instead, as that function is faster and requires no searching.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Menus.h`

RemoveMenuItemProperty

Removes a piece of data that has been previously associated with a menu item.

```
OSStatus RemoveMenuItemProperty (
   MenuRef menu,
   MenuItemIndex item,
   OSType propertyCreator,
   OSType propertyTag
);
```

Parameters

menu

The menu containing the item whose associated data is to be removed.

item

The index number of the menu item or 0 if the data is associated with the menu as a whole.

propertyCreator

A four-character code. Pass your program's signature, as registered through Apple Developer Technical Support. If your program is of a type that would not normally have a signature (for example, a plug-in), you should still register and use a signature in this case, even though your program's file may not have the same creator code as the signature that you register. The `'macs'` property signature is reserved for the system and may not be used.

propertyTag

A four-character code. Pass the application-defined code identifying the associated data.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
Your application may remove data set with the `SetMenuItemProperty` (page 1341) function by calling the `RemoveMenuItemProperty` function.

When a menu is destroyed, all of its properties are removed automatically.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

RetainMenu

Increments the reference count of a menu. (Deprecated in Mac OS X v10.5.)

```
OSStatus RetainMenu (
   MenuRef inMenu
);
```

Parameters

inMenu

The menu whose reference count you want to increment.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
`RetainMenu` does not create a new menu. It simply adds one to the reference count.

Version Notes
In Mac OS X v10.2 and later, all menus are Core Foundation CFTypes, so you can optionally call `CFRetain` instead of `RetainMenu`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Menus.h`

ScrollMenuImage

Scrolls a portion of the menu image. (Deprecated in Mac OS X v10.5.)

```
OSStatus ScrollMenuImage (
   MenuRef inMenu,
   const Rect *inScrollRect,
   int inHScroll,
   int inVScroll,
   CGContextRef inContext
);
```

Parameters

inMenu

The menu to scroll.

inScrollRect

The bounds of the area to scroll.

inHScroll

The distance to scroll horizontally, in pixels.

inVScroll

The distance to scroll vertically, in pixels.

inContext

The Core Graphics context to scroll. If you pass `NULL`, the function creates a context based on the current port.

Return Value

A result code. See "Menu Manager Result Codes" (page 1399).

Discussion

Scrolling menus on Mac OS X using `ScrollRect` or other QuickDraw functions destroys the alpha channel data, so you should use `ScrollMenuImage` instead.

Availability

Available in Mac OS X v10.1 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Menus.h`

SetItemCmd

Sets the value of the keyboard equivalent field of a menu item.

Not recommended

```
void SetItemCmd (
   MenuRef theMenu,
   MenuItemIndex item,
   CharParameter cmdChar
);
```

Parameters

theMenu

The menu containing the item.

item

The menu index of the item.

cmdChar

The value to set for the item's keyboard equivalent field. The Menu Manager uses this value to map keyboard equivalents to menu s or to define special characteristics of the menu item.

To indicate that the menu item has a submenu, specify 0x1B in the `cmdChar` parameter; specify a value of 0x1C to indicate that the item has a special text encoding; specify a value of 0x1D to indicate that the Menu Manager should reduce the item's `'ICON'` resource to the size of a small icon; and specify a value of 0x1E to indicate that the item has an `'SICN'` resource.

The values 0x01 through 0x1A, as well as 0x1F and 0x21, are reserved for use by Apple. You should not use any of these reserved values in the `cmdChar` parameter.

Discussion

You should call `SetMenuItemCommandKey` (page 1334) , `SetMenuItemHierarchicalID` (page 1336) , and `SetMenuItemTextEncoding` (page 1343) instead of `SetItemCmd` to set a menu item's keyboard equivalent and text encoding and to indicate that a menu item has a submenu.

You usually define the keyboard equivalents and other characteristics of your menu items in the `'MENU'` resource rather than using the `SetItemCmd` function. The `SetItemCmd` function sets the value in the keyboard equivalent field of the specified menu item to the value specified by the `cmdChar` parameter (you can specify 0 if the item doesn't have a keyboard equivalent, submenu, text encoding, reduced icon, or small icon). If you specify that the item has a submenu, you should provide the menu ID of the submenu as the item's marking character. If you specify that the item has a special text encoding, you must provide the text encoding in the icon field of the menu item. If you specify that the item has an `'SICN'` or a reduced `'ICON'` resource, you must provide the icon number in the icon field of the item.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Menus.h`

SetItemIcon

Sets a menu item's icon or text encoding. (Deprecated in Mac OS X v10.5.)

```
void SetItemIcon (
   MenuRef theMenu,
   MenuItemIndex item,
   short iconIndex
);
```

Parameters

theMenu

The menu containing the item.

item

The menu index of the item.

iconIndex

An integer representing the icon number or text encoding for the specified menu item. If the menu item's keyboard equivalent field does not contain 0x1C, the `SetItemIcon` function sets the icon number of the item's icon to the number defined in this parameter. The icon number you specify should be a value from 1 through 255 (or from 1 through 254 if the item has a small or reduced icon) or 0 if the item does not have an icon.

The Menu Manager adds 256 to the icon number to generate the resource ID of the '`cicn`' or '`ICON`' resource that describes the icon of the menu item. For example, if you specify 5 as the value of the `iconIndex` parameter, when the Menu Manager needs to draw the item, it looks for an icon resource with resource ID 261.

If the menu item's keyboard equivalent field contains 0x1C, the `SetItemIcon` function sets the text encoding of the menu item to the number defined in the `iconIndex` parameter. The Menu Manager displays the menu item using the specified text encoding if the corresponding script system is installed.

You can specify 0 in the `iconIndex` parameter to indicate that the item uses the current system script and does not have an icon number.

Discussion

In most cases, you should use `SetMenuItemTextEncoding` (page 1343) rather than `SetItemIcon` to set the menu item's text encoding.

The `SetItemIcon` function sets the icon number (for resource-based icons) or text encoding of the specified menu item to the value in the `iconIndex` parameter. To use handle-based icons, call `SetMenuItemIconHandle` (page 1338). Usually you display menu items in the current system script; however, if needed, you can use the `SetItemIcon` function to set the text encoding of a menu item. For an item's text encoding to be set, the keyboard equivalent field of the item must contain 0x1C. If the keyboard equivalent field contains any other value, the `SetItemIcon` function interprets the specified number as the item's icon number.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Menus.h`

SetItemMark

Sets the mark of a menu item.

```
void SetItemMark (
   MenuRef theMenu,
   MenuItemIndex item,
   CharParameter markChar
);
```

Parameters

theMenu

The menu containing the item.

item

The menu index of the item.

markChar

The mark of the menu item or its submenu (if the item has a submenu). To set the submenu associated with this menu item, specify the menu ID of the submenu in the `markChar` parameter. You can pass the character marking constants defined in the Font Manager. Pass 0 (`noMark`)if the menu item has neither mark nor submenu.

Discussion

You should call `SetMenuItemHierarchicalID` (page 1336) instead of `SetItemMark` to set the menu ID of a menu item's submenu. However, you can still use `SetItemMark` to set the mark of a menu item.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

QTCarbonShell

Declared In

`Menus.h`

SetItemStyle

Sets a menu item's text style.

```
void SetItemStyle (
   MenuRef theMenu,
   MenuItemIndex item,
   StyleParameter chStyle
);
```

Parameters

theMenu

The menu containing the item.

item

The menu index of the item.

chStyle

An integer representing the menu item's text style. You can choose from the following constants: `normal`, `bold`, `italic`, `underline`, `outline`, `shadow`, `condense`, and `extend`.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
Menus.h

SetMCEntries

Sets entries in an application's menu color information table. (Deprecated in Mac OS X v10.5.)

Not recommended

```
void SetMCEntries (
   short numEntries,
   MCTablePtr menuCEntries
);
```

Parameters

numEntries
The number of entries contained in the array of menu color entry structures.

menuCEntries
A pointer to an array of menu color entry structures. Specify the number of structures in the array in the numEntries parameter.

Discussion
The SetMCEntries function sets any specified menu bar entry, menu title entry, or menu item entry according to the values specified in the menu color entry structures. If an entry already exists for a specified menu color entry, the SetMCEntries function updates the entry in your application's menu color information table with the new values. If the entry doesn't exist, it is added to your application's menu color information table.

If any of the added entries specify a new menu bar color or new menu title colors, your application should call DrawMenuBar (page 1255) to update the menu bar with the new colors.

Special Considerations

The SetMCEntries function may move or purge memory. Your application should make sure that the array specified by the menuCEntries parameter is nonrelocatable before calling SetMCEntries.

Carbon Porting Notes

SetMCEntries does nothing, because Appearance Manager doesn't use color tables.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
Menus.h

SetMCInfo

Makes a copy of your application's menu color information table. (Deprecated in Mac OS X v10.5.)

Not recommended

```
void SetMCInfo (
   MCTableHandle menuCTbl
);
```

Parameters

menuCTbl

A handle to a copy of your application's menu color information table.

Discussion

The `SetMCInfo` function copies the table specified by the `menuCTbl` parameter to your application's menu color information table. If successful, the `SetMCInfo` function is responsible for disposing of your application's current menu color information table, so your application does not need to explicitly dispose of the current table.

Your application should call the Memory Manager function `MemError` to determine whether the `SetMCInfo` function successfully copied the table. If the `SetMCInfo` function cannot successfully copy the table, it does not dispose of the current menu color information table and the `MemError` function returns a nonzero result code. If the `SetMCInfo` function is able to successfully copy the table, it disposes of the current menu color information table and the `MemError` function returns the `noErr` result code.

If the menu color information table specifies a new menu bar color or new menu title colors, your application should call `DrawMenuBar` (page 1255) after calling `SetMCInfo`.

Note that `GetNewMBar` (page 1290) does not save your application's current menu color information table. If your application changes menu bars, you can save and restore your application's current menu color information table by calling `GetMCInfo` (page 1266) before `GetNewMBar` and calling `SetMCInfo` afterward.

Carbon Porting Notes

`SetMCInfo` does nothing, because Appearance Manager doesn't use color tables.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Menus.h`

SetMenuBar

Sets the current menu list to a specified menu list.

```
void SetMenuBar (
   MenuBarHandle mbar
);
```

Parameters

mbar

A handle to a menu list that specifies the menus for a menu bar. You should specify a handle returned by `GetMenuBar` (page 1268) or `GetNewMBar` (page 1290).

Discussion
The `SetMenuBar` function copies the given menu list to the current menu list. As with `GetMenuBar` (page 1268) , `SetMenuBar` doesn't copy the menu structures, just the menu list (which contains handles to the menu structures).

You can use `SetMenuBar` to restore a menu list that you previously saved using `GetMenuBar` or to set the current menu list to a menu list created from a nib file or from `GetNewMBar` (page 1290).

The `SetMenuBar` function sets the current menu list and calls `InvalMenuBar` (page 1301) so the menu bar can be updated the next time through the event loop; if you want to redraw the menu bar immediately, call the `DrawMenuBar` (page 1255) function.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
BSDLLCTest

Declared In
`Menus.h`

SetMenuCommandMark

Locates the menu item with a specified command ID and sets its mark character.

```
OSStatus SetMenuCommandMark (
   MenuRef inMenu,
   MenuCommand inCommandID,
   UniChar inMark
);
```

Parameters

inMenu
The menu in which to begin searching for the item. Pass `NULL` to begin searching with the root menu. The search will descend into all submenus of this menu.

inCommandID
The command ID of the menu item to be modified. If more than one item has this command ID, only the first will be modified.

inMark
The new mark character. While this is a Unicode character, only the low byte is currently used as the mark character, and it is interpreted using the application text encoding.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
If you have access to the menu item index, in most cases you should use `SetItemMark` (page 1324) instead, as that function is faster and requires no searching.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
Menus.h

SetMenuCommandProperty

Sets the property data for a menu item with the specified command ID.

```
OSStatus SetMenuCommandProperty (
   MenuRef inMenu,
   MenuCommand inCommandID,
   OSType inPropertyCreator,
   OSType inPropertyTag,
   ByteCount inPropertySize,
   const void *inPropertyData
);
```

Parameters

inMenu

inCommandID

The command ID of the menu item whose property you want to set. If more than one item has the same command ID, only the first item's property is set.

inPropertyCreator

The four-character creator code for the application.

inPropertyTag

The four-character tag for the property you want to set.

inPropertySize

The side of the property data, in bytes.

inPropertyData

The property data to set.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
If you have access to the menu item index, in most cases you should use `SetMenuItemProperty` (page 1341) instead, as that function is faster and requires no searching.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
Menus.h

SetMenuDefinition

Sets the menu definition structure for a menu.

```
OSStatus SetMenuDefinition (
   MenuRef menu,
   const MenuDefSpec *defSpec
);
```

Parameters

menu

The menu whose menu definition structure you want to set.

defSpec

The new menu definition structure.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
You can use this function to change your menu definition on-the-fly.

Availability
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
`Menus.h`

SetMenuExcludesMarkColumn

Sets whether a menu contains space for mark characters.

Not recommended

```
OSStatus SetMenuExcludesMarkColumn (
   MenuRef menu,
   Boolean excludesMark
);
```

Parameters

menu

The menu whose width is to be set.

excludesMark

Pass `true` to specify that the menu be drawn without space for mark characters; `false` to specify that the menu be drawn in its full width, with space for mark characters.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
Your application may use the `SetMenuExcludesMarkColumn` function to set the width of an individual menu, so that no space is provided for mark characters such as checkmarks, dashes, or notification symbols (diamonds).

The `SetMenuExcludesMarkColumn` function is only recommended for use with pop-up menus, and then only in special cases. Mac OS human interface guidelines require that all standard (menu bar) menus include space for mark characters, and pop-up menus that present user-selectable attributes or commands should

also contain space for marks. If a pop-up menu does not present a list of user-selectable attributes or commands, as is the case with the Mac OS 8.5 Window Manager window proxy pop-up menus that display a standard file system path, then narrowing the menu to exclude space for marks may be appropriate.

See also the `GetMenuExcludesMarkColumn` (page 1272) function.

Carbon Porting Notes

You should instead set the `kMenuExcludesMarkColumn` menu attribute using the `ChangeMenuAttributes` (page 1235) function.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Menus.h`

SetMenuFlashCount

Specifies whether a menu should fade slowly or immediately disappear when closing. (Deprecated in Mac OS X v10.5.)

Modified

```
void SetMenuFlashCount (
   short count
);
```

Parameters

count

See the Discussion.

Discussion

Unlike the classic Mac OS version, `SetMenuFlashCount` no longer lets you set the flash count for a menu. In Mac OS X, the flash count is always 1, and it not user adjustable.

Passing zero for the `count` parameter causes menus to disappear immediately when closing; passing any nonzero parameter causes the menus to fade out when closing (the default).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Menus.h`

SetMenuFont

Sets the font to be used in a menu.

```
OSStatus SetMenuFont (
   MenuRef menu,
   SInt16 inFontID,
   UInt16 inFontSize
);
```

Parameters

menu

The menu whose font is to be set.

inFontID

The font family ID for the font to be used. Pass 0 to use the current system font. Note that this is the font to be used in the menu items, not the menu title. You cannot change the menu title font.

inFontSize

The size, in points, of the font to be used. Valid font size values range from 9 to 24 points, inclusive. Pass 0 to use the font size of the current system font.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
Your application may use the `SetMenuFont` function to set the font for an individual menu, such as a pop-up menu. This function sets the font used by all the menu items in the menu. If you want to set the font of only a particular menu item, use the `SetMenuItemFontID` (page 1335) function instead.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

SetMenuHeight

Set the height of a menu.

```
void SetMenuHeight (
   MenuRef menu,
   SInt16 height
);
```

Parameters

menu

The menu whose height you want to set.

height

The height of the menu, in pixels.

Discussion
Calling `CalcMenuSize` (page 1234) on the menu overwrites the menu height you set with this function.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
Menus.h

SetMenuID

Assigns a menu ID to a menu.

```
void SetMenuID (
   MenuRef menu,
   MenuID menuID
);
```

Parameters

menu
The menu you want to assign an ID to.

menuID
The menu ID to assign.

Discussion
In most cases you assign an ID to a menu when you first create it, but you can use this function to change it later.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
Menus.h

SetMenuItemCommandID

Sets a menu item's command ID.

```
OSErr SetMenuItemCommandID (
   MenuRef inMenu,
   MenuItemIndex inItem,
   MenuCommand inCommandID
);
```

Parameters

inMenu
The menu that contains the menu item whose command ID you want to set.

inItem
The menu index of the item.

inCommandID
An integer representing the command ID that you wish to set.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
You can use a menu item's command ID as a position-independent method of signalling a specific action in an application. See *Carbon Event Manager Programming Guide* for more information about command IDs.

Note that Apple reserves all command IDs that contain all lowercase letters; your application is free to use any command ID containing uppercase characters.

See also the function `GetMenuItemCommandID` (page 1275).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

SetMenuItemCommandKey

Sets the keyboard equivalent of a menu item.

```
OSStatus SetMenuItemCommandKey (
   MenuRef inMenu,
   MenuItemIndex inItem,
   Boolean inSetVirtualKey,
   UInt16 inKey
);
```

Parameters

inMenu
The menu containing the item.

inItem
The item whose keyboard equivalent you want to set.

inSetVirtualKey
Indicates whether to set the item's character code (`false`) or virtual keycode equivalent. (`true`).

inKey
The character code or virtual keycode equivalent to set.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
A menu item's keyboard equivalent may be either a character code or a virtual keycode. The character code is always used to draw the item's keyboard equivalent in the menu, but either may be used for keyboard equivalent matching by `MenuEvent` and `IsMenuKeyEvent`, depending on whether the `kMenuItemAttrUseVirtualKey` item attribute is set. If `SetMenuItemCommandKey` is used to set the virtual keycode equivalent for a menu item, it also automatically sets the `kMenuItemAttrUseVirtualKey` item attribute. To make the menu item stop using the virtual keycode equivalent and use the character code equivalent instead, use `ChangeMenuItemAttributes` to clear the `kMenuItemAttrUseVirtualKey` item attribute.

Version Notes
Prior to Mac OS X v10.3, passing a character code in the range 0x1A to 0x21 (the range of command key metacharacters such as `hMenuCmd`) returned an error. In Mac OS X v10.3 and later, the Menu Manager interprets codes in this range as the ASCII character for that value, and displays the appropriate command key glyph. For example, passing `hMenuCmd` would set the keyboard equivalent to be command-Escape, because the value of `hMenuCmd` (0x1B) is the character code for the Escape character.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

SetMenuItemData

Sets multiple attributes of a menu item at once.

```
OSStatus SetMenuItemData (
   MenuRef inMenu,
   MenuItemID inItem,
   Boolean inIsCommandID,
   const MenuItemDataRec *inData
);
```

Parameters

inMenu
The menu whose attributes you want to set. Note that if you pass `true` for the `inIsCommandID` parameter, you can pass `NULL` here, in which case the Menu Manager searches the root menu for the first menu that matches the specified command ID.

inItem
The menu item index or the command ID of the menu item.

inIsCommandID
A Boolean value indicating whether the value passed for the `inItem` parameter is a command ID or a menu item index. Pass `true` to indicate a command ID, `false` to indicate that it is a menu item index. If you pass `true`, the Menu Manager sets the data for the first menu item that matches the specified command ID.

inData
A pointer to a `MenuItemDataRec` structure. Before calling, you should set the `whichData` field to indicate what data you want to set and fill out those fields appropriately. For more details on the types of data you can set, see "Menu Item Data Flags" (page 1390).

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
This function is often more efficient than calling individual accessor functions if you want to set multiple attributes simultaneously.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

SetMenuItemFontID

Sets the font for a menu item.

```
OSErr SetMenuItemFontID (
   MenuRef inMenu,
   MenuItemIndex inItem,
   SInt16 inFontID
);
```

Parameters

inMenu

The menu that contains the menu item for which you wish to set the font.

inItem

The menu index of the item.

inFontID

An integer representing the font ID that you wish to set.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
You can use this function to set up a font menu that displays item in its appropriate font. If you want to set the font for all the items in a menu, you can use the `SetMenuFont` (page 1331) function.

See also the function `GetMenuItemFontID` (page 1277).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

SetMenuItemHierarchicalID

Attaches a submenu to a menu item. (Deprecated in Mac OS X v10.5.)

```
OSErr SetMenuItemHierarchicalID (
   MenuRef inMenu,
   MenuItemIndex inItem,
   MenuID inHierID
);
```

Parameters

inMenu

The menu that contains the menu item to which you wish to attach a submenu.

inItem

The menu index of the item.

inHierID

An integer representing the menu ID of the submenu you wish to attach. This menu should be inserted into the menu list by calling `InsertMenu` with the `kInsertHierarchicalMenu` constant.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
The `SetMenuItemHierarchicalID` function should be called instead of setting the keyboard equivalent to 0x1B. You should call `SetMenuItemHierarchicalID` instead of `SetItemMark` to set the menu ID of a menu item's submenu. However, you can still use `SetItemMark` to set the mark of a menu item.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Menus.h`

SetMenuItemHierarchicalMenu

Attaches a submenu to a menu item.

```
OSStatus SetMenuItemHierarchicalMenu (
   MenuRef inMenu,
   MenuItemIndex inItem,
   MenuRef inHierMenu
);
```

Parameters
inMenu
The parent menu.
inItem
The parent item.
inHierMenu
The submenu to attach. Pass `NULL` to remove an existing submenu.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
Using `SetMenuItemHierarchicalMenu`, it is possible to directly specify the submenu for a menu item without specifying its menu ID. It is not necessary to insert the submenu into the hierarchical portion of the menubar, and it is not necessary for the submenu to have a unique menu ID. Simply use 0 as the menu ID for the submenu, and identify selections from the menu by command ID.

The Menu Manager will increment the reference count of the submenu that you specify, and the submenu's reference count will be decremented automatically when the parent menu item is deleted or the parent menu is disposed.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

SetMenuItemIconHandle

Sets a menu item's icon.

```
OSErr SetMenuItemIconHandle (
   MenuRef inMenu,
   MenuItemIndex inItem,
   UInt8 inIconType,
   Handle inIconHandle
);
```

Parameters

inMenu

The menu that contains the menu item for which you wish to set an icon.

inItem

The menu index of the item.

inIconType

Pass a value representing the type of icon (`'ICON'`, `'cicn'`, `'SICN'`, icon suite, or `IconRef`) you wish to attach; see "Menu Item Icon Type Constants" (page 1394) for descriptions of possible values.

inIconHandle

Pass a handle to the icon you wish to attach to a menu item.

Return Value

A result code. See "Menu Manager Result Codes" (page 1399).

Discussion

The `SetMenuItemIconHandle` function sets the icon of a menu item with an icon handle instead of a resource ID. `SetMenuItemIconHandle` allows you to set icons of type `'ICON'`, `'cicn'`, `'SICN'`, `IconRef`, `CGImageRef`, as well as icon suites. To set resource-based icons for a menu item, call `SetItemIcon`.

With the exception of types `IconRef` and `CGImageRef`,disposing of the menu will not dispose of the icon handles set by this function. The Menu Manager retains `IconRef`, `CGImageRef` icons and releases them when the menu is disposed or the menu item is removed. For all other icon types, your application should dispose of the icons when you dispose of the menu, to prevent memory leaks.

See also the function `GetMenuItemIconHandle` (page 1279).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

QTCarbonShell

Declared In

`Menus.h`

SetMenuItemIndent

Sets the indent level of a menu item.

```
OSStatus SetMenuItemIndent (
   MenuRef inMenu,
   MenuItemIndex inItem,
   UInt32 inIndent
);
```

Parameters

inMenu

The menu containing the item.

inItem

The item whose indent level you want to set.

inIndent

The new indent level of the item.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
The indent level of an item is an amount of extra space added to the left of the item's icon or checkmark. The level is simply a number, starting at zero, which the Menu Manager multiplies by a constant to get the indent in pixels. The default indent level is zero.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

SetMenuItemKeyGlyph

Sets the command key glyph code for a menu item.

```
OSErr SetMenuItemKeyGlyph (
   MenuRef inMenu,
   MenuItemIndex inItem,
   SInt16 inGlyph
);
```

Parameters

inMenu

The menu that contains the menu item for which you wish to substitute a keyboard glyph.

inItem

The menu index of the item.

inGlyph

An integer representing the substitute glyph to display. Pass 0 to remove an existing glyph code. For a description of available keyboard glyphs, see "Menu Glyph Constants" (page 1382).

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
The `SetMenuItemKeyGlyph` function overrides the character that would normally be displayed in a menu item's keyboard equivalent with a substitute keyboard glyph. This is useful if the keyboard glyph in the font doesn't match the actual character generated. For example, you might use this function to display function keys.

In addition, the glyph code you specify is used for command key matching if the menu item does not already have a command key or virtual keycode assigned to it.

See also the function `GetMenuItemKeyGlyph` (page 1280).

Version Notes
In CarbonLib 1.2 and later and Mac OS X v10.0 and later, the Menu Manager automatically draws the appropriate glyph for a menu item that has a virtual keycode assigned to it; you do not have to set both virtual keycode and the glyph.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

SetMenuItemModifiers

Sets the modifier key(s) that must be pressed with a character key to select a particular menu item.

```
OSErr SetMenuItemModifiers (
   MenuRef inMenu,
   MenuItemIndex inItem,
   UInt8 inModifiers
);
```

Parameters

inMenu
The menu that contains the menu item for which you wish to set the modifier key(s).

inItem
The menu index of the item.

inModifiers
A value representing the modifier key(s) to be used in selecting the menu item; see "Modifier Key Mask Constants" (page 1396).

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
You can call the `SetMenuItemModifiers` function to change the modifier key(s) you can include with a character key to create your keyboard equivalent. For example, you can change Command-x to Command-Option-Shift-x. By default, the Command key is always specified; you can remove it by calling `SetMenuItemModifiers` with the `kMenuNoCommandModifier` mask constant, or (if you are using a nib file) by unchecking the appropriate command checkbox in the Interface Builder menu item inspector.

See also the function `GetMenuItemModifiers` (page 1281).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

SetMenuItemProperty

Associates data with a menu item.

```
OSStatus SetMenuItemProperty (
   MenuRef menu,
   MenuItemIndex item,
   OSType propertyCreator,
   OSType propertyTag,
   ByteCount propertySize,
   const void *propertyData
);
```

Parameters

menu
The menu containing the item with which you wish to associate data.

item
The index number of the menu item or 0 if the data is to be associated with the menu as a whole.

propertyCreator
A four-character code. Pass your program's signature, as registered through Apple Developer Technical Support. If your program is of a type that would not normally have a signature (for example, a plug-in), you should still register and use a signature in this case, even though your program's file may not have the same creator code as the signature that you register. The `'macs'` property signature is reserved for the system and may not be used.

propertyTag
A four-character code that identifies the property to set. You define the tag your application uses to identify the data; this code is not to be confused with the file type for the data, but may coincide if you wish.

propertySize
The size of the data.

propertyData
A pointer to the data.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
You may use the `SetMenuItemProperty` function to associate arbitrary data, tagged with an identifying code, with a menu item.

See also the `GetMenuItemProperty` (page 1281) and `RemoveMenuItemProperty` (page 1320) functions.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
HID Explorer
QTCarbonShell

Declared In
`Menus.h`

SetMenuItemRefCon

Sets application-specific information for a menu item.

```
OSErr SetMenuItemRefCon (
   MenuRef inMenu,
   MenuItemIndex inItem,
   URefCon inRefCon
);
```

Parameters

inMenu
The menu that contains the menu item with which you wish to associate information.

inItem
The menu index of the item. In CarbonLib 1.6 and later and Mac OS X v10.2 and later, you may pass zero to set a reference constant for the menu itself.

inRefCon
An unsigned 32-bit integer value. Pass a reference constant to associate with the menu item.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
If you have any data you want to associate with a menu item, you can do so using the `SetMenuItemRefCon` function. Note that you can also use `SetMenuItemProperty` (page 1341), which allows more flexibility in specifying application-specific data.

See also the function `GetMenuItemRefCon` (page 1284).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

SetMenuItemText

Sets menu item text to a specified string. (Deprecated in Mac OS X v10.5.)

Not recommended

```
void SetMenuItemText (
   MenuRef theMenu,
   MenuItemIndex item,
   ConstStr255Param itemString
);
```

Parameters

theMenu

The menu containing the item.

item

The menu index of the item.

itemString

The menu item's text string. This parameter must not be `NULL` or an empty (zero-length) string. Do not use meta–font characters in this parameter.

Your menu item text string must be limited to 250 bytes. 251 or more bytes will cause the Menu Manager to crash.

Discussion

The `SetMenuItemText` function does not recognize any metacharacters used by `AppendMenu` (page 1229) and `InsertMenuItem` (page 1295).

If you set the text of a menu item using the `SetMenuItemText` function, you should store the text in a string resource so that your application can be more easily localized.

Carbon Porting Notes

n Carbon, you should use the `SetMenuItemTextWithCFString` (page 1344) function instead.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Menus.h`

SetMenuItemTextEncoding

Sets the text encoding for a menu item's text. (Deprecated in Mac OS X v10.5.)

```
OSErr SetMenuItemTextEncoding (
   MenuRef inMenu,
   MenuItemIndex inItem,
   TextEncoding inScriptID
);
```

Parameters

inMenu

The menu containing the menu item whose text encoding you wish to set.

inItem

The menu index of the item.

inScriptID

The script code that corresponds to the text encoding you wish to set.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
If a menu item has a command code of 0x1C when `SetMenuItemTextEncoding` is called, the values in the menu item's command key and icon ID fields are cleared and replaced with the value in the `inScriptID` parameter of `SetMenuItemTextEncoding`.

See also the function `GetMenuItemTextEncoding` (page 1285).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Menus.h`

SetMenuItemTextWithCFString

Sets the text of a menu item to the text contained in a CFString.

```
OSStatus SetMenuItemTextWithCFString (
   MenuRef inMenu,
   MenuItemIndex inItem,
   CFStringRef inString
);
```

Parameters

inMenu
The menu containing the item.

inItem
The item whose text you want to set.

inString
The CFString containing the new menu item text.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
If the CFString is mutable, the Menu Manager will make its own copy of the CFString before returning from `SetMenuItemTextWithCFString`. Modifying the string after calling `SetMenuItemTextWithCFString` will have no effect on the item's actual text.

If the CFString is immutable, the Menu Manager increments the reference count of the string before returning.

The caller may release the string after calling `SetMenuItemTextWithCFString`.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
BSDLLCTest

Declared In
Menus.h

SetMenuTitle

Sets the title of a menu. (Deprecated in Mac OS X v10.5.)

Not Recommended

```
OSStatus SetMenuTitle (
   MenuRef menu,
   ConstStr255Param title
);
```

Parameters

menu
The menu whose title you want to set.

title
A string containing the menu title to set.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
Unless you need to support legacy code, you should use the `SetMenuTitleWithCFString` (page 1346) function instead.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
Menus.h

SetMenuTitleIcon

Sets the title of a menu to be an icon.

```
OSStatus SetMenuTitleIcon (
   MenuRef inMenu,
   UInt32 inType,
   void *inIcon
);
```

Parameters

inMenu
The menu whose title to set.

inType
The type of icon being used to specify the icon title; use `kMenuNoIcon` to remove the icon from the menu title. The supported types are `kMenuIconSuiteType`, `kMenuIconRefType`, and (in Mac OS X v10.3 and later) `kMenuIconCGImageRefType`.

inIcon

The icon; this parameter must be `NULL` if `inType` is `kMenuNoIcon`. The supported icon formats are `IconSuiteRef`, `IconRef`, and (in Mac OS X v10.3 and later) `CGImageRef`.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
The Menu Manager takes ownership of the supplied icon after this call. When a menu with an title icon is disposed, the Menu Manager will dispose the icon as well. The Menu Manager will also dispose of the current title icon when a new text or icon title is supplied for a menu. If you specify an `IconRef` or `CGImageRef`, the Menu Manager will increment its reference count, so you can release your reference to the `IconRef` without invalidating the Menu Manager's copy. The menu bar is invalidated by this call and will be redrawn at the first opportunity.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

SetMenuTitleWithCFString

Sets the title of a menu to the text contained in a CFString.

```
OSStatus SetMenuTitleWithCFString (
   MenuRef inMenu,
   CFStringRef inString
);
```

Parameters

inMenu

The menu whose title you want to set.

inString

The string containing the new menu title text.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
If the string is mutable, the Menu Manager will make its own copy of the CFString before returning from `SetMenuTitleWithCFString`. Modifying the string after calling `SetMenuTitleWithCFString` will then have no effect on the menu's actual title.

If the string is immutable, the Menu Manager simply increments the string's reference count.

The caller may release the string after calling `SetMenuTitleWithCFString`.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

SetMenuWidth

Sets the width of a menu.

```
void SetMenuWidth (
   MenuRef menu,
   SInt16 width
);
```

Parameters

menu

The menu whose width you want to set.

width

The width of the menu, in pixels.

Discussion

Calling `CalcMenuSize` (page 1234) on the menu overwrites the menu width you set with this function.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Menus.h`

SetRootMenu

Sets the menu whose contents are displayed in the menubar.

```
OSStatus SetRootMenu (
   MenuRef inMenu
);
```

Parameters

inMenu

The new root menu.

Return Value

A result code. See "Menu Manager Result Codes" (page 1399).

Discussion

The root menu should contain one menu item for each top-level menu to be displayed in the menu bar. Each menu item in turn should have a submenu that was specified using `SetMenuItemHierarchicalMenu` (page 1337).

You can use the `SetRootMenu` function along with the `AcquireRootMenu` (page 1229) function to save and restore the contents of the menu bar. Note that calling `SetRootMenu` sets the contents of the hierarchical portion of the menu list as well as the top-level menus displayed in the menu bar. Before returning the root menu, `AcquireRootMenu` calls `InsertMenu` (page 1294) with the `kInsertHierarchialMenu` option to attach to the root menu a list of the menus that are currently inserted into the hierarchical portion of the menu. `SetRootMenu` reinserts any attached hierarchical menus into the hierarchical portion of the menu list. If you pass a newly-created menu to `SetRootMenu`, the hierarchical menu list is cleared and is empty.

Calling `SetRootMenu` also implicitly retains the new root menu, and you should release it at the appropriate time by calling `ReleaseMenu` (page 1319).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

ShowMenuBar

Displays the menu bar.

```
void ShowMenuBar (
   void
);
```

Discussion
The `ShowMenuBar` function makes the menu bar visible and selectable by the user.

Note that calling this function also causes a `kEventMenuBarShown` event to be sent to the application target (if your application has registered for the event).

See also the `HideMenuBar` (page 1291) and `IsMenuBarVisible` (page 1303) functions.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

UpdateInvalidMenuItems

Redraws the invalid items of an open menu.

```
OSStatus UpdateInvalidMenuItems (
   MenuRef inMenu
);
```

Parameters
inMenu
The menu whose menu items you want to update.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
This function redraws menu items in an open menu that has either had some visible aspect of its menu items change (such as their enable state) or has been invalidated by a call to `InvalidateMenuItems` (page 1299).

It is not necessary to use `UpdateInvalidMenuItems` if you are using Carbon's built-in support for dynamic items based on modifier key state. However, if you are modifying items dynamically using your own implementation, you should call `UpdateInvalidMenuItems` after completing your modifications for a single menu. It will redraw any items that have been marked as invalid, and clear the invalid flag for those items.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

UpdateStandardFontMenu

Updates a standard Font menu.

```
OSStatus UpdateStandardFontMenu (
   MenuRef menu,
   ItemCount *outHierMenuCount
);
```

Parameters

menu
The menu you want to update.

outHierMenuCount
The number of hierarchical menus attached to the standard Font menu. This value may be `NULL` if the hierarchical menus count is not useful. For example, if the only submenus in your application are those created by `CreateStandardFontMenu`, then you don't need to worry about the hierarchical menu count, as any existing submenu must be a font menu.

Return Value
A result code. See "Menu Manager Result Codes" (page 1399).

Discussion
`UpdateStandardFontMenu` calls the Font Manager function `FMGetFontGeneration` to determine if the fonts have changed and returns immediately without modifying the font menu if no changes occurred. As there is little overhead if no fonts have changed, you can call this function whenever you think it may be useful; for example, when your application is activated, or whenever you receive a `kEventMenuPopulate` event for the font menu.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

Callbacks

MenuDefProcPtr

Defines a pointer to a custom menu definition callback function. Your menu definition callback function draws the menu items in the menu, determines which item the user chose from the menu, and calculates the menu's dimensions.

Not recommended

```
typedef void (*MenuDefProcPtr) (
    SInt16 message,
    MenuRef theMenu,
    Rect *menuRect,
    Point hitPt,
    SInt16 *whichItem
);
```

If you name your function `MyMenuDefProc`, you would declare it like this:

```
void *MyMenuDefProc (
    SInt16 message,
    MenuRef theMenu,
    Rect *menuRect,
    Point hitPt,
    SInt16 *whichItem
);
```

Parameters

message

A constant that identifies the operation the menu definition function should perform; see "Custom Menu Definition Message Constants" (page 1372) and "Obsolete Menu Definition Messages" (page 1374) for a description of the messages that your menu definition function can receive. Your menu definition function should not respond to any value other than these defined messages. Other messages are reserved for internal use by Apple Computer, Inc.

theMenu

The menu that the operation should affect. This menu reference may refer to a regular menu or a popup menu.

menuRect

A pointer to the rectangle (in global coordinates) in which the menu is located.

hitPt

A mouse location (in global coordinates). The Menu Manager provides information in this parameter to the menu definition function when the `kMenuFindItemsMsg` or `kMenuPopUpMsg` messages are sent.

whichItem

A pointer to a value that specifies the item number of the menu item to act upon, or a pointer to a data structure related to one or more menu items. The data pointed to depends on the message sent; see the discussion for specifics.

Discussion

Carbon does not let you store menu definitions as resources. If you need to upgrade legacy code, see the *Carbon Porting Guide* for information about converting your resource-based definitions. See the Carbon Porting Notes for this callback for additional information about changes to older messages and access to low-memory global data.

On Mac OS X v10.3 and later, all standard menus are implemented as HIViews, and custom menus can be implemnented as custom HIViews. If you want to create custom menu definitions, you should subclass the HIMenuView class. See Introducing HIView in Carbon User Experience documentation for more information.

When you define your menus, you specify the menu definition function the Menu Manager should use when managing them. You'll usually want to use the standard menu definition function for your application. However, if you need a feature not provided by the standard menu definition function (for example, if you want to include more graphics in your menus), you can choose to write your own menu definition function.

The menu definition function is responsible for drawing the contents of the menu and its menu items, determining whether the cursor is in a displayed menu, highlighting and unhighlighting menu items, and calculating a menu's dimensions. To create an instance of your custom menu, call the `CreateCustomMenu` (page 1245) function.

The Menu Manager calls your menu definition function whenever it needs your definition function to perform a certain action on a specific menu. The action your menu definition function should perform depends on the value of the `message` parameter. When the Menu Manager requests your menu definition function to perform an action on a menu, it provides your function with the appropriate menu reference. This allows your function to access the data in the menu structure and to use any data in the variable data portion of the menu structure to appropriately handle the menu items.

Your menu definition function should support the following messages:

- `kMenuInitMsg`

  The Menu Manager sends this message when creating a menu to give your menu definition a chance to perform any required initialization. If an error occurs during initialization, your menu definition should return a nonzero error code in the `*whichItem` parameter. This error is then returned by the function used to create the menu.

- `kMenuDisposeMsg`

  Sent when a menu is destroyed. The Menu Manager sends this message to give your menu definition a chance to release or dispose of any related data.

- `kMenuFindItemMsg`

  Sent when the Menu Manager is displaying a menu and needs to know what item is under the mouse.

  The `whichItem` parameter points to a `MenuTrackingData` structure. On entry, the `menu`, `virtualMenuTop`, and `virtualMenuBottom` fields of this structure are valid. Your menu definition should determine which item, if any, contains the point passed to you in the `hitPt` parameter and fill in the `itemUnderMouse`, `itemSelected`, and `itemRect` fields. If an item is found, the menu definition should always fill in the `itemUnderMouse` and `itemRect` fields. The menu definition should only fill in the `itemSelected` field if the item is available for selection; if it is unavailable (because it is disabled, or for some other reason), the menu definition should set the `itemSelected` field to zero.

  The index values placed in the `itemUnderMouse` and `itemSelected` fields should be less than or equal to the number of items returned by `CountMenuItems` (page 1244) on this menu. These values should also be identical if both are nonzero. The `itemUnderMouse` field should always be nonzero if the mouse is actually over an item.

  The menu definition should not highlight the found item in response to this message, as the Menu Manager will send a separate `kMenuHiliteItemMsg` to request highlighting of the item.

  If the menu definition supports scrolling, it should scroll the menu during this message, and update the `virtualMenuTop` and `virtualMenuBottom` fields of the MenuTrackingData to indicate the menu's new scrolled position.

  If the menu definition uses QuickDraw to draw while scrolling, it should draw into the current port.

  If the menu definition uses CoreGraphics to draw while scrolling, it should use the `CGContextRef` passed in the context field of the `MDEFHiliteItemData` structure. Menu definitions must use the `ScrollMenuImage` (page 1322) function, if available, to scroll the menu contents. (This function is available in CarbonLib 1.5 and later, and in Mac OS X 10.1 and later.) `ScrollMenuImage` properly supports scrolling the alpha channel in the menu's image data. Use of the QuickDraw function `ScrollRect` to scroll the menu contents will result in the alpha channel being set to 0xFF (opaque) and the menu will no longer be translucent.

The menu definition should not modify the `menu` field of the `MenuTrackingData` structure.

- `kMenuHiliteItemMsg`

  Sent when the Menu Manager is displaying a menu and needs to highlight a newly selected item.

  The `whichItem` parameter points to a `MDEFHiliteItemData` structure. The menu definition should unhighlight the item in the `previousItem` field, if non-zero, and highlight the item in the `newItem` field.

  If the menu definition is using the Appearance Manager's menu drawing APIs, you should use the `EraseMenuBackground` (page 1258) function to erase the old menu contents before unhighlighting a menu item. This is necessary because the background of a menu is translucent in Aqua, and if the old highlight is not erased first, it will show through the new unhighlighted menu background.

  If the menu definition uses QuickDraw to draw, it should draw into the current port. If it uses CoreGraphics to draw, it should use the `CGContextRef` passed in the context field of the `MDEFHiliteItemData` structure.

- `kMenuDrawItemsMsg`

  Sent when the Menu Manager is displaying a menu and needs to redraw a portion of the menu. This message is used by the dynamic menu item support code in the Menu Manager; for example, if items five and six in a menu are a dynamic group, the Menu Manager will send a `kMenuDrawItemsMsg` message when the group's modifier key is pressed or released to redraw the appropriate item, but no other items in the menu.

  The whichItem parameter for this message points to an `MDEFDrawItemsData` structure. The menu definition should redraw the items starting with `firstItem` and ending with `lastItem`, inclusive.

  If the menu definition uses QuickDraw to draw, it should draw into the current port. If it uses CoreGraphics to draw, it should use the `CGContextRef` passed in the context field of the `MDEFHiliteItemData` structure.

- `kMenuDrawMsg`

  Sent when the Menu Manager is displaying a menu and needs to redraw the entire menu.

  The `whichItem` parameter is actually a pointer to a `MenuTrackingData` structure. On entry, the menu field of this structure is valid. The menu definition should draw the menu and, if it supports scrolling, should also fill in the `virtualMenuTop` and `virtualMenuBottom` fields of the structure to indicate the menu's initial unscrolled position; typically, `virtualMenuTop` would be set to the same value as the top coordinate of the menu bounds, and `virtualMenuBottom` would be set to `virtualMenuTop` plus the virtual height of the menu.

  If the menu definition uses QuickDraw to draw, it should draw into the current port. If it uses CoreGraphics to draw, it should use the `CGContextRef` passed in the context field of the `MDEFHiliteItemData` structure.

- `kMenuSizeMsg`

  Sent when the Menu Manager needs to determine the size of a menu.

  The menu definition should calculate the width and height of the menu and store the sizes into the menu with `SetMenuWidth` (page 1347) and `SetMenuHeight` (page 1332). If the `gestaltMenuMgrSendsMenuBoundsToDefProc` bit is set in the Menu Manager's Gestalt value, then the `hitPt` parameter sent with this message is the maximum width (`hitPt.h`) and height (`hitPt.v`) of the menu. The menu definition should ensure that the width and height that it places in the menu do not exceed these values. If the gestalt bit is not set, the menu definition should just use the main display device's (`GDevice`) width and height as constraints on the menu's width and height.

- `kMenuPopUpMsg`

  Sent when the Menu Manager is about to display a popup menu. The Menu Manager uses the menu definition function to support pop-up menus that are not implemented as controls. If your menu definition function supports pop-up menus, it should respond appropriately to the `kMenuPopUpMsg` message.

  The menu definition should calculate the appropriate menu bounds to contain the menu based on the requested menu location and selected item. It should write the menu's bounds into the `Rect` structure passed by the `menuRect` parameter. If the `gestaltMenuMgrSendsMenuBoundsToDefProc` bit is set in the Menu Manager's Gestalt value, then the `menuRect` parameter on entry to this message contains a constraint rectangle, in global coordinates, outside of which the popup menu should not be positioned. The menu definition should take these constraint bounds into account as it calculates the menu bounds. If the gestalt bit is not set, the menu definition should use the bounds of the display device (`GDevice`) containing the menu's top left corner as a constraint on the menu's position.

  The `hitPt` parameter is the requested location for the top left corner of the menu. The coordinates of this parameter are swapped from their normal order; `hitPt.h` contains the vertical coordinate and `hitPt.v` contains the horizontal coordinate.

  On entry, the `whichItem` parameter points at a menu item index which is requested to be the initial selection when the menu is displayed. After calculating the menu's bounds, the menu definition should write the menu's virtual top coordinate into the location pointed at by the `whichItem` parameter. If displaying the menu at the requested location does not require scrolling, the virtual top will be the same as the menu bounds top; if the menu must scroll to fit in the requested location, the virtual top may be different.

- `kMenuCalcItemMsg`

  Sent when the Menu Manager needs to know the bounds of a menu item.

  The menu definition should calculate the size of the menu item specified by the `whichItem` parameter and store the bounds in the `Rect` structure specified by the `menuRect` parameter. Some sample menu definition code provided by Apple has previously shown an implementation of this message that always sets the top left corner of the item bounds to (0,0), regardless of the item's actual position in the menu. For best future compatibility, menu definitions should begin storing an item bounds that gives the item's actual position in the menu based on the menu's current virtual top. For example, if the virtual menu top starts at 20, then the menu definition would calculate an item bounds for the first item that starts at (0,20), an item bounds for the second item that starts at (0,40), and so on. The menu definition should call `GetMenuTrackingData` (page 1288)to get the menu's current virtual position, and use zero for the menu top if `GetMenuTrackingData` returns an error.

- `kMenuThemeSavvyMsg`

  Sent by the Menu Manager to determine whether the menu definition uses the Appearance Manager menu-drawing functions to draw its menu. If it does, the menu definition should return `kThemeSavvyMenuResponse` in the location pointed to by `whichItem`. If the menu definition draws its own custom content without using the Appearance Manager menu-drawing functions, it should ignore this message.

The Menu Manager defines the data type `MenuDefUPP` to identify the universal procedure pointer for an application-defined menu definition function:

```
typedef UniversalProcPtr MenuDefUPP;
```

You typically use the `NewMenuDefProc` function like this:

```
MenuDefUPP myMenuDefProc;
```

```
myMenuDefProc = NewMenuDefProc(MyMenu);
```

Carbon Porting Notes

Prior to Carbon, menu definitions needed to use several low-memory globals to communicate with the Menu Manager. These globals have all been replaced or made obsolete in Carbon, as follows:

- `MenuDisable`

  `MenuDisable` is now set automatically by the Menu Manager using the value returned in the itemUnderMouse field of the MenuTrackingData structure passed to kMenuFindItemMsg.

- `TopMenuItem, AtMenuBottom`

  `TopMenuItem` and `AtMenuBottom` are now set automatically by the Menu Manager using the values returned in the `virtualMenuTop` and `virtualMenuBottom` fields of the `MenuTrackingData` structure passed to `kMenuDrawMsg` and `kMenuFindItemMsg`.

- `mbSaveLoc`

  This undocumented low-memory global was used by older menu definitions to store the bounding rect of the currently selected item and to avoid drawing glitches while the menu definition was scrolling the contents of a menu that had submenus. The Menu Manager now automatically sets the selected item bounds using the value returned in the itemRect field of the MenuTrackingData structure passed to kMenuFindItemMsg. In order to correctly support scrolling of menus with submenus, a menu definition should verify, before scrolling the menu contents, that no submenus of the scrolling menu are currently visible. A menu definition can use `GetMenuTrackingData` to verify this condition, as follows:

  ```
  Boolean SafeToScroll( MenuRef menuBeingScrolled )
   {
          MenuTrackingData lastMenuData;
          return GetMenuTrackingData( NULL, &lastMenuData ) ==  noErr
                              && lastMenuData.menu == menuBeingScrolled;
      }
  ```

  If `SafeToScroll` returns `false`, the menu definition should not scroll the menu.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

Data Types

HMenuBarHeader

Defines a list of hierarchical menus that have been inserted into a menu bar.

```
struct HMenuBarHeader {
    UInt16 lastHMenu;
    PixMapHandle menuTitleBits;
};
typedef struct HMenuBarHeader HMenuBarHeader;
```

Fields

`lastHMenu`
Offset from the start of the header to the last menu in the array of `HMenuBar` structures.

`menuTitleBits`
The saved bits behind the highlighted menu title. This value is undefined in menu bar handles returned by `GetNewMBar` (page 1290) or `GetMenuBar` (page 1268).

Discussion

The hierarchical portion of the menu bar follow the nonhierarchical portion in a menu bar handle. The hierarchical section consists of the `HMenuBarHeader` structure followed by an array of `HMenuBarMenu` (page 1355) structures.

You insert a hierarchical menu by specifying -1 for the `beforeID` parameter in `InsertMenu` (page 1294).

Availability

Available in Mac OS X v10.2 and later.

Declared In

`Menus.h`

HMenuBarMenu

Defines a hierarchical menu.

```
struct HMenuBarMenu {
    MenuRef menu;
    SInt16 reserved;
};
typedef struct HMenuBarMenu HMenuBarMenu;
```

Fields

`menu`
The menu.

`reserved`
Currently unused.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`Menus.h`

MCEntry

Specifies a menu color information table

Not recommended

```
struct MCEntry {
    MenuID mctID;
    short mctItem;
    RGBColor mctRGB1;
    RGBColor mctRGB2;
    RGBColor mctRGB3;
    RGBColor mctRGB4;
    short mctReserved;
};
typedef struct MCEntry MCEntry;
typedef MCEntry * MCEntryPtr;
typedef MCEntry MCTable[1];
```

Fields

`mctID`

Defines, along with the `mctItem` field, whether the entry is a menu bar entry, a menu title entry, or a menu item entry. The `mctID` field contains either a menu ID or 0 (for a menu bar).

`mctItem`

Defines, along with the `mctID` field, whether the entry is a menu bar entry, a menu title entry, or a menu item entry. The `mctItem` field contains either a menu item number or 0 (for a menu bar or menu title).

`mctRGB1`

Specifies color information for the entry, as follows. For a menu bar entry, this value is the default color for menu titles. For a menu title entry, this value is the title color of a specific menu. For a menu item entry, this value is the mark color for a specific item.

`mctRGB2`

Specifies color information for the entry, as follows. For a menu bar entry, this value is the default background color of a displayed menu. For a menu title entry, this value is the default color for the menu bar. For a menu item entry, this value is the color for the text of a specific item.

`mctRGB3`

Specifies color information for the entry, as follows. For a menu bar entry, this value is the default color of items in a displayed menu. For a menu title entry, this value is the default color for items in a specific menu. For a menu item entry, this value is the color for the modifier of a specific item.

`mctRGB4`

Specifies color information for the entry, as follows. For a menu bar entry, this value is the default color of the menu bar. For a menu title entry, this value is the background color of a specific menu. For a menu item entry, this value is the background color of a specific menu.

`mctReserved`

Reserved.

Discussion

The menu color information table defines the standard color for the menu bar, menu titles, menu items, and the background color of a displayed menu. If you do not add any menu color entries to this table, the Menu Manager draws your menus using the current default colors. Using the menu color information table to define custom colors for your menus is not recommended with Appearance Manager 1.0 and later.

When the Appearance Manager is available and you are using standard menus, if you do not include a menu bar entry in your menu color information table, only the menu title color and menu item text color values from menu color entries are used. If you do include a menu bar entry in your menu color information table, all menu colors are used, and the menus revert to a standard System 7 appearance.

If you are creating your own custom menu definition function, all entries in the table are used.

You can add custom colors to your menus by adding entries to your application's menu color information table, using Menu Manager functions or by defining these entries in an 'mctb' resource. Note that the menu color information table uses a format different from the standard color table format.

The value of the `mctID` field in the last entry in a menu color information table is `mctLastIDIndic`, and the rest of the fields of the last entry are reserved. The Menu Manager automatically creates the last entry in a menu color information table; your application should not use the value `mctLastIDIndic` as the menu ID of a menu if you wish to add a menu color entry for it.

The contents of a menu color table entry structure are interpreted differently, depending upon the values of the `mctID` and `mctItem` fields. Depending upon the value of these fields, the `MCEntry` structure represents a menu bar entry, a menu title entry, a menu item entry, or the last entry.

A *menu bar entry* is defined by a menu color entry structure that contains 0 in both the `mctID` and `mctItem` fields. You can define only one menu bar entry in a menu color information table. If you don't provide a menu bar entry for your application's menu color information table, the Menu Manager uses the standard menu bar colors (black text on a white background), and it uses the standard colors for the other menu elements. You can provide a menu bar entry to specify default colors for the menu title, the background of a displayed menu, the items in a menu, and the menu bar. The color information fields for a menu bar entry are interpreted as follows:

- `mctRGB1` specifies the default color for menu titles. If a menu doesn't have a menu title entry, the Menu Manager uses the value in this field as the color of the menu title.
- `mctRGB2` specifies the default color for the background of a displayed menu. If a menu doesn't have a menu title entry, the Menu Manager uses the value in this field as the color of the menu's background when it is displayed.
- `mctRGB3` specifies the default color for the items in a displayed menu. If a menu item doesn't have a menu item entry or a default color defined in a menu title entry, the Menu Manager uses the value in this field as the color of the menu item.
- `mctRGB4` specifies the default color for the menu bar. If a menu doesn't have a menu bar entry (and doesn't have any menu title entries), the Menu Manager uses the standard colors for the menu bar.

A *menu title entry* is defined by a menu color entry structure that contains a menu ID in the `mctID` field and 0 in the `mctItem` field. You can define only one menu title entry for each menu. If you don't provide a menu title entry for a menu in your application's menu color information table, the Menu Manager uses the colors defined by the menu bar entry. If a menu bar entry doesn't exist, the Menu Manager uses the standard colors (black on white). You can provide a menu title entry to specify a color for the title and background of a specific menu and a default color for its items. The color information fields for a menu title entry are interpreted as follows:

- `mctRGB1` specifies the color for the menu title of the specified menu. If a menu doesn't have a menu title entry, the Menu Manager uses the default value defined in the menu bar entry.
- `mctRGB2` specifies the default color for the menu bar. If a menu color information table doesn't have a menu bar entry, the Menu Manager uses the value in this field as the color of the menu bar. If a menu bar entry already exists, the Menu Manager replaces the value in the mctRGB2 field of the menu title entry with the value defined in the mctRGB4 field of the menu bar entry.
- `mctRGB3` specifies the default color for the items in the menu. If a menu item doesn't have a menu item entry or a default color defined in a menu bar entry, the Menu Manager uses the value in this field as the color of the menu item.
- `mctRGB4` specifies the color for the background of the menu.

A *menu item entry* is defined by a menu color entry structure that contains a menu ID in the `mctID` field and an item number in the `mctItem` field. You can define only one menu item entry for each menu item. If you don't provide a menu item entry for an item in your application's menu color information table, the Menu Manager uses the colors defined by the menu title entry (or by the menu bar entry if the menu containing the item doesn't have a menu title entry). If neither a menu title entry nor a menu bar entry exists, the Menu Manager draws the mark, text, and modifier in black. You can provide a menu item entry to specify a color for the mark, text, and keyboard equivalent of a specific menu item. The color information fields for a menu item entry are interpreted as follows:

- `mctRGB1` specifies the color for the mark of the menu item. If a menu item doesn't have a menu item entry, the Menu Manager uses the default value defined in the menu title entry or the menu bar entry.
- `mctRGB2` specifies the color for the text of the menu item. If a menu item doesn't have a menu item entry, the Menu Manager uses the default value defined in the menu title entry or the menu bar entry. The Menu Manager also draws a black-and-white icon of a menu item using the same color as defined by the `mctRGB2` field. (Use a `'cicn'` resource to provide a menu item with a color icon.)
- `mctRGB3` specifies the color for the modifier of the menu item. If a menu item doesn't have a menu item entry, the Menu Manager uses the default value defined in the menu title entry or the menu bar entry.
- `mctRGB4` specifies the color for the background of the menu. If the menu color information table doesn't have a menu title entry for the menu this item is in, or doesn't have a menu bar entry, the Menu Manager uses the value in this field as the background color of the menu. If a menu title entry already exists, the Menu Manager replaces the value in the `mctRGB4` field of the menu item entry with the value defined in the `mctRGB4` field of the menu title entry (or with the `mctRGB2` field of the menu bar entry).

You can use the `GetMCInfo` (page 1266) function to get a copy of your application's menu color information table and the `SetMCEntries` (page 1326) function to set entries of your application's menu color information table, or you can provide `'mctb'` resources that define the color entries for your menus.

The `GetMenu` (page 1266), `GetNewMBar` (page 1290), and `ClearMenuBar` (page 1237) functions can also modify the entries in the menu color information table. The `GetMenu` function looks for an `'mctb'` resource with a resource ID equal to the value in the `menuID` parameter. If it finds one, it adds the entries to the application's menu color information table.

The `GetNewMBar` function builds a new menu color information table when it creates the new menu list. If you want to save the current menu color information table, call `GetMCInfo` before calling `GetNewMBar`.

The `ClearMenuBar` function reinitializes both the current menu list and the menu color information table.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Menus.h`

MDEFDrawData

Contains information needed to draw a menu.

```
struct MDEFDrawData {
    MenuTrackingData trackingData;
    void * context;
};
typedef struct MDEFDrawData MDEFDrawData;
typedef MDEFDrawData * MDEFDrawDataPtr;
```

Fields

`trackingData`

A data structure containing information about the menu to be drawn. Your menu definition should fill in the `virtualMenuTop` and `virtualMenuBottom` fields of this structure while drawing the menu.

`context`

The Core Graphics context that your menu definition should draw into. The Menu Manager flushes the context after returning from the menu definition.

Discussion

The Menu Manager passes this structure to your custom menu definition in the `whichItem` parameter of the `kMenuDrawMsg` message.

Availability

Available in Mac OS X v10.1 and later.

Declared In

`Menus.h`

MDEFDrawItemsData

Contains information about which menu items to redraw.

```
struct MDEFDrawItemsData {
    MenuItemIndex firstItem;
    MenuItemIndex lastItem;
    MenuTrackingData * trackingData;
    void * context;
};
typedef struct MDEFDrawItemsData MDEFDrawItemsData;
typedef MDEFDrawItemsData * MDEFDrawItemsDataPtr;
```

Fields

`firstItem`

The first item to draw.

`lastItem`

The last item to draw.

`trackingData`

Information about the menu's tracking state. The `virtualMenuTop` and `virtualMenuBottom` fields in this structure will be the most useful in handling the `DrawItems` message.

`context`

The Core Graphics drawing context that your menu definition should draw into. The Menu Manager flushes the context after returning from the menu definition.

Discussion

The Menu Manager passes this structure to your custom menu definition in the `whichItem` parameter of the `kMenuDrawItemsMsg` message.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Menus.h`

MDEFFindItemData

Contains information used to determine which item the user has currently selected.

```
struct MDEFFindItemData {
    MenuTrackingData trackingData;
    void * context;
};
typedef struct MDEFFindItemData MDEFFindItemData;
typedef MDEFFindItemData * MDEFFindItemDataPtr;
```

Fields

`trackingData`
A data structure containing information about the menu to be drawn. Your menu definition should fill in the `itemSelected`, `ItemUnderMouse`and `itemRect` fields of this structure after determining which item is under the specified point.

`context`
The Core Graphics context the menu definition should draw into if it needs to scroll the menu during the `kMenuFindItemMsg` message. The Menu Manager flushes the context after the menu definition returns.

Discussion
The Menu Manager passes this structure to your custom menu definition in the `whichItem` parameter of the `kMenuFindItemsMsg` message.

Availability
Available in Mac OS X v10.1 and later.

Declared In
`Menus.h`

MDEFHiliteItemData

Contains information about which menu items should be highlighted and unhighlighted as the user moves through the menus. .

```
struct MDEFHiliteItemData {
    MenuItemIndex previousItem;
    MenuItemIndex newItem;
    void * context;
};
typedef struct MDEFHiliteItemData MDEFHiliteItemData;
typedef MDEFHiliteItemData * MDEFHiliteItemDataPtr;
typedef MDEFHiliteItemData HiliteMenuItemData;
typedef MDEFHiliteItemDataPtr HiliteMenuItemDataPtr;
```

Fields

`previousItem`

The menu item that was previously selected. This item needs to be redrawn in an unhighlighted state. This parameter can be zero if no item was previously selected.

`newItem`

The menu item that is now selected. This item needs to be redrawn in a highlighted state. This parameter can be zero if no item is currently highlighted.

`context`

The Core Graphics context the menu definition should draw into. The Menu Manager flushes the context after the menu definition returns.

Discussion

This structure is used by menu definition functions, which receive a pointer to an `MDEFHiliteItemData` structure as the `whichItem` parameter during the `kMenuHiliteItemMsg` message.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Menus.h`

MenuBarHandle

A handle to a menu bar header.

```
typedef Handle MenuBarHandle;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Menus.h`

MenuBarHeader

Defines a list of nonhierarchical menus that that have been placed in the menu bar.

```
struct MenuBarHeader {
    UInt16 lastMenu;
    SInt16 lastRight;
    SInt16 mbResID;
};
typedef struct MenuBarHeader MenuBarHeader;
```

Fields

`lastMenu`

Offset from the start of the header to the last menu in the array of `MenuBarMenu` (page 1362) structures, in bytes.

`lastRight`

The x-coordinate of the right edge of the rightmost menu, in global coordinates. This value is undefined in menu bar handles returned by `GetNewMBar` (page 1290) or `GetMenuBar` (page 1268).

`mbResID`

The MBDF resource ID. This value is undefined in menu bar handles returned by `GetNewMBar` (page 1290) or `GetMenuBar` (page 1268).

Discussion

This structure is contained within a menu bar handle (`MenuBarHandle`).

Availability

Available in Mac OS X v10.2 and later.

Declared In

`Menus.h`

MenuBarMenu

Defines a nonhierarchical menu.

```
struct MenuBarMenu {
    MenuRef menu;
    SInt16 menuLeft;
};
typedef struct MenuBarMenu MenuBarMenu;
```

Fields

`menu`

`menuLeft`

The x-coordinate of the left edge of the menu title, in global coordinates. This value is undefined in menu bar handles returned by `GetNewMBar` (page 1290) or `GetMenuBar` (page 1268).

Availability

Available in Mac OS X v10.2 and later.

Declared In

`Menus.h`

MenuCommand

Specifies a menu item's command ID.

```
typedef UInt32 MenuCommand;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
Menus.h

MenuCRsrc

Specifies a set of menu color table entries

Not recommended

```
struct MenuCRsrc {
    short numEntries;
    MCTable mcEntryRecs;
};
typedef struct MenuCRsrc MenuCRsrc;
typedef MenuCRsrc * MenuCRsrcPtr;
```

Fields
numEntries
mcEntryRecs

Availability
Available in Mac OS X v10.0 and later.

Declared In
Menus.h

MenuDefSpec

Defines the type of menu definition.

```
struct MenuDefSpec {
    MenuDefType defType
    union {
        MenuDefUPP defProc;
        struct {
                CFStringRef classID;
                EventRef initEvent;
                } view;
    } u;
};
typedef struct MenuDefSpec MenuDefSpec;
typedef MenuDefSpec * MenuDefSpecPtr;
```

Fields
defType
The type of menu definition. See "Menu Definition Type Constants" (page 1379) for a list of possible values.

`u.defproc`
: If the `defType` field is `kMenuDefProcPtr`, the menu definition is a older procedure pointer–based definition. This field then contains a UPP to the menu definition function.

`u.view.classID`
: If the `defType` field is `kMenuDefClassID`, the menu definition is HIView–based. This field then contains the ID of the HIView subclass that defines this menu.

`u.view.initEvent`
: If the `defType` field is `kMenuDefClassID`, the menu definition is HIView–based. This field then contains the initialization event for the HIView subclass if one exists. Otherwise, it is `NULL`.

Version Notes
HIView-based menu definitions are available in Mac OS X 10.3 and later.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Menus.h`

MenuDefUPP

Defines a universal procedure pointer to a menu definition function.

Not recommended

```
typedef MenuDefProcPtr MenuDefUPP;
```

Discussion
For more information, see the description of the `MenuDefProcPtr` (page 1349) callback.

Version Notes
In Mac OS X 10.3 and later, you should use HIView-based menu definitions instead.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Menus.h`

MenuHandle

Defines a menu reference.

```
typedef MenuRef MenuHandle;
```

Discussion
You should refer to menus using the `MenuRef` (page 1367) type instead.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Menus.h`

MenuID

Defines a menu ID.

```
typedef SInt16 MenuID;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Menus.h`

MenuItemDataRec

Used with the `SetMenuItemData` (page 1335) and `CopyMenuItemData` (page 1241) functions to get or change aspects of a menu item.

```
struct MenuItemDataRec {
    MenuItemDataFlags whichData;
    StringPtr text;
    UniChar mark;
    UniChar cmdKey;
    UInt32 cmdKeyGlyph;
    UInt32 cmdKeyModifiers;
    Style style;
    Boolean enabled;
    Boolean iconEnabled;
    UInt8 filler1;
    SInt32 iconID;
    UInt32 iconType;
    Handle iconHandle;
    MenuCommand cmdID;
    TextEncoding encoding;
    MenuID submenuID;
    MenuRef submenuHandle;
    SInt32 fontID;
    UInt32 refcon;
    OptionBits attr;
    CFStringRef cfText;
    Collection properties;
    UInt32 indent;
    UInt16 cmdVirtualKey;
};
typedef struct MenuItemDataRec MenuItemDataRec;
typedef MenuItemDataRec * MenuItemDataPtr;
```

Fields

`whichData`
The fields to be set or obtained. You pass a bit mask as specified by "Menu Item Data Flags" (page 1390) to indicate which values you want to get or set. The values themselves are set or populated in the fields that follow.

`text`
The menu item title, as an `Str255` string.

`mark`
The menu item's mark.

`cmdKey`
The menu item's command key. This can be either a character code or a virtual key code.

`cmdKeyGlyph`
The menu item's command key glyph.

`cmdKeyModifiers`
The menu item's command key modifiers.

`style`
The menu item's QuickDraw text style.

`enabled`
The menu item's enable state.

`iconEnabled`
The enable state of the menu item icon.

`filler1`
Reserved.

`iconID`
The icon resource ID of the menu item.

`iconHandle`
The icon handle of the menu item.

`cmdID`
The command ID for the menu item.

`encoding`
The text encoding of the menu item.

`submenuID`
The menu ID of the submenu associated with this menu item.

`submenuHandle`
The MenuRef of the submenu associated with this menu item.

`fontID`
The font ID for the menu item.

`refcon`
The reference constant associated with this menu item.

`attr`
The menu item's attributes.

`cfText`
The menu item's title, as a Core Foundation string.

`properties`
A collection holding the menu item's properties.

`indent`
The menu item's indent level.

`cmdVirtualKey`
The menu item's virtual key.

Discussion
When using this structure with `CopyMenuItemData` (page 1241) or `SetMenuItemData` (page 1335), the caller must first set the `whichData` field to a combination of `MenuItemDataFlags` indicating which specific data should be retrieved or set. Some fields also require initialization before calling `CopyMenuItemData` (page 1241); see "Menu Item Data Flags" (page 1390) for details.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Menus.h`

MenuItemID

Defines a menu item.

```
typedef UInt32 MenuItemID;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Menus.h`

MenuItemIndex

Specifies a particular menu item in a menu.

```
typedef UInt16 MenuItemIndex;
```

Discussion
The menu item index is one-based, so item 1 is the first menu item, item 2 is the second, and so on. Some functions allow you to pass an index of zero, which specifies the menu itself.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Menus.h`

MenuRef

Defines a menu.

```
typedef struct OpaqueMenuRef * MenuRef;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Menus.h`

MenuTrackingData

Contains information about the menu currently being displayed during menu tracking.

```
struct MenuTrackingData {
    MenuRef menu;
    MenuItemIndex itemSelected;
    MenuItemIndex itemUnderMouse;
    Rect itemRect;
    SInt32 virtualMenuTop;
    SInt32 virtualMenuBottom;
};
typedef struct MenuTrackingData MenuTrackingData;
typedef MenuTrackingData * MenuTrackingDataPtr;
```

Fields

`menu`

The menu.

`itemSelected`

The index of the menu item that is currently selected. This field should either match the `itemUnderMouse` field, or should be zero if the item under the mouse cannot be selected (for example, if the item is disabled).

`itemUnderMouse`

The index of the menu item that is currently under the mouse.

`itemRect`

The `Rect` that defines the area of the menu item currently under the mouse. Note that the `itemRect` field is not supported in CarbonLib and is always set to be empty. It is, however, supported in Mac OS X.

`virtualMenuTop`

The y-coordinate of the actual top of the menu. Because the user can scroll the menu, the menu top coordinate may be above the top of the visible screen (in which case it has a negative value).

`virtualMenuBottom`

The y-coordinate of the actual bottom of the menu. Because the user can scroll the menu, the menu bottom coordinate may be below the bottom of the visible screen

Discussion

You can call `GetMenuTrackingData` (page 1288) to obtain this structure during menu tracking.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Menus.h`

Constants

Contextual Menu Gestalt Selector Constants

Determine which contextual menu features are available.

```
enum {
    gestaltContextualMenuAttr = 'cmnu',
    gestaltContextualMenuUnusedBit = 0,
    gestaltContextualMenuTrapAvailable = 1,
    gestaltContextualMenuHasAttributeAndModifierKeys = 2,
    gestaltContextualMenuHasUnicodeSupport = 3
};
```

Constants

`gestaltContextualMenuAttr`
The `Gestalt` selector passed to the `Gestalt` function to determine whether contextual menu functions are available. Produces a value whose bits you should test to determine whether the contextual menu functions are available.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`gestaltContextualMenuUnusedBit`
Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`gestaltContextualMenuTrapAvailable`
If this bit is set, the contextual menu functions are available to 68K applications. If this bit is not set, these functions are not available to 68K applications.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`gestaltContextualMenuHasAttributeAndModifierKeys`
The contextual menu supports attributes and modifier keys.

Available in Mac OS X v10.2 and later.

Declared in `Menus.h`.

`gestaltContextualMenuHasUnicodeSupport`
The contextual menu supports Unicode text.

Available in Mac OS X v10.2 and later.

Declared in `Menus.h`.

Discussion

Mac OS X and all versions of Mac OS that shipped with CarbonLib support contextual menus, so you need to check only to see if certain features are available.

Contextual Menu Help Type Constants

Indicates what types of contextual menu help is available.

```
enum {
    kCMHelpItemNoHelp = 0,
    kCMHelpItemAppleGuide = 1,
    kCMHelpItemOtherHelp = 2,
    kCMHelpItemRemoveHelp = 3
};
```

Constants

`kCMHelpItemNoHelp`
The application does not support any help. The Menu Manager will put an appropriate help string into the menu and disable it.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kCMHelpItemAppleGuide`
The application supports Apple Guide help. The Menu Manager will put the name of the main Guide file into the menu and enable it.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kCMHelpItemOtherHelp`
The application supports some other form of help. In this case, the application must also pass a valid string into the `inHelpItemString` parameter of `ContextualMenuSelect`. This string will be the text of the help item in the menu, and the help item will be enabled.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kCMHelpItemRemoveHelp`
The application does not support any help. The Menu Manager will remove the Help item from the contextual menu.

Available in CarbonLib 1.6 and Mac OS X and later. Note however, that in CarbonLib, this constant is equivalent to `kCMItemNoHelp`, which only disables the Help item.

Available in Mac OS X v10.2 and later.

Declared in `Menus.h`.

Discussion

You can pass these constants in the `inHelpType` parameter of the function `ContextualMenuSelect` (page 1239) to specify the kind of help the application supports. Contextual menu help type constants are available with Appearance Manager 1.0 and later.

Contextual Menu Selection Type Constants

Indicates the type of item the user selected from a contextual menu.

```
enum {
    kCMNothingSelected = 0,
    kCMMenuItemSelected = 1,
    kCMShowHelpSelected = 3
};
```

Constants

`kCMNothingSelected`

The user did not choose an item from the contextual menu and the application should do no further processing of the event.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kCMMenuItemSelected`

The user chose one of the application's items from the menu. The application can examine the `outMenuID` and `outMenuItem` parameters of `ContextualMenuSelect` to see what the menu selection was, and it should then handle the selection appropriately.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kCMShowHelpSelected`

The user chose the Help item from the menu. The application should open an Apple Help database to a section appropriate for the selection. If the application supports some other form of help, it should be presented instead.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

Discussion

These constants are returned in the `outUserSelectionType` parameter of the function `ContextualMenuSelect` (page 1239) to specify what the user selected from the contextual menu. Contextual menu selection type constants are available with Appearance Manager 1.0 and later.

Contextual Menu Item Content Constants

Specify contents of menu items in contextual menus.

```
enum {
    keyContextualMenuName = 'pnam',
    keyContextualMenuCommandID = 'cmcd',
    keyContextualMenuSubmenu = 'cmsb',
    keyContextualMenuAttributes = 'cmat',
    keyContextualMenuModifiers = 'cmmd'
};
```

Constants

`keyContextualMenuName`

The menu item text. In Mac OS X v10.1 and earlier, the data format must be either `typeChar` or `typeIntlText`. In Mac OS X v10.2 and later, you can also specify `typeStyledText`, `typeAEText`, `typeUnicodeText`, and `typeCFStringRef`. Note that if you specify `typeCFStringRef`, the Menu Manager releases the CFString reference after displaying the menu. If you need to hold onto the CFString reference, you should retain it before inserting it into the Apple event record (`AERecord`).

Available in Mac OS X v10.2 and later.

Declared in `Menus.h`.

`keyContextualMenuCommandID`
: The menu item command ID. The data format for this parameter must be `typeLongInteger`.

  Available in Mac OS X v10.2 and later.

  Declared in `Menus.h`.

`keyContextualMenuSubmenu`
: The submenu of the menu item. You typically use this with the Apple Event Manager function `AEPutDesc` to add an entire `AEDesc` record (which contains the submenu) as the parameter data.

  Available in Mac OS X v10.2 and later.

  Declared in `Menus.h`.

`keyContextualMenuAttributes`
: Specifies the menu item attributes. The data format for this parameter must be `typeLongInteger`.

  Available in Mac OS X v10.2 and later.

  Declared in `Menus.h`.

`keyContextualMenuModifiers`
: Specifies the modifer keys for the menu item. The data format for this parameter must be `typeLongInteger`. By using this parameter along with the `keyContextualMenuAttributes` parameter, you can create dynamic contextual menu items that change according to the state of the modifier keys.

  Declared in `Menus.h`.

  Available in Mac OS X v10.2 and later.

Discussion
You use these keyword constants to specify parameters in an Apple Event record (`AERecord`) that defines a contextual menu item. Typically you assign these values in your ExamineContext method of a contextual menu plugin.

Custom Menu Definition Message Constants

Indicate messages used for non-HIView–based custom menu definitions.

```
enum {
    kMenuDrawMsg = 0,
    kMenuSizeMsg = 2,
    kMenuPopUpMsg = 3,
    kMenuCalcItemMsg = 5,
    kMenuThemeSavvyMsg = 7,
    kMenuInitMsg = 8,
    kMenuDisposeMsg = 9,
    kMenuFindItemMsg = 10,
    kMenuHiliteItemMsg = 11,
    kMenuDrawItemsMsg = 12,
    mDrawMsg = kMenuDrawMsg,
    mSizeMsg = kMenuSizeMsg,
    mPopUpMsg = kMenuPopUpMsg,
    mCalcItemMsg = kMenuCalcItemMsg
};
```

Constants

`kMenuDrawMsg`
Draw the menu in the specified rectangle.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuSizeMsg`
Calculate the dimensions of the menu rectangle and store them in the menu structure.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuPopUpMsg`
Calculate the dimensions of the pop-up menu.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuCalcItemMsg`
Calculate the dimensions of the specified menu item.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuThemeSavvyMsg`
Identify whether your menu definition function is theme-compliant. If so, your menu definition function should respond by passing back `kThemeSavvyMenuResponse` in the `whichItem` parameter. The Menu Manager then draws the menu background as appropriate for the current theme.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuInitMsg`
Perform any initializations required for the menu. Return an error code in `*whichItem` to indicate success or failure.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuDisposeMsg`
Dispose of the menu.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuFindItemMsg`
Determine the item underneath the mouse.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuHiliteItemMsg`
Highlight the specified menu item.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuDrawItemsMsg`
Draw the specified menu items.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`mDrawMsg`
Same as `kMenuDrawMsg`. Obsolete.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`mSizeMsg`
Same as `kMenuSizeMsg`. Obsolete.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`mPopUpMsg`
Same as `kMenuPopUpMsg`. Obsolete.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`mCalcItemMsg`
Same as `kMenuCalcItemMsg`. Obsolete.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

Discussion
For applications running on Mac OS X v10.3 and later, Apple recommends creating custom menu definitions using HIView subclasses rather than MDEF messages. See Introducing HIView for more information.

The Menu Manager passes a value defined by one of these constants in the `message` parameter of your menu definition function specifying what action your function must perform. Other messages are reserved for internal use by Apple Computer, Inc. For more information on how to respond to the various messages, see `MenuDefProcPtr` (page 1349).

Obsolete Menu Definition Messages

Older MDEF messages.

```
enum {
    mChooseMsg = 1,
    mDrawItemMsg = 4,
    kMenuChooseMsg = mChooseMsg,
    kMenuDrawItemMsg = mDrawItemMsg
};
```

Constants

`mChooseMsg`

Determine whether the specified mouse location is in an enabled menu item, and highlight or unhighlight the menu item appropriately. Carbon `MDEF`s must replace `mChooseMsg` with the new messages `kMenuFindItemMsg` and `kMenuHiliteItemMsg`.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`mDrawItemMsg`

Draw the specified menu item in the specified rectangle. `mDrawItemMsg` was used by the popup menu control in versions of the Mac OS prior to Mac OS 8.5, but is no longer used.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuChooseMsg`

Same as `mChooseMsg`.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuDrawItemMsg`

Same as `mDrawItemMsg`.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

Hierarchical Font Menu Option Constant

Indicates that the font menu should be hierarchical.

```
enum {
    kHierarchicalFontMenuOption = 0x00000001
};
```

Constants

`kHierarchicalFontMenuOption`

The parent menu displays the font families, with font variations (plain, bold, and so on) displayed in submenus.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

Discussion

Use this option constant when calling the `CreateStandardFontMenu` (page 1247) function.

Menu Attribute Constants

Specify menu attributes.

```
typedef UInt32 MenuAttributes;
enum {
    kMenuAttrExcludesMarkColumn = (1 << 0),
    kMenuAttrAutoDisable = (1 << 2),
    kMenuAttrUsePencilGlyph = (1 << 3),
    kMenuAttrHidden = (1 << 4),
    kMenuAttrCondenseSeparators = (1 << 5),
    kMenuAttrDoNotCacheImage = (1 << 6),
    kMenuAttrDoNotUseUserCommandKeys = (1 << 7)
};
```

Constants

`kMenuAttrExcludesMarkColumn`
No column space is allocated for the mark character when this menu is drawn.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuAttrAutoDisable`
The menu title is automatically disabled when all of its menu items are disabled.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuAttrUsePencilGlyph`
Use the pencil glyph from the Keyboard font (`kMenuPencilGlyph`) to draw the control modifier keys when drawing keyboard equivalents. Typically used only for Japanese input method menus.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuAttrHidden`
Do not draw the menu title, even when the menu is inserted in the menu bar. This attribute is useful for specifying keyboard equivalent commands that don't correspond with visible menu items. That is, you can add command key equivalents to to menu items and keep the menu itself from appearing in the menu.

Declared in `Menus.h`.

Available in Mac OS X v10.2 and later.

`kMenuAttrCondenseSeparators`
Hides extra separators to avoid blank spaces in a menu. That is, if separators exists at the beginning or end of a menu, or if multiple contiguous separators exist, the Menu Manager marks the extra separator items as hidden. The Menu Manager checks for extra separators whenever it recalculates the menu size.

Declared in `Menus.h`.

Available in Mac OS X v10.3 and later.

`kMenuAttrDoNotCacheImage`
Disables automatic caching of the menu image. Normally, the Menu Manager caches images of all HIView–based menus. (All standard menus are drawn using HIViews in Mac OS X v10.3 and later.) If you specify this attribute, the Menu Manager draws the menu each time it is displayed.

Declared in `Menus.h`.

Available in Mac OS X v10.3 and later.

`kMenuAttrDoNotUseUserCommandKeys`

Disables substitution of command key equivalents from the `NSUserKeysEquivalents` dictionary. By default, the Menu Manager checks for matches in the dictionary for every menu item. Note that this attribute is effective only if you set it when you create the menu; After the Menu Manager searches the dictionary and sets the user command keys (which occurs in the `CalcMenuSize` (page 1234), `GetMenuItemCommandKey` (page 1276), `GetItemCmd` (page 1262) and before command key matching), you cannot retrieve the original command keys. Similarly, clearing this attribute does not restore the original command keys.

Declared in `Menus.h`.

Available in Mac OS X v10.3 and later.

Discussion

Menu attributes control behavior of the entire menu. They are used with the `ChangeMenuAttributes` (page 1235) and `GetMenuAttributes` (page 1268).

Menu Item Attribute Constants

Specify attributes for menu items.

```
typedef UInt32 MenuItemAttributes;
enum {
    kMenuItemAttrDisabled = (1 << 0),
    kMenuItemAttrIconDisabled = (1 << 1),
    kMenuItemAttrSubmenuParentChoosable = (1 << 2),
    kMenuItemAttrDynamic = (1 << 3),
    kMenuItemAttrNotPreviousAlternate = (1 << 4),
    kMenuItemAttrHidden = (1 << 5),
    kMenuItemAttrSeparator = (1 << 6),
    kMenuItemAttrSectionHeader = (1 << 7),
    kMenuItemAttrIgnoreMeta = (1 << 8),
    kMenuItemAttrAutoRepeat = (1 << 9),
    kMenuItemAttrUseVirtualKey = (1 << 10),
    kMenuItemAttrCustomDraw = (1 << 11),
    kMenuItemAttrIncludeInCmdKeyMatching = (1 << 12),
    kMenuItemAttrAutoDisable = (1 << 13),
    kMenuItemAttrUpdateSingleItem = (1 << 14)
};
```

Constants

`kMenuItemAttrDisabled`

This menu item is disabled.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuItemAttrIconDisabled`

This menu item's icon is disabled.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuItemAttrSubmenuParentChoosable`

The user can select the parent item of a submenu.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuItemAttrDynamic`

This menu item changes dynamically based on the state of the modifier keys. For example, holding down the command key might change the menu item from “Select widget” to “Select all widgets.”

When a menu item has alternate dynamic states, you should group them together sequentially in the menu and assign them the same command key. A collection of menu item alternates is called a dynamic group.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuItemAttrNotPreviousAlternate`

This item is not part of the same dynamic group as the previous item. The Menu Manager determines which menu items belong to a dynamic group by examining the command keys of each item; if a menu item has the same command key as the previous item, the Menu Manager considers it to be part of the same dynamic group.

However, in some cases you may have sequential items with the same command key (or no command key at all) that should not be considered part of the same dynamic group. To distinguish the separation, you should set this flag for the first menu item in the new group.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuItemAttrHidden`

The menu item is not drawn when displaying the menu. The item is also not included in command-key matching unless the `kMenuItemAttrDynamic` or `kMenuItemIncludeInCmdKeyMatching` attribute is set.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuItemAttrSeparator`

The menu item is a separator; any text in the item is ignored.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuItemAttrSectionHeader`

The menu item is a menu section header; this item is disabled and not selectable.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuItemAttrIgnoreMeta`

Ignore the dash (-) metacharacter in this menu item. Dashes at the beginning of a menu item title traditionally signify that the menu item is a separator. However, in some cases you might want to display the dash in the title (for example, if you wanted the menu item to read “-40 degrees F.”)

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuItemAttrAutoRepeat`

The `IsMenuKeyEvent` (page 1305) event function recognizes this menu item when it receives an autorepeat keyboard event.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuItemAttrUseVirtualKey`
When `MenuEvent` (page 1309) and `IsMenuKeyEvent` (page 1305) compare this menu item's keyboard equivalent against a keyboard event, they use the item's virtual keycode equivalent rather than its character code equivalent.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuItemAttrCustomDraw`
This is a custom menu item. Setting this attribute causes custom menu item drawing Carbon events to be sent to your application. Available in CarbonLib 1.4 and Mac OS X v10.1 and later.

Available in Mac OS X v10.1 and later.

Declared in `Menus.h`.

`kMenuItemAttrIncludeInCmdKeyMatching`
If this attribute is set, functions such as `MenuKey` (page 1311), `MenuEvent` (page 1309) and `IsMenuKeyEvent` (page 1305) examine this menu item during command key matching. Typically, visible items are examined and hidden items (unless they have the `kMenuItemAttrDynamic` attribute set) are ignored during command key matching. However, by setting this attribute, you can force hidden items to be included in the matching. Available in CarbonLib 1.6 and Mac OS X v10.2 and later.

Available in Mac OS X v10.2 and later.

Declared in `Menus.h`.

`kMenuItemAttrAutoDisable`
Disables the menu item if it does not respond to the `kEventCommandUpdateStatus` event . That is, if no `kEventCommandUpdateStatus` handler is installed on this item, or if all the handlers installed for the update event return `eventNotHandledErr`, this item is automatically disabled. This attribute is useful if your application uses the `kEventCommandUpdateStatus` event to enable menu items; for example you no longer have to install an update status handler on the application target to disable menu items when there are no document windows open.

Available in Mac OS X v10.3 and later.

Declared in `Menus.h`.

`kMenuItemAttrUpdateSingleItem`
Update only the menu item that matches when searching available command keys. Normally when the Menu Manager does command key matching, it sends a `kEventMenuEnableItems` event to the menu containing the matching item and then sends a `kEventCommandUpdateStatus` to each item in the menu. Doing so can be inefficient, since in most cases only the item that matches needs to be updated. By setting this attribute, only the matching item receives the update event and `kEventMenuEnableItems` is not sent to the menu. If your application enables menu items solely through `kEventCommandUpdateStatus` event handlers, you should set this attribute for your menu items.

Declared in `Menus.h`.

Available in Mac OS X v10.3 and later.

Discussion
Menu item attributes control behavior of individual menu items. They are used with the `GetMenuItemAttributes` (page 1275) and `ChangeMenuItemAttributes` (page 1235) APIs.

Menu Definition Type Constants

Indicate the type of menu definition being used.

```
enum {
    kMenuDefProcPtr = 0,
    kMenuDefClassID = 1
};
typedef UInt32 MenuDefType;
```

Constants

`kMenuDefProcPtr`

A custom menu definition using the older MDEF messaging model.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuDefClassID`

A custom menu definition using an HIView subclass.

Available in Mac OS X v10.3 and later.

Declared in `Menus.h`.

Menu Definition Feature Constants

Indicate menu definition features.

```
enum {
    kThemeSavvyMenuResponse = 0x7473
};
```

Constants

`kThemeSavvyMenuResponse`

Indicates that the menu is Appearance theme–savvy.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

Discussion

The Menu Manager may pass the `kMenuThemeSavvyMsg` constant in the `message` parameter of your menu definition function to determine if your custom menu is theme-savvy (that is, whether it draws using Appearance Manager functions). In response, your menu definition function may respond with this flag in the `whichItem` parameter.

Menu Definition IDs

Specify options used in menu item functions.

```
enum {
    textMenuProc = 0,
    hMenuCmd = 27,
    hierMenu = -1,
    kInsertHierarchicalMenu = -1,
};
```

Constants

`textMenuProc`

The menu definition ID for menus that are not Appearance-compliant. When mapping is enabled, this constant is mapped to `kMenuStdMenuProc`, its Appearance-compliant equivalent. Not normally used.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`hMenuCmd`

Deprecated. Use `SetMenuItemHierarchicalMenu` (page 1337) or `SetMenuItemHierarchicalID` (page 1336) to specify a hierarchical menu instead.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`hierMenu`

Deprecated. Use `kInsertHierarchicalMenu` instead.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kInsertHierarchicalMenu`

Used with `InsertMenu` (page 1294) to insert a submenu or pop-up menu into the submenu portion of the current menu list.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kHIMenuAppendItem`

Pass to `InsertMenuItem` (page 1295), `InsertMenuItemText` (page 1296), or `InsertMenuItemTextWithCFString` (page 1296) to indicate that the new menu item should be added to the end of the menu. Note that you can simply call `AppendMenu` (page 1229), `AppendMenuItemText` (page 1231), or `AppendMenuItemTextWithCFString` (page 1231) instead.

Available in Mac OS X v10.3 and later.

Declared in `Menus.h`.

Discussion

A menu definition ID is supplied to the menu resource or a menu-creation function such as `NewMenu` to specify which menu definition function to use in creating the menu. The menu definition ID contains the resource ID of the menu definition function.

Menu Event Option Constants

Specify options when attempting to match a keyboard event to a menu item.

```
typedef UInt32 MenuEventOptions;
enum {
    kMenuEventIncludeDisabledItems = 0x0001,
    kMenuEventQueryOnly = 0x0002,
    kMenuEventDontCheckSubmenus = 0x0004
};
```

Constants

`kMenuEventIncludeDisabledItems`

Disabled items are examined for a match.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuEventQueryOnly`

Don't highlight the menu title if a match is found.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuEventDontCheckSubmenus`

Don't search the submenus of the starting menu when looking for a match.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

Discussion

Menu event options control how the menus are searched for an item matching a particular keyboard event. They are used with the `IsMenuKeyEvent` (page 1305) API.

Menu Glyph Constants

Specify menu glyphs.

```
enum {
     kMenuNullGlyph = 0x00,
     kMenuTabRightGlyph = 0x02,
     kMenuTabLeftGlyph = 0x03,
     kMenuEnterGlyph = 0x04,
     kMenuShiftGlyph = 0x05,
     kMenuControlGlyph = 0x06,
     kMenuOptionGlyph = 0x07,
     kMenuSpaceGlyph = 0x09,
     kMenuDeleteRightGlyph = 0x0A,
     kMenuReturnGlyph = 0x0B,
     kMenuReturnR2LGlyph = 0x0C,
     kMenuNonmarkingReturnGlyph = 0x0D,
     kMenuPencilGlyph = 0x0F,
     kMenuDownwardArrowDashedGlyph = 0x10,
     kMenuCommandGlyph = 0x11,
     kMenuCheckmarkGlyph = 0x12,
     kMenuDiamondGlyph = 0x13,
     kMenuAppleLogoFilledGlyph = 0x14,
     kMenuParagraphKoreanGlyph = 0x15,
     kMenuDeleteLeftGlyph = 0x17,
     kMenuLeftArrowDashedGlyph = 0x18,
     kMenuUpArrowDashedGlyph = 0x19,
     kMenuRightArrowDashedGlyph = 0x1A,
     kMenuEscapeGlyph = 0x1B,
     kMenuClearGlyph = 0x1C,
     kMenuLeftDoubleQuotesJapaneseGlyph = 0x1D,
     kMenuRightDoubleQuotesJapaneseGlyph = 0x1E,
     kMenuTrademarkJapaneseGlyph = 0x1F,
     kMenuBlankGlyph = 0x61,
     kMenuPageUpGlyph = 0x62,
     kMenuCapsLockGlyph = 0x63,
     kMenuLeftArrowGlyph = 0x64,
     kMenuRightArrowGlyph = 0x65,
     kMenuNorthwestArrowGlyph = 0x66,
     kMenuHelpGlyph = 0x67,
     kMenuUpArrowGlyph = 0x68,
     kMenuSoutheastArrowGlyph = 0x69,
     kMenuDownArrowGlyph = 0x6A,
     kMenuPageDownGlyph = 0x6B,
     kMenuAppleLogoOutlineGlyph = 0x6C,
     kMenuContextualMenuGlyph = 0x6D,
     kMenuPowerGlyph = 0x6E,
     kMenuF1Glyph = 0x6F,
     kMenuF2Glyph = 0x70,
     kMenuF3Glyph = 0x71,
     kMenuF4Glyph = 0x72,
     kMenuF5Glyph = 0x73,
     kMenuF6Glyph = 0x74,
     kMenuF7Glyph = 0x75,
     kMenuF8Glyph = 0x76,
     kMenuF9Glyph = 0x77,
     kMenuF10Glyph = 0x78,
     kMenuF11Glyph = 0x79,
     kMenuF12Glyph = 0x7A,
     kMenuF13Glyph = 0x87,
     kMenuF14Glyph = 0x88,
     kMenuF15Glyph = 0x89,
```

```
    kMenuControlISOGlyph = 0x8A,
    kMenuEjectGlyph = 0x8C
};
```

Constants

`kMenuNullGlyph`
The null character. Note that this glyph has no visible representation (that is, nothing appears in the menu).

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuTabRightGlyph`
The Tab-to-the-right key. Used in left to right script systems.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuTabLeftGlyph`
The Tab-to-the-left key. Used in right to left script systems.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuEnterGlyph`
The Enter key.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuShiftGlyph`
The Shift key.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuControlGlyph`
The Control key.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuOptionGlyph`
The Option key.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuSpaceGlyph`
The Space bar. Note that this glyph has no visible representation (that is, nothing appears in the menu).

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuDeleteRightGlyph`
The Delete-to-the-right key. Used in right-to-left script systems.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuReturnGlyph`
The Return key for left-to-right script systems.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuReturnR2LGlyph`
The Return key for right-to-left script systems.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuNonmarkingReturnGlyph`
The nonmarking Return key. Note that this glyph has no visible representation (that is, nothing appears in the menu).

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuPencilGlyph`
The Pencil key.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuDownwardArrowDashedGlyph`
The downward dashed arrow key.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuCommandGlyph`
The Command key.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuCheckmarkGlyph`
The Check mark key.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuDiamondGlyph`
The diamond mark.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuAppleLogoFilledGlyph`
The filled Apple logo.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuParagraphKoreanGlyph`
Unassigned. (Paragraph glyph in Korean)

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuDeleteLeftGlyph`
The Delete-to-the-left key.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuLeftArrowDashedGlyph`
The dashed left arrow key.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuUpArrowDashedGlyph`
The dashed up arrow key.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuRightArrowDashedGlyph`
The dashed right arrow key.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuEscapeGlyph`
The Escape key.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuClearGlyph`
The Clear key.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuLeftDoubleQuotesJapaneseGlyph`
Unassigned. (Left double quotation marks in Japanese)
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuRightDoubleQuotesJapaneseGlyph`
Unassigned (Right double quotation marks in Japanese)
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuTrademarkJapaneseGlyph`
Unassigned. (Trademark in Japanese)
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuBlankGlyph`
The blank key.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuPageUpGlyph`
The Page Up key.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuCapsLockGlyph`
The Caps Lock key.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuLeftArrowGlyph`
The left arrow key.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuRightArrowGlyph`
The right arrow key.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuNorthwestArrowGlyph`
The northwest arrow key
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuHelpGlyph`
The Help key.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuUpArrowGlyph`
The up arrow key.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuSoutheastArrowGlyph`
The southeast arrow key.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuDownArrowGlyph`
The down arrow key.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuPageDownGlyph`
The Page Down key.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuAppleLogoOutlineGlyph`
The outlined Apple logo.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuContextualMenuGlyph`
The contextual menu key
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuPowerGlyph`
The power key (that is, the startup key).
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuF1Glyph`
The F1 key.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuF2Glyph`
The F2 key.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuF3Glyph`
The F3 key.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuF4Glyph`
The F4 key.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuF5Glyph`
The F5 key.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuF6Glyph`
The F6 key.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuF7Glyph`
The F7 key.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuF8Glyph`
The F8 key.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuF9Glyph`
The F9 key.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuF10Glyph`
The F10 key.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuF11Glyph`
The F11 key.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuF12Glyph`
The F12 key.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuF13Glyph`
The F13 key.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuF14Glyph`
The F14 key.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuF15Glyph`
The F15 key.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuControlISOGlyph`
The ISO standard control key.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuEjectGlyph`
The Eject key (available in Mac OS X v10.2 and later).
Available in Mac OS X v10.2 and later.
Declared in `Menus.h`.

Discussion
Use these constants with `GetMenuItemKeyGlyph` (page 1280) and `SetMenuItemKeyGlyph` (page 1339).

Menu Item Data Flags

Indicate which fields of a `MenuItemDataRec` structure are to be copied or set.

```
enum {
    kMenuItemDataText = (1 << 0),
    kMenuItemDataMark = (1 << 1),
    kMenuItemDataCmdKey = (1 << 2),
    kMenuItemDataCmdKeyGlyph = (1 << 3),
    kMenuItemDataCmdKeyModifiers = (1 << 4),
    kMenuItemDataStyle = (1 << 5),
    kMenuItemDataEnabled = (1 << 6),
    kMenuItemDataIconEnabled = (1 << 7),
    kMenuItemDataIconID = (1 << 8),
    kMenuItemDataIconHandle = (1 << 9),
    kMenuItemDataCommandID = (1 << 10),
    kMenuItemDataTextEncoding = (1 << 11),
    kMenuItemDataSubmenuID = (1 << 12),
    kMenuItemDataSubmenuHandle = (1 << 13),
    kMenuItemDataFontID = (1 << 14),
    kMenuItemDataRefcon = (1 << 15),
    kMenuItemDataAttributes = (1 << 16),
    kMenuItemDataCFString = (1 << 17),
    kMenuItemDataProperties = (1 << 18),
    kMenuItemDataIndent = (1 << 19),
    kMenuItemDataCmdVirtualKey = (1 << 20),
    kMenuItemDataAllDataVersionOne = 0x000FFFFF,
    kMenuItemDataAllDataVersionTwo = kMenuItemDataAllDataVersionOne
| kMenuItemDataCmdVirtualKey
};
enum {
    kMenuItemDataAllData = kMenuItemDataAllDataVersionTwo
};
typedef UInt64 MenuItemDataFlags;
```

Constants

`kMenuItemDataText`

Set or return the `Str255` text of a menu using the `MenuItemDataRec.text` field. If getting the text, the text field must be initialized with a pointer to a `Str255` variable before calling `CopyMenuItemData`. If both `kMenuItemDataText` and `kMenuItemCFString` are set on entry to `CopyMenuItemData`, the API will determine whether the menu text was most recently set using a `Str255` or CFString, and return only that text format; the flags value for the other format will be cleared. Valid for both menu items and the menu title (if item number is 0).

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuItemDataMark`

Set or return the mark character of a menu item using the `MenuItemDataRec.mark` field. Valid only for menu items.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuItemDataCmdKey`
: Set or return the command key of a menu using the `MenuItemDataRec.cmdKey` field. Valid only for menu items.

  Available in Mac OS X v10.0 and later.

  Declared in `Menus.h`.

`kMenuItemDataCmdKeyGlyph`
: Set or return the command key glyph of a menu using the `MenuItemDataRec.cmdKeyGlyph` field. Valid only for menu items.

  Available in Mac OS X v10.0 and later.

  Declared in `Menus.h`.

`kMenuItemDataCmdKeyModifiers`
: Set or return the command key modifiers of a menu using the `MenuItemDataRec.cmdKeyModifiers` field. Valid only for menu items.

  Available in Mac OS X v10.0 and later.

  Declared in `Menus.h`.

`kMenuItemDataStyle`
: Set or return the QuickDraw text style of a menu item using the `MenuItemDataRec.style` field. Valid only for menu items.

  Available in Mac OS X v10.0 and later.

  Declared in `Menus.h`.

`kMenuItemDataEnabled`
: Set or return the enable state of a menu using the `MenuItemDataRec.enabled` field. Valid for both menu items and the menu itself (if the item number is zero).

  Available in Mac OS X v10.0 and later.

  Declared in `Menus.h`.

`kMenuItemDataIconEnabled`
: Set or return the enable state of the menu item icon using the `MenuItemDataRec.iconEnabled` field. Valid only for menu items.

  Available in Mac OS X v10.0 and later.

  Declared in `Menus.h`.

`kMenuItemDataIconID`
: Set or return the icon resource ID of the menu item icon using the `MenuItemDataRec.iconID` field. Valid only for menu items.

  Available in Mac OS X v10.0 and later.

  Declared in `Menus.h`.

`kMenuItemDataIconHandle`

Set or return the icon handle of a menu item using the `MenuItemDataRec.iconType` and `MenuItemDataRec.iconHandle` field. You must initialize both fields if you are setting the handle; both fields are returned when obtaining the handle.

The `iconType` field can contain one of the following constants: `kMenuIconType`, `kMenuShrinkIconType`, `kMenuSmallIconType`, `kMenuColorIcontype`, `kMenuIconSuiteType`, or `kMenuIconRefType`. The icon handle may be a handle to an `'ICON'` resource, a `'SICN'` resource, a `'cicn'` resource, an icon suite, or an icon reference. Valid only for menu items.

In Mac OS X v10.0 and later, the `iconType` field can also contain `kMenuCGImageType`, with the icon handle being of type `CGImageRef`.

In Mac OS X v10.1 and later, the `iconType` field can also contain `kMenuSystemIconSelectorType` or `kMenuIconResource`, which have icon handles of type `OSType` and `CFStringRef` respectively.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuItemDataCommandID`

Set or return the command ID of a menu using the `MenuItemDataRec.cmdID` field. Valid only for menu items.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuItemDataTextEncoding`

Set or return the text encoding of a menu item using the `MenuItemDataRec.encoding` field. Valid only for menu items.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuItemDataSubmenuID`

Set or return the menu ID of the submenu associated with this menu item using the `MenuItemDataRec.submenuID` field. Valid only for menu items.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuItemDataSubmenuHandle`

Set or return the menu reference (`MenuRef`) of the submenu associated with this menu using the `MenuItemDataRec.submenuHandle` field. Valid only for menu items.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuItemDataFontID`

Set or return the font ID associated with this menu item using the `MenuItemDataRec.fomtID` field. Valid only for menu items.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuItemDataRefcon`

Set or return the reference constant associated with this menu item using the `MenuItemDataRec.refcon` field. If you specified a menu item index of 0, you can set or obtain the menu reference constant.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuItemDataAttributes`
Set or return the attribute bits associated with this menu item using the `MenuItemDataRec.attr` field. If you specified a menu item index of 0, you can set or obtain a `MenuAttributes` bit field, not a `MenuItemAttributes` bit field.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuItemDataCFString`
Set or return the title of the menu item (as a Core Foundation string) using the `MenuItemDataRec.cFText` field. If you specified a menu item index of 0, you can set or obtain the menu title.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuItemDataProperties`
Set or return the properties of a menu using the `MenuItemDataRec.properties` field.

If setting properties, the `properties` field should contain a collection with the new properties; note that this will overwrite any existing properties with the same collection creator and tag.

If getting properties, you should set the `properties` field to either a valid collection or `NULL`. A valid collection is overwritten by the new properties. If you pass `NULL`, the `CopyMenuItemData` (page 1241) function allocates a new collection and returns it in the `properties` field.

You can set this flag for both menu items and the menu itself (if the item number is zero).

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuItemDataIndent`
Set or return the indent level of a menu item using the `MenuItemDataRec.indent` field. Valid only for menu items.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuItemDataCmdVirtualKey`
Set or return the virtual key code for this menu item using the `MenuItemDataRec.cmdVirtualKey` field. Valid only for menu items.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuItemDataAllDataVersionOne`
Sets all flags, except for `kMenuItemDataCmdVirtualKey`.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuItemDataAllDataVersionTwo`
Sets all flags, including `kMenuItemDataCmdVirtualKey`.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

Discussion
A `MenuItemDataFlags` value indicates which fields of a "MenuItemDataRec" (page 1365) structure should be used by the `CopyMenuItemData` (page 1241) or `SetMenuItemData` (page 1335) functions. All menu item data flags may be used when getting or setting the contents of a menu item; some may also be used when getting or setting information about the menu itself, if the item index given to `CopyMenuItemData` (page 1241) or `SetMenuItemData` (page 1335) is 0.

Menu Item Icon Type Constants

Specify types of icons to attach to menu items.

```
enum {
    kMenuNoIcon = 0,
    kMenuIconType = 1,
    kMenuShrinkIconType = 2,
    kMenuSmallIconType = 3,
    kMenuColorIconType = 4,
    kMenuIconSuiteType = 5,
    kMenuIconRefType = 6,
    kMenuCGImageRefType = 7,
    kMenuSystemIconSelectorType = 8,
    kMenuIconResourceType = 9
};
```

Constants

`kMenuNoIcon`
No icon.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuIconType`
Identifies an icon of type `'ICON'`.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuShrinkIconType`
Identifies a 32-by-32-pixel icon of type `'ICON'`, shrunk (at display time) to 16-by-16.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuSmallIconType`
Identifies an icon of type `'SICN'`.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuColorIconType`
Identifies an icon of type 'cicn'.
Available in Mac OS X v10.0 and later.
Declared in `Menus.h`.

`kMenuIconSuiteType`
Identifies an icon suite.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuIconRefType`
Identifies an icon of type `IconRef`. This value is supported under Mac OS 8.5 and later.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuCGImageRefType`
Identifies an icon of type `CGImageRef`.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuSystemIconSelectorType`
Identifies an `OSType` value that corresponds to an icon (type `IconRef`)registered with Icon Services under `kSystemIconsCreator`.

Available in Mac OS X v10.1 and later.

Declared in `Menus.h`.

`kMenuIconResourceType`
Identifies a CFString that names an icon resource in the main bundle of the application.

Available in Mac OS X v10.1 and later.

Declared in `Menus.h`.

Discussion
These constants specify the type of an icon attached to a menu item. They are passed in `SetMenuItemIconHandle` (page 1338) and obtained by `GetMenuItemIconHandle` (page 1279). Menu item icon type constants are available with Appearance Manager 1.0 and later.

Menu Item Property Attribute Constant

Define attributes to associate with menu item properties.

```
enum {
    kMenuPropertyPersistent = 0x00000001
};
```

Constants
`kMenuPropertyPersistent`
If this bit is set, the menu item property is saved when the menu is flattened.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

Menu Tracking Mode Constants

Indicates how the menu is being tracked.

```
typedef UInt32 MenuTrackingMode;
enum {
    kMenuTrackingModeMouse = 1,
    kMenuTrackingModeKeyboard = 2
};
```

Constants

`kMenuTrackingModeMouse`

Menus are being tracked using the mouse.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuTrackingModeKeyboard`

Menus are being tracked using the keyboard.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

Discussion

A menu tracking mode constant is part of the `kEventMenuBeginTracking` and `kEventMenuChangeTrackingMode` Carbon events. It indicates whether a menu is being tracked by mouse movement or by directional keyboard input.

Modifier Key Mask Constants

Specify modifier keys used with menu item selections.

```
enum {
    kMenuNoModifiers = 0,
    kMenuShiftModifier = (1 << 0),
    kMenuOptionModifier = (1 << 1),
    kMenuControlModifier = (1 << 2),
    kMenuNoCommandModifier = (1 << 3)
};
```

Constants

`kMenuNoModifiers`

If no bit is set, only the Command key is used in the keyboard equivalent.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuShiftModifier`

If this bit (bit 0) is set, the Shift key is used in the keyboard equivalent.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuOptionModifier`

If this bit (bit 1) is set, the Option key is used in the keyboard equivalent.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuControlModifier`
If this bit (bit 2) is set, the Control key is used in the keyboard equivalent.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

`kMenuNoCommandModifier`
If this bit (bit 3) is set, the Command key is not used in the keyboard equivalent.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

Discussion
You can use one or more of these mask constants to determine which modifier key(s) must be pressed along with a character key to create a keyboard equivalent for selecting a menu item. You set and obtain these constants by calling `SetMenuItemModifiers` (page 1340) and `GetMenuItemModifiers` (page 1281), respectively.

No Mark Marking Character Constant

Indicates that a menu item contains no marking characters.

```
enum {
    noMark = 0
};
```

Constants
`noMark`
No marking character to be associated with a menu or submenu item.

Available in Mac OS X v10.0 and later.

Declared in `Menus.h`.

Discussion
You can pass this constant, as well as those character marking constants defined in the Font Manager, in the `markChar` parameter of the function `SetItemMark` (page 1324) and the marking character field of the menu resource (of type '`MENU`') and return these constants in the `markChar` parameter of the function `GetItemMark` (page 1263) to specify the mark of a specific menu item or the menu ID of the submenu associated with the menu item.

Menu Dismissal Constants

Specify reasons why menu tracking ended.

```
enum {
kHIMenuDismissedBySelection   = 1,
kHIMenuDismissedByUserCancel  = 2,
kHIMenuDismissedByMouseDown   = 3,
kHIMenuDismissedByMouseUp     = 4,
kHIMenuDismissedByKeyEvent    = 5,
kHIMenuDismissedByAppSwitch   = 6,
kHIMenuDismissedByTimeout     = 7,
kHIMenuDismissedByCancelMenuTracking = 8,
kHIMenuDismissedByActivationChange = 9,
kHIMenuDismissedByFocusChange = 10
};
```

Constants

`kHIMenuDismissedBySelection`
The user selected a menu item.
Available in Mac OS X v10.3 and later.
Declared in `Menus.h`.

`kHIMenuDismissedByUserCancel`
The user cancelled menu tracking.
Available in Mac OS X v10.3 and later.
Declared in `Menus.h`.

`kHIMenuDismissedByMouseDown`
The user pressed the mouse someplace that did not result in a menu item selection.
Available in Mac OS X v10.3 and later.
Declared in `Menus.h`.

`kHIMenuDismissedByMouseUp`
The user released the mouse someplace that did not result in a menu item selection.
Available in Mac OS X v10.3 and later.
Declared in `Menus.h`.

`kHIMenuDismissedByKeyEvent`
A keyboard event occurred.
Available in Mac OS X v10.3 and later.
Declared in `Menus.h`.

`kHIMenuDismissedByAppSwitch`
The application with the menu is no longer frontmost.
Available in Mac OS X v10.3 and later.
Declared in `Menus.h`.

`kHIMenuDismissedByTimeout`
The menu tracking mode timed out.
Available in Mac OS X v10.3 and later.
Declared in `Menus.h`.

`kHIMenuDismissedByCancelMenuTracking`
The application called `CancelMenuTracking`.
Available in Mac OS X v10.3 and later.
Declared in `Menus.h`.

`kHIMenuDismissedByActivationChange`
The active window changed.

Available in Mac OS X v10.3 and later.

Declared in `Menus.h`.

`kHIMenuDismissedByFocusChange`
The user focus window changed, or the keyboard focus was removed from the current process.

Available in Mac OS X v10.3 and later.

Declared in `Menus.h`.

Discussion
The Carbon Event Manager passes these constants in the `kEventMenuEndTrackingEvent` to indicate why menu tracking ended.

Standard Menu Definition Constants

Specify the menu definitions for standard menus and menu bars.

```
enum {
    kMenuStdMenuProc = 63,
    kMenuStdMenuBarProc = 63
};
```

Constants
`kMenuStdMenuProc`
The menu definition ID for Appearance-compliant menus.

Available with Appearance Manager 1.0 and later.

Declared in `Menus.h`.

`kMenuStdMenuBarProc`
The menu bar definition ID for Appearance-compliant menu bars.

Available with Appearance Manager 1.0 and later.

Declared in `Menus.h`.

Result Codes

This table lists result codes defined for the Menu Manager.

Result Code	Value	Description
`menuPropertyInvalidErr`	-5603	You specified an Apple-reserved creator type in a menu property function. Available in Mac OS X v10.0 and later.
`menuPropertyNotFoundErr`	-5604	The specified property creator/ID combination was not found. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
`menuNotFoundErr`	-5620	The specified menu or menu ID wasn't found. Available in Mac OS X v10.0 and later.
`menuUsesSystemDefErr`	-5621	`GetMenuDefinition` failed because the menu uses the system menu definition. Available in Mac OS X v10.0 and later.
`menuItemNotFoundErr`	-5622	The specified menu item wasn't found. Available in Mac OS X v10.0 and later.
`menuInvalidErr`	-5623	The menu reference passed to the function was invalid. Available in Mac OS X v10.0 and later.

Navigation Services Reference

Framework: Carbon/Carbon.h

Declared in Navigation.h

Overview

Navigation Services is an application programming interface that allows your application to provide a user interface for navigating, opening, and saving Mac OS file objects.

This reference describes the application programming interface for Navigation Services, as introduced with CarbonLib 1.1. Navigation Services establishes a new model for creating, displaying, and processing dialogs. This new functionality gives you the ability to create truly modeless dialogs and provides support for Unicode and Mac OS X sheets.

Navigation Services replaces the Standard File Package, which is not supported in Carbon.

Functions by Task

Creating Dialogs

`NavGetDefaultDialogCreationOptions` (page 1437)
Determines the default attributes or behavior for dialogs.

Choosing Files, Folders and Volumes

`NavCreateChooseFileDialog` (page 1419)
Creates a Choose File dialog, which prompts the user to select a single file as the target of an operation.

`NavCreateChooseFolderDialog` (page 1420)
Creates a Choose Folder dialog, which prompts the user to select a folder as the target of an operation.

`NavCreateChooseVolumeDialog` (page 1422)
Creates a Choose Volume dialog, which prompts the user to select a volume.

`NavCreateChooseObjectDialog` (page 1421)
Creates a Choose Object dialog, which prompts the user to select a file. folder or volume.

`NavCreateGetFileDialog` (page 1423)
Creates an Open dialog, which prompts the user to select a file or files to be opened.

`NavCreateNewFolderDialog` (page 1424)
Creates a New Folder dialog.

Saving Files

`NavCreatePutFileDialog` (page 1426)
Creates a Save dialog, which prompts the user for the name and location of a file to be saved.

`NavCreateAskSaveChangesDialog` (page 1418)
Creates a dialog that asks the user whether to save changes.

`NavCreateAskReviewDocumentsDialog` (page 1417)
Creates a Review Changes dialog, which notifies the user of multiple unsaved documents and gives the user the option to review them.

`NavCreateAskDiscardChangesDialog` (page 1416)
Creates a dialog that asks the user whether to discard changes.

`NavDialogSetSaveFileName` (page 1435)
Specifies the current value of the filename text field in a Save dialog.

`NavDialogSetSaveFileExtensionHidden` (page 1435)
Sets the current state of extension hiding in a Save dialog.

`NavDialogGetSaveFileName` (page 1431)
Obtains the current value of the filename text field in a Save dialog.

`NavDialogGetSaveFileExtensionHidden` (page 1431)
Gets the current state of extension hiding in a Save dialog.

`NavCompleteSave` (page 1415)
Completes a save operation and performs any needed translation on the file.

`NavCreatePreview` (page 1425) Deprecated in Mac OS X v10.5
Creates a document preview in a specified file.

Customizing Dialogs

`NavCustomControl` (page 1429)
Allows your application to control various settings in Navigation Services dialogs.

`NavDialogSetFilterTypeIdentifiers` (page 1434)
Sets UTI filtering criteria for “get file” and “choose file” dialogs.

Running And Disposing of Dialogs

`NavDialogRun` (page 1433)
Displays a previously created dialog.

`NavDialogDispose` (page 1430)
Disposes of a dialog reference.

Obtaining Dialog Information

`NavDialogGetWindow` (page 1433)
Obtains a window reference for a dialog.

`NavDialogGetUserAction` (page 1432)
Reports the user action taken to dismiss a dialog.

`NavDialogGetReply` (page 1430)
Reports the results of a dialog session (unless cancelled or programmatically terminated).

`NavDisposeReply` (page 1436)
Releases the memory allocated for a `NavReplyRecord` structure after your application has finished using the structure.

Translating Files

`NavTranslateFile` (page 1445) Deprecated in Mac OS X v10.5
Provides a means for files opened through Navigation Services to be read from different file formats.

Identifying Navigation Services Availability

`NavServicesAvailable` (page 1444)
Reports whether the Navigation Services library is available on the user's system.

`NavLibraryVersion` (page 1440) Deprecated in Mac OS X v10.5
Reports the currently installed version of the Navigation Services shared library.

Working With Universal Procedure Pointers

`NewNavEventUPP` (page 1446)
Creates a new universal procedure pointer to your application-defined event–handling function.

`NewNavObjectFilterUPP` (page 1447)
Creates a new universal procedure pointer to your application-defined filter function.

`NewNavPreviewUPP` (page 1447)
Creates a new universal procedure pointer to your application-defined preview function.

`DisposeNavEventUPP` (page 1404)
Disposes of a UPP to an application-defined event–handling function.

`DisposeNavObjectFilterUPP` (page 1405)
Disposes of a UPP to an application-defined filter function.

`DisposeNavPreviewUPP` (page 1405)
Disposes of a UPP to an application-defined preview function.

`InvokeNavEventUPP` (page 1406)
Calls your application-defined event–handling function.

`InvokeNavObjectFilterUPP` (page 1406)
Calls your application-defined filter function.

`InvokeNavPreviewUPP` (page 1407)
Calls your application-defined preview function.

Deprecated Functions

`NavAskDiscardChanges` (page 1407) Deprecated in Mac OS X v10.5
Displays an alert box that asks the user whether to discard changes to a particular document.

`NavAskSaveChanges` (page 1408) Deprecated in Mac OS X v10.5
Displays a Save Changes alert box.

`NavChooseFile` (page 1409) Deprecated in Mac OS X v10.5
Creates a simple dialog box that prompts the user to select a file.

`NavChooseFolder` (page 1411) Deprecated in Mac OS X v10.5
Displays a dialog box that prompts the user to choose a folder or volume.

`NavChooseObject` (page 1412) Deprecated in Mac OS X v10.5
Displays a dialog box that prompts the user to choose a file, folder, or volume.

`NavChooseVolume` (page 1413) Deprecated in Mac OS X v10.5
Displays a dialog box that prompts the user to choose a volume.

`NavCustomAskSaveChanges` (page 1428) Deprecated in Mac OS X v10.5
Displays a Save Changes alert box with a custom alert message.

`NavGetDefaultDialogOptions` (page 1437) Deprecated in Mac OS X v10.5
Determines the default attributes or behavior for dialog boxes.

`NavGetFile` (page 1438) Deprecated in Mac OS X v10.5
Displays a dialog box that prompts the user to select a file or files to be opened.

`NavNewFolder` (page 1441) Deprecated in Mac OS X v10.5
Displays a dialog box that prompts the user to create a new folder.

`NavPutFile` (page 1442) Deprecated in Mac OS X v10.5
Displays a Save dialog box.

Unsupported Functions

`NavLoad` (page 1441)
Pre-loads the Navigation Services shared library.

`NavUnload` (page 1446)
Unloads the Navigation Services shared library.

`NavServicesCanRun` (page 1444)

Functions

DisposeNavEventUPP

Disposes of a UPP to an application-defined event–handling function.

```
void DisposeNavEventUPP (
   NavEventUPP userUPP
);
```

Parameters

userUPP

The UPP to dispose of.

Discussion

For more information on event–handling functions, see `NavEventProcPtr` (page 1448).

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X 10.0 and later.

Related Sample Code

QTMetaData

Declared In

`Navigation.h`

DisposeNavObjectFilterUPP

Disposes of a UPP to an application-defined filter function.

```
void DisposeNavObjectFilterUPP (
   NavObjectFilterUPP userUPP
);
```

Parameters

userUPP

The UPP to dispose of.

Discussion

For more information on filter functions, see `NavObjectFilterProcPtr` (page 1449).

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X 10.0 and later.

Declared In

`Navigation.h`

DisposeNavPreviewUPP

Disposes of a UPP to an application-defined preview function.

```
void DisposeNavPreviewUPP (
   NavPreviewUPP userUPP
);
```

Parameters

userUPP

The UPP to dispose of.

Discussion
For more information on preview functions, see `NavPreviewProcPtr` (page 1450).

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.

Declared In
`Navigation.h`

InvokeNavEventUPP

Calls your application-defined event–handling function.

```
void InvokeNavEventUPP (
   NavEventCallbackMessage callBackSelector,
   NavCBRecPtr callBackParms,
   void *callBackUD,
   NavEventUPP userUPP
);
```

Discussion
You should not need to use the function `InvokeNavEventUPP`, as the system calls your event–handling function for you.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.

Declared In
`Navigation.h`

InvokeNavObjectFilterUPP

Calls your application-defined filter function.

```
Boolean InvokeNavObjectFilterUPP (
   AEDesc *theItem,
   void *info,
   void *callBackUD,
   NavFilterModes filterMode,
   NavObjectFilterUPP userUPP
);
```

Discussion
You should not need to use the function `InvokeNavObjectFilterUPP`, as the system calls your filter function for you.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.

Declared In
`Navigation.h`

InvokeNavPreviewUPP

Calls your application-defined preview function.

```
Boolean InvokeNavPreviewUPP (
   NavCBRecPtr callBackParms,
   void *callBackUD,
   NavPreviewUPP userUPP
);
```

Discussion

You should not need to use the function `InvokeNavPreviewUPP`, as the system calls your preview function for you.

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X 10.0 and later.

Declared In

`Navigation.h`

NavAskDiscardChanges

Displays an alert box that asks the user whether to discard changes to a particular document. (Deprecated in Mac OS X v10.5.)

Not recommended

```
OSErr NavAskDiscardChanges (
   NavDialogOptions *dialogOptions,
   NavAskDiscardChangesResult *reply,
   NavEventUPP eventProc,
   void *callBackUD
);
```

Parameters

dialogOptions

A pointer to a structure of type `NavDialogOptions` (page 1463). Before calling `NavAskDiscardChanges`, set up this structure to specify dialog box settings. In this case, the `savedFileName` field is the only one you must supply with a value.

reply

A pointer to a structure of type `NavAskDiscardChanges`. On return, the value describes the user's response to the Discard Changes alert box. For a description of the constants used to represent possible responses, see "Discard Changes Actions" (page 1475).

eventProc

A Universal Procedure Pointer (UPP) to your application-defined event-handling function. You obtain this UPP by calling the function `NewNavEventUPP`. Implementing an event-handling function allows your application to update windows after the user moves or resizes the dialog box. If you pass `NULL` in this parameter, the Discard Changes alert box is not movable. For more information on event-handling functions, see `NavEventProcPtr` (page 1448).

callBackUD

A pointer to a value set by your application. When the `NavAskDiscardChanges` function calls your event-handling function, the `callBackUD` value is passed back to your application.

Return Value
A result code. See "Navigation Services Result Codes" (page 1486).

Discussion
If your application provides a Revert to Saved command, you can use the `NavAskDiscardChanges` function to display a confirmation alert box when a user selects Revert to Saved for a document with unsaved changes. Navigation Services uses the string you supply in the `savedFileName` field of the `NavDialogOptions` structure you passed in the `dialogOptions` parameter to display the alert message, "Discard changes to [savedFilename]?".

Availability
Available in CarbonLib 1.0 and later when Navigation Services 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.

Carbon Porting Notes

Apple recommends you use the function `NavCreateAskDiscardChangesDialog` (page 1416) in order to take advantage of Mac OS X features like Unicode, long filenames and enhanced modality.

Declared In
`Navigation.h`

NavAskSaveChanges

Displays a Save Changes alert box. (Deprecated in Mac OS X v10.5.)

Not recommended

```
OSErr NavAskSaveChanges (
   NavDialogOptions *dialogOptions,
   NavAskSaveChangesAction action,
   NavAskSaveChangesResult *reply,
   NavEventUPP eventProc,
   void *callBackUD
);
```

Parameters

dialogOptions
A pointer to a structure of type `NavDialogOptions` (page 1463). Before calling `NavAskSaveChanges`, set up this structure to specify dialog box settings. When calling `NavAskSaveChanges`, the `clientName` and `savedFileName` fields are the only two fields you must supply with values.

action
A value of type `NavAskSaveChangesAction`. Pass a constant describing the user action that prompted the Save Changes alert box. For a description of the constants, see "Save Changes Requests" (page 1481).

reply
A pointer to a value of type `NavAskSaveChangesResult`. On return, the value describes the user's response to the Save Changes alert box. For a description of the constants used to represent possible responses, see "Save Changes Actions" (page 1481).

eventProc

A Universal Procedure Pointer (UPP) to your application-defined event-handling function. You obtain this UPP by calling the function `NewNavEventUPP`. Implementing an event-handling function allows your application to update windows after the user moves or resizes the dialog box. If you pass `NULL` in this parameter, the Save Changes alert box is not movable. For more information on event-handling functions, see `NavEventProcPtr` (page 1448).

callBackUD

A pointer to a value set by your application. When the `NavAskSaveChanges` function calls your event-handling function, the `callBackUD` value is passed back to your application.

Return Value
A result code. See "Navigation Services Result Codes" (page 1486).

Discussion
This function is useful when your application needs to display an alert when the user attempts to close a document or an application with unsaved changes.

Availability
Available in CarbonLib 1.0 and later when Navigation Services 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.

Carbon Porting Notes

Apple recommends you use the function `NavCreateAskSaveChangesDialog` (page 1418) in order to take advantage of Mac OS X features like Unicode, long filenames and enhanced modality.

Declared In
`Navigation.h`

NavChooseFile

Creates a simple dialog box that prompts the user to select a file. (Deprecated in Mac OS X v10.5.)

Not recommended

```
OSErr NavChooseFile (
   AEDesc *defaultLocation,
   NavReplyRecord *reply,
   NavDialogOptions *dialogOptions,
   NavEventUPP eventProc,
   NavPreviewUPP previewProc,
   NavObjectFilterUPP filterProc,
   NavTypeListHandle typeList,
   void *callBackUD
);
```

Parameters

defaultLocation

A pointer to an Apple event descriptor structure (`AEDesc`). Before calling `NavChooseFile`, you can set up a structure of `AEDesc` type `'typeFSS'` to specify a default location to be viewed. If you pass `NULL` in this parameter, Navigation Services displays the last location visited during a call to the `NavChooseFile` function. If the file system specification in the `AEDesc` structure does not describe a directory or volume, Navigation Services uses the desktop as the default location.

reply

A pointer to a structure of type `NavReplyRecord` (page 1458). Upon return, Navigation Services uses this structure to provide data to your application about the results of your `NavChooseFile` call.

dialogOptions

A pointer to a structure of type `NavDialogOptions` (page 1463). Before calling `NavChooseFile`, you can set up this structure to specify dialog box settings. If you pass `NULL` in this parameter, Navigation Services uses the defaults for all options. See "Dialog Configuration Options" (page 1471) for a description of the default settings.

eventProc

A Universal Procedure Pointer (UPP) to your application-defined event-handling function. You obtain this UPP by calling the function `NewNavEventUPP`. Implementing an event-handling function allows your application to update windows after the user moves or resizes the dialog box. If you pass `NULL` in this parameter, the Choose a File dialog box is not movable or resizable. For more information on event-handling functions, see `NavEventProcPtr` (page 1448).

previewProc

A Universal Procedure Pointer (UPP) to your application-defined preview function. Obtain this UPP by calling the function `NewNavPreviewUPP`. A preview function allows your application to draw previews or to override Navigation Services previews. For more information on preview functions, see `NavPreviewProcPtr` (page 1450).

filterProc

A Universal Procedure Pointer (UPP) to your application-defined filter function. Obtain this UPP by calling the function `NewNavObjectFilterUPP`. An application-defined filter function determines if a volume, directory, or file should be displayed in the browser list or pop-up menus. For more information on filter functions, see `NavObjectFilterProcPtr` (page 1449).

typeList

A handle to a structure of type `NavTypeList` (page 1460). Before calling `NavChooseFile`, you can set up this structure to declare file types that your application can open.

callBackUD

A pointer to a value set by your application. When the `NavChooseFile` function calls your event-handling function, the `callBackUD` value is passed back to your application.

Return Value
A result code. See "Navigation Services Result Codes" (page 1486).

Discussion
This function allows the user to choose a single file, such as a preferences file, for an action other than opening. The `NavChooseFile` function is similar to `NavGetFile` (page 1438), but is limited to selecting a single file.

The dialog box displayed by the `NavChooseFile` function does not display a Show menu. If you wish to control the files displayed by the browser list or the pop-up menus, you must specify a list of file types in the `typeList` parameter or specify a filter function in the `filterProc` parameter. If you specify a list of file types in the `typeList` parameter, the `NavChooseFile` function ignores the `signature` field of the `NavTypeList` structure. This means that all files of the types specified in the list of file types will be displayed, regardless of their application signature.

Availability
Available in CarbonLib 1.0 and later when Navigation Services 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.

Carbon Porting Notes

Apple recommends you use the function `NavCreateChooseFileDialog` (page 1419) in order to take advantage of Mac OS X features like Unicode, long filenames and enhanced modality.

Declared In

`Navigation.h`

NavChooseFolder

Displays a dialog box that prompts the user to choose a folder or volume. (Deprecated in Mac OS X v10.5.)

Not recommended

```
OSErr NavChooseFolder (
   AEDesc *defaultLocation,
   NavReplyRecord *reply,
   NavDialogOptions *dialogOptions,
   NavEventUPP eventProc,
   NavObjectFilterUPP filterProc,
   void *callBackUD
);
```

Parameters

defaultLocation

A pointer to an Apple event descriptor structure (`AEDesc`). Before calling `NavChooseFolder`, you can set up a structure of `AEDesc` type `'typeFSS'` to specify a default location to be viewed. If you pass `NULL` in this parameter, Navigation Services displays the last location visited during a call to the `NavChooseFolder` function. If the file system specification in the `AEDesc` structure does not describe a directory or volume, Navigation Services uses the desktop as the default location.

reply

A pointer to a structure of type `NavReplyRecord` (page 1458). Upon return, Navigation Services uses this structure to provide data to your application about the results of your `NavChooseFolder` call.

dialogOptions

A pointer to a structure of type `NavDialogOptions` (page 1463). Before calling `NavChooseFolder`, set up this structure to specify dialog box settings. If you pass `NULL` in this parameter, Navigation Services uses the defaults for all options. See "Dialog Configuration Options" (page 1471) for a description of the default settings.

eventProc

A Universal Procedure Pointer (UPP) to your application-defined event-handling function. You obtain this UPP by calling the function `NewNavEventUPP`. Implementing an event-handling function allows your application to update windows after the user moves or resizes the dialog box. If you pass `NULL` in this parameter, the dialog box is not movable or resizable. For more information on event-handling functions, see `NavEventProcPtr` (page 1448).

filterProc

A Universal Procedure Pointer (UPP) to your application-defined filter function. Obtain this UPP by calling the function `NewNavObjectFilterUPP`. An application-defined filter function determines if a volume, directory, or file should be displayed in the browser list or pop-up menus. For more information on filter functions, see `NavObjectFilterProcPtr` (page 1449).

callBackUD

A pointer to a value set by your application. When the `NavChooseFolder` function calls your event-handling function, the `callBackUD` value is passed back to your application.

Return Value
A result code. See "Navigation Services Result Codes" (page 1486).

Discussion
This function provides a way for your application to prompt the user to select a folder or volume. This might be useful if you need to install application files, for example.

Availability
Available in CarbonLib 1.0 and later when Navigation Services 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.

Carbon Porting Notes

Apple recommends you use the function `NavCreateChooseFolderDialog` (page 1420) in order to take advantage of Mac OS X features like Unicode, long filenames and enhanced modality.

Declared In
`Navigation.h`

NavChooseObject

Displays a dialog box that prompts the user to choose a file, folder, or volume. (Deprecated in Mac OS X v10.5.)

Not recommended

```
OSErr NavChooseObject (
   AEDesc *defaultLocation,
   NavReplyRecord *reply,
   NavDialogOptions *dialogOptions,
   NavEventUPP eventProc,
   NavObjectFilterUPP filterProc,
   void *callBackUD
);
```

Parameters

defaultLocation

A pointer to an Apple event descriptor structure (`AEDesc`). Before calling `NavChooseObject`, you can set up a structure of `AEDesc` type `'typeFSS'` to specify a default location to be viewed. If you pass `NULL` in this parameter, Navigation Services displays the last location visited during a call to the `NavChooseObject` function. If the file system specification in the `AEDesc` structure does not describe a directory or volume, Navigation Services uses the desktop as the default location.

reply

A pointer to a structure of type `NavReplyRecord` (page 1458). Upon return, Navigation Services uses this structure to provide data to your application about the results of your `NavChooseObject` call.

dialogOptions

A pointer to a structure of type `NavDialogOptions` (page 1463). Before calling `NavChooseObject`, set up this structure to specify dialog box settings. If you do not provide this structure, Navigation Services uses the defaults for all options. See "Dialog Configuration Options" (page 1471) for a description of the default settings.

eventProc

A Universal Procedure Pointer (UPP) to your application-defined event-handling function. You obtain this UPP by calling the function `NewNavEventUPP`. Implementing an event-handling function allows your application to update windows after the user moves or resizes the dialog box. If you pass `NULL` in this parameter, the dialog box is not movable or resizable. For more information on event-handling functions, see `NavEventProcPtr` (page 1448).

filterProc

A Universal Procedure Pointer (UPP) to your application-defined filter function. Obtain this UPP by calling the function `NewNavObjectFilterUPP`. An application-defined filter function determines if a volume, directory, or file should be displayed in the browser list or pop-up menus. For more information on filter functions, see `NavObjectFilterProcPtr` (page 1449).

callBackUD

A pointer to a value set by your application. When the `NavChooseObject` function calls your event-handling function, the `callBackUD` value is passed back to your application.

Return Value
A result code. See "Navigation Services Result Codes" (page 1486).

Discussion
This function is useful when you need to display a dialog box that prompts the user to choose a file object that might be a file, folder, or volume. If you want the user to choose a specific type of file object, you should use the function designed for that type of object; to select a file, for example, use the function `NavChooseFile` (page 1409).

Availability
Available in CarbonLib 1.0 and later when Navigation Services 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.

Carbon Porting Notes

Apple recommends you use the function `NavCreateChooseObjectDialog` (page 1421) in order to take advantage of Mac OS X features like Unicode, long filenames and enhanced modality.

Declared In
`Navigation.h`

NavChooseVolume

Displays a dialog box that prompts the user to choose a volume. (Deprecated in Mac OS X v10.5.)

Not recommended

```
OSErr NavChooseVolume (
   AEDesc *defaultSelection,
   NavReplyRecord *reply,
   NavDialogOptions *dialogOptions,
   NavEventUPP eventProc,
   NavObjectFilterUPP filterProc,
   void *callBackUD
);
```

Parameters

defaultSelection

A pointer to an Apple event descriptor structure (`AEDesc`). Before calling `NavChooseVolume`, you can set up a structure of `AEDesc` type `'typeFSS'` to specify a default location to be viewed. If you pass `NULL` in this parameter, Navigation Services displays the last location visited during a call to the `NavChooseVolume` function. If the file system specification in the `AEDesc` structure does not describe a directory or volume, Navigation Services uses the desktop as the default location.

reply

A pointer to a structure of type `NavReplyRecord` (page 1458). Upon return, Navigation Services uses this structure to provide data to your application about the results of your `NavChooseVolume` call.

dialogOptions

A pointer to a structure of type `NavDialogOptions` (page 1463). Before calling, set up this structure to specify dialog box settings. If you pass `NULL` in this parameter, Navigation Services uses the defaults for all options. See "Dialog Configuration Options" (page 1471) for a description of the default settings.

eventProc

A Universal Procedure Pointer (UPP) to your application-defined event-handling function. You obtain this UPP by calling the function `NewNavEventUPP`. Implementing an event-handling function allows your application to update windows after the user moves or resizes the dialog box. If you pass `NULL` in this parameter, the Choose a Volume dialog box is not movable or resizable. For more information on event-handling functions, see `NavEventProcPtr` (page 1448).

filterProc

A Universal Procedure Pointer (UPP) to your application-defined filter function. Obtain this UPP by calling the function `NewNavObjectFilterUPP`. An application-defined filter function determines if a volume, directory, or file should be displayed in the browser list or pop-up menus. For more information on filter functions, see `NavObjectFilterProcPtr` (page 1449).

callBackUD

A pointer to a value set by your application. When the `NavChooseVolume` function calls your event-handling function, the `callBackUD` value is passed back to your application.

Return Value

A result code. See "Navigation Services Result Codes" (page 1486).

Discussion

This function provides a way for your application to prompt the user to select a volume. This might be useful for a disk repair utility, for example.

Availability

Available in CarbonLib 1.0 and later when Navigation Services 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Carbon Porting Notes

Apple recommends you use the function `NavCreateChooseVolumeDialog` (page 1422) in order to take advantage of Mac OS X features like Unicode, long filenames and enhanced modality.

Declared In

`Navigation.h`

NavCompleteSave

Completes a save operation and performs any needed translation on the file.

```
OSErr NavCompleteSave (
   const NavReplyRecord *reply,
   NavTranslationOptions howToTranslate
);
```

Parameters

reply

A pointer to a structure of type `NavReplyRecord` (page 1458). Upon return, Navigation Services uses this structure to provide data to your application about the results of your `NavCompleteSave` call.

howToTranslate

A pointer to a structure of type `NavTranslationOptions`. Pass one of two values to specify how to perform any needed translation. For a description of the constants you can use to represent these values, see "Translation Options" (page 1483). Translating in-place causes the source file to be replaced by the translation. Translating to a copy results in a file name followed by the string "(converted)" to avoid unwanted replacement. If you call the `NavCompleteSave` function in response to a Save a Copy command, you should pass the `kNavTranslateInPlace` constant in this parameter.

Return Value

A result code. See "Navigation Services Result Codes" (page 1486). Since this function performs any needed translation, it may return a translation error.

Discussion

You should always call `NavCompleteSave` to complete any file saving operation performed with the `NavCreatePutFileDialog` function. `NavCompleteSave` performs any needed translation, so you do not have to use the function `NavTranslateFile` (page 1445) when saving. If you wish to turn off automatic translation, set to `false` the value of the `translationNeeded` field of the `NavReplyRecord` structure you pass in the `reply` parameter of the `NavPutFile` function. If you turn off automatic translation, your application is responsible for any required translation.

Availability

Available in CarbonLib 1.0 and later when Navigation Services 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Related Sample Code

CarbonSketch

QTCarbonShell

Declared In

`Navigation.h`

NavCreateAskDiscardChangesDialog

Creates a dialog that asks the user whether to discard changes.

```
OSStatus NavCreateAskDiscardChangesDialog (
   const NavDialogCreationOptions *inOptions,
   NavEventUPP inEventProc,
   void *inClientData,
   NavDialogRef *outDialog
);
```

Parameters

inOptions

A pointer to a structure specifying options that control the appearance and behavior of the dialog. You must supply a string in the `saveFileName` field of this structure; otherwise the function returns `paramErr`.

inEventProc

A Universal Procedure Pointer (UPP) to your application's event-handling function. You are strongly advised to create and register an event-handling function, as described in `NavEventProcPtr` (page 1448). You must have an event-handling function in order to create modeless or window-modal (sheet) dialogs. Specify `NULL` in this parameter if you do not implement an event-handling function.

inClientData

A pointer to an application-defined value that is passed back to all callback functions. You can use this value to provide context information, for example. You may pass `NULL` in this parameter.

outDialog

On successful creation of a Discard Changes dialog, A pointer to a Navigation Services dialog reference that you can pass to the `NavDialogRun` (page 1433) function

Return Value

A result code.

Discussion

This function creates a dialog that gives the user the option of discarding unsaved changes to a file or cancelling the operation. This dialog is most commonly used when the user wants to revert to the last saved version of a document.

Once you have successfully created the Discard Changes dialog, you display it by calling the `NavDialogRun` (page 1433) function. After the user interacts with the dialog, you can obtain information about the dialog session by calling the `NavDialogGetReply` (page 1430) function. When you are finished with the dialog, dispose of it by calling the `NavDialogDispose` (page 1430) function.

This function replaces the `NavAskDiscardChanges` function and adds support for Unicode and new window modalities.

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X 10.0 and later.

Declared In

`Navigation.h`

NavCreateAskReviewDocumentsDialog

Creates a Review Changes dialog, which notifies the user of multiple unsaved documents and gives the user the option to review them.

```
OSStatus NavCreateAskReviewDocumentsDialog (
   const NavDialogCreationOptions *inOptions,
   ItemCount inDocumentCount,
   NavEventUPP inEventProc,
   void *inClientData,
   NavDialogRef *outDialog
);
```

Parameters

inOptions

A pointer to a structure specifying options that control the appearance and behavior of the dialog.

inDocumentCount

The number of documents needing review. This number appears in the text presented to the user. If the total number of unsaved documents is unknown, specify 0; Navigation Services uses a general message. You should not specify 1; this alert should be used only when more than one document needs review. For more information, see *Inside Mac OS X: Aqua Human Interface Guidelines*.

inEventProc

A universal procedure pointer (UPP) to an application-defined event-handling function. You are strongly advised to create and register an event-handling function, as described in `NavEventProcPtr` (page 1448). You must have an event-handling function in order to create modeless or window-modal (sheet) dialogs. Specify `NULL` in this parameter if you do not implement an event-handling function.

inClientData

A pointer to an application-defined value that is passed back to all callback functions.

outDialog

Upon successful completion, a reference to the created dialog.

Return Value

A result code. See "Navigation Services Result Codes" (page 1486).

Discussion

The Review Changes dialog tells the user how many unsaved documents there are and asks the user to choose one of the following options:

- review the unsaved documents
- don't save any documents
- cancel

Use of this dialog is appropriate when an application is quitting and there is more than one unsaved document. It is supported only on Mac OS X; prior to Mac O X, this dialog is not part of the application quit sequence.

Upon successful creation, the dialog is not visible; to present and run the dialog, call the `NavDialogRun` (page 1433) function. After the dialog is complete, dispose of it with the `NavDialogDispose` function. Upon dismissal of the dialog, the user's action is set to one of the following actions: `kNavUserActionReviewDocuments`, `kNavUserActionDiscardDocuments`, or `kNavUserActionCancel`. You can obtain this reply by calling the `NavDialogGetReply` (page 1430).

Availability
Not available in CarbonLib.
Available in Mac OS X 10.1 and later.

Related Sample Code
QTCarbonShell

Declared In
`Navigation.h`

NavCreateAskSaveChangesDialog

Creates a dialog that asks the user whether to save changes.

```
OSStatus NavCreateAskSaveChangesDialog (
   const NavDialogCreationOptions *inOptions,
   NavAskSaveChangesAction inAction,
   NavEventUPP inEventProc,
   void *inClientData,
   NavDialogRef *outDialog
);
```

Parameters

inOptions
A pointer to a structure specifying options that control the appearance and behavior of the dialog.

inAction
A value indicating whether the user is closing a document or quitting the application and thereby determines the message displayed to the user. To provide a customized message for the dialog, specify a non-`NULL` value in the `message` field of the structure provided in the *inOptions* parameter.

inEventProc
A Universal Procedure Pointer (UPP) to your application's event-handling function. You are strongly advised to create and register an event-handling function, as described in `NavEventProcPtr` (page 1448). You must have an event-handling function in order to create modeless or window-modal (sheet) dialogs. Specify `NULL` in this parameter if you do not implement an event-handling function.

inClientData
A pointer to an application-defined value that is passed back to all callback functions. You can use this value to provide context information, for example. You may pass `NULL` in this parameter.

outDialog
On successful creation of the Save Changes dialog, a pointer to a Navigation Services dialog reference that you can pass to the `NavDialogRun` (page 1433) function.

Return Value
A result code. See "Navigation Services Result Codes" (page 1486). A result code.

Discussion
This function creates a Save Changes dialog, which your application should display when the user attempts to close a document or quit the application with unsaved changes. The Save Changes dialog allows the user to choose one of the following options:

- save the changes
- discard the unsaved changes

- cancel the operation

Once you have successfully created the Save Changes dialog, you display it by calling the `NavDialogRun` (page 1433) function. After the user interacts with the dialog, you can obtain information about the dialog session by calling the `NavDialogGetReply` (page 1430) function. When you are finished with the dialog, dispose of it by calling the `NavDialogDispose` (page 1430) function.

If there is more than one document with unsaved changes when the user attempts to quit your application, you should display a Review Changes dialog instead. You can create a Review Changes dialog with the `NavCreateAskReviewDocumentsDialog` (page 1417) function.

This function replaces the `NavAskSaveChanges` function and adds support for Unicode and new window modalities.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X 10.0 and later.

Related Sample Code
QTCarbonShell

Declared In
`Navigation.h`

NavCreateChooseFileDialog

Creates a Choose File dialog, which prompts the user to select a single file as the target of an operation.

```
OSStatus NavCreateChooseFileDialog (
   const NavDialogCreationOptions *inOptions,
   NavTypeListHandle inTypeList,
   NavEventUPP inEventProc,
   NavPreviewUPP inPreviewProc,
   NavObjectFilterUPP inFilterProc,
   void *inClientData,
   NavDialogRef *outDialog
);
```

Parameters

inOptions
A pointer to a structure specifying options that control the appearance and behavior of the dialog.

inTypeList
A structure specifying a creator signature and a list of file types to show in the Choose File dialog.

inEventProc
A Universal Procedure Pointer (UPP) to your application's event-handling function. You are strongly advised to create and register an event-handling function, as described in `NavEventProcPtr` (page 1448). You must have an event-handling function in order to create modeless or window-modal (sheet) dialogs. Specify `NULL` in this parameter if you do not implement an event-handling function.

inPreviewProc
A Universal Procedure Pointer (UPP) to your application's preview function. You may specify `NULL` if you don't need to register a preview function. For more information on preview functions, see `NavPreviewProcPtr` (page 1450).

inFilterProc

A Universal Procedure Pointer (UPP) to your application's filter function. You may specify `NULL` if you don't need to register a filter function. For more information on filter functions, see `NavObjectFilterProcPtr` (page 1449).

inClientData

A pointer to an application-defined value that is passed back to all callback functions. You can use this value to provide context information, for example. You may pass `NULL` in this parameter.

outDialog

On successful creation of a Choose File dialog, A pointer to a Navigation Services dialog reference that you can pass to the `NavDialogRun` (page 1433) function.

Return Value
A result code. See "Navigation Services Result Codes" (page 1486). A result code.

Discussion
Once you have successfully created the Choose File dialog, you display it by calling the `NavDialogRun` (page 1433) function. After the user interacts with the dialog, you can obtain information about the dialog session by calling the `NavDialogGetReply` (page 1430) function. When you are finished with the dialog, you should dispose of it by calling the `NavDialogDispose` (page 1430) function.

This function replaces the `NavChooseFile` function and adds support for Unicode and new window modalities.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X 10.0 and later.

Related Sample Code
QTCarbonShell

Declared In
`Navigation.h`

NavCreateChooseFolderDialog

Creates a Choose Folder dialog, which prompts the user to select a folder as the target of an operation.

```
OSStatus NavCreateChooseFolderDialog (
   const NavDialogCreationOptions *inOptions,
   NavEventUPP inEventProc,
   NavObjectFilterUPP inFilterProc,
   void *inClientData,
   NavDialogRef *outDialog
);
```

Parameters

inOptions

A pointer to a structure specifying options that control the appearance and behavior of the dialog.

inEventProc

A Universal Procedure Pointer (UPP) to your application's event-handling function. You are strongly advised to create and register an event-handling function, as described in `NavEventProcPtr` (page 1448). You must have an event-handling function in order to create modeless or window-modal (sheet) dialogs. Specify `NULL` in this parameter if you do not implement an event-handling function.

inFilterProc

A Universal Procedure Pointer (UPP) to your application's filter function. You may specify `NULL` if you don't need to register a filter function. For more information on filter functions, see `NavObjectFilterProcPtr` (page 1449).

inClientData

A pointer to an application-defined value that is passed back to all callback functions. You can use this value to provide context information, for example. You may pass `NULL` in this parameter.

outDialog

On successful creation of a Choose Folder dialog, A pointer to a Navigation Services dialog reference that you can pass to the `NavDialogRun` (page 1433) function.

Return Value

A result code. See "Navigation Services Result Codes" (page 1486). A result code.

Discussion

Once you have successfully created the Choose Folder dialog, you display it by calling the `NavDialogRun` (page 1433) function. After the user interacts with the dialog, you can obtain information about the dialog session by calling the `NavDialogGetReply` (page 1430) function. When you are finished with the dialog, you should dispose of it by calling the `NavDialogDispose` (page 1430) function.

This function replaces the `NavChooseFolder` function and adds support for Unicode and new window modalities.

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X 10.0 and later.

Declared In

`Navigation.h`

NavCreateChooseObjectDialog

Creates a Choose Object dialog, which prompts the user to select a file. folder or volume.

```
OSStatus NavCreateChooseObjectDialog (
   const NavDialogCreationOptions *inOptions,
   NavEventUPP inEventProc,
   NavPreviewUPP inPreviewProc,
   NavObjectFilterUPP inFilterProc,
   void *inClientData,
   NavDialogRef *outDialog
);
```

Parameters

inOptions

A pointer to a structure specifying options that control the appearance and behavior of the dialog.

inEventProc

A Universal Procedure Pointer (UPP) to your application's event-handling function. You are strongly advised to create and register an event-handling function, as described in `NavEventProcPtr` (page 1448). You must have an event-handling function in order to create modeless or window-modal (sheet) dialogs. Specify `NULL` in this parameter if you do not implement an event-handling function.

inPreviewProc

A Universal Procedure Pointer (UPP) to your application's preview function. You may specify `NULL` if you don't need to register a preview function. For more information on preview functions, see `NavPreviewProcPtr` (page 1450).

inFilterProc

A Universal Procedure Pointer (UPP) to your application's filter function. You may specify `NULL` if you don't need to register a filter function. For more information on filter functions, see `NavObjectFilterProcPtr` (page 1449).

inClientData

A pointer to an application-defined value that is passed back to all callback functions. You can use this value to provide context information, for example. You may pass `NULL` in this parameter.

outDialog

On successful creation of a Choose Object dialog, a pointer to a Navigation Services dialog reference that you can pass to the `NavDialogRun` (page 1433) function.

Return Value
A result code. See "Navigation Services Result Codes" (page 1486). A result code.

Discussion
Once you have successfully created the Choose Object dialog, you display it by calling the `NavDialogRun` (page 1433) function. After the user interacts with the dialog, you can obtain information about the dialog session by calling the `NavDialogGetReply` (page 1430) function. When you are finished with the dialog, you should dispose of it by calling the `NavDialogDispose` (page 1430) function.

This function replaces the `NavChooseObject` function and adds support for Unicode and new window modalities.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X 10.0 and later.

Declared In
`Navigation.h`

NavCreateChooseVolumeDialog

Creates a Choose Volume dialog, which prompts the user to select a volume.

```
OSStatus NavCreateChooseVolumeDialog (
   const NavDialogCreationOptions *inOptions,
   NavEventUPP inEventProc,
   NavObjectFilterUPP inFilterProc,
   void *inClientData,
   NavDialogRef *outDialog
);
```

Parameters

inOptions

A pointer to a structure specifying options that control the appearance and behavior of the dialog.

inEventProc

A Universal Procedure Pointer (UPP) to your application's event-handling function. You are strongly advised to create and register an event-handling function, as described in `NavEventProcPtr` (page 1448). You must have an event-handling function in order to create modeless or window-modal (sheet) dialogs. Specify `NULL` in this parameter if you do not implement an event-handling function.

inFilterProc

A Universal Procedure Pointer (UPP) to your application's filter function. You may specify `NULL` if you don't need to register a filter function. For more information on filter functions, see `NavObjectFilterProcPtr` (page 1449).

inClientData

A pointer to an application-defined value that is passed back to all callback functions. You can use this value to provide context information, for example. You may pass `NULL` in this parameter.

outDialog

On successful creation of a Choose Volume dialog, A pointer to a Navigation Services dialog reference that you can pass to the `NavDialogRun` (page 1433) function.

Return Value
A result code. See "Navigation Services Result Codes" (page 1486). A result code.

Discussion
Once you have successfully created the Choose Volume dialog, you display it by calling the `NavDialogRun` (page 1433) function. After the user interacts with the dialog, you can obtain information about the dialog session by calling the `NavDialogGetReply` (page 1430) function. When you are finished with the dialog, you should dispose of it by calling the `NavDialogDispose` (page 1430) function.

This function replaces the `NavChooseVolume` function and adds support for Unicode and new window modalities.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X 10.0 and later.

Declared In
`Navigation.h`

NavCreateGetFileDialog

Creates an Open dialog, which prompts the user to select a file or files to be opened.

```
OSStatus NavCreateGetFileDialog (
   const NavDialogCreationOptions *inOptions,
   NavTypeListHandle inTypeList,
   NavEventUPP inEventProc,
   NavPreviewUPP inPreviewProc,
   NavObjectFilterUPP inFilterProc,
   void *inClientData,
   NavDialogRef *outDialog
);
```

Parameters

inOptions

A pointer to a structure specifying options that control the appearance and behavior of the dialog.

inTypeList
A structure specifying an application signature and a list of file types to show in the Open dialog.

inEventProc
A Universal Procedure Pointer (UPP) to your application's event-handling function. You are strongly advised to create and register an event-handling function, as described in `NavEventProcPtr` (page 1448). You must have an event-handling function in order to create modeless or window-modal (sheet) dialogs. Specify `NULL` in this parameter if you do not implement an event-handling function.

inPreviewProc
A Universal Procedure Pointer (UPP) to your application's preview function. You may specify `NULL` if you don't need to register a preview function. For more information on creating a preview function, see `NavPreviewProcPtr` (page 1450).

inFilterProc
A Universal Procedure Pointer (UPP) to your application's filter function. You may specify `NULL` if you don't need to register a filter function. For more information on creating a filter function, see `NavObjectFilterProcPtr` (page 1449).

inClientData
A pointer to an application-defined value that is passed back to all callback functions. You can use this value to provide context information, for example. You may pass `NULL` in this parameter.

outDialog
On successful creation of an Open dialog instance, this value specifies a Navigation Services dialog reference that you can pass to the `NavDialogRun` (page 1433) function.

Return Value
A result code. See "Navigation Services Result Codes" (page 1486). A result code.

Discussion
Once you have successfully created the Open dialog, you display it by calling the `NavDialogRun` (page 1433) function. After the user interacts with the dialog, you can obtain information about the dialog session by calling the `NavDialogGetReply` (page 1430) function. When you are finished with the dialog, you should dispose of it by calling the `NavDialogDispose` (page 1430) function.

This function replaces the `NavGetFile` function and adds support for Unicode and new window modalities.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X 10.0 and later.

Declared In
`Navigation.h`

NavCreateNewFolderDialog

Creates a New Folder dialog.

```
OSStatus NavCreateNewFolderDialog (
   const NavDialogCreationOptions *inOptions,
   NavEventUPP inEventProc,
   void *inClientData,
   NavDialogRef *outDialog
);
```

Parameters

inOptions

A pointer to a structure specifying options that control the appearance and behavior of the dialog.

inEventProc

A Universal Procedure Pointer (UPP) to your application's event-handling function. You are strongly advised to create and register an event-handling function, as described in `NavEventProcPtr` (page 1448). You must have an event-handling function in order to create modeless or window-modal (sheet) dialogs. Specify `NULL` in this parameter if you do not implement an event-handling function.

inClientData

A pointer to an application-defined value that is passed back to all callback functions. You can use this value to provide context information, for example. You may pass `NULL` in this parameter.

outDialog

On successful creation of a New Folder dialog, a pointer to a Navigation Services dialog reference that you can pass to the `NavDialogRun` (page 1433) function.

Return Value

A result code. See "Navigation Services Result Codes" (page 1486). A result code.

Discussion

Once you have successfully created the New Folder dialog, you display it by calling the `NavDialogRun` (page 1433) function. After the user interacts with the dialog, you can obtain information about the dialog session by calling the `NavDialogGetReply` (page 1430) function. When you are finished with the dialog, you should dispose of it by calling the `NavDialogDispose` (page 1430) function

Use the New Folder dialog to allow the user to create a new folder. Navigation Services creates the folder as specified by the user and returns a reference to the folder in the `selection` field of the reply record.

This function replaces the `NavNewFolder` function and adds support for Unicode and new window modalities.

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X 10.0 and later.

Declared In

`Navigation.h`

NavCreatePreview

Creates a document preview in a specified file. (Deprecated in Mac OS X v10.5.)

```
OSErr NavCreatePreview (
   AEDesc *theObject,
   OSType previewDataType,
   const void *previewData,
   Size previewDataSize
);
```

Parameters

theObject

A pointer to an Apple Event Descriptor (`AEDesc`) structure specifying the file in which to create the preview.

previewDataType

A four character code specifying the type of preview data to create. If you pass `NULL` in this parameter, Navigation Services creates a preview of type '`PICT`'.

previewData

A pointer to a buffer holding preview data. If you pass `NULL` in this parameter, Navigation Services provides its own data.

previewDataSize

A value specifying the size, in bytes, of the preview data you are providing. If you pass `NULL` in this parameter, Navigation Services provides its own data.

Return Value

A result code. See "Navigation Services Result Codes" (page 1486).

Discussion

This function creates a preview of the specified file and stores the data in an appropriate resource. If you call this function without passing in preview data, as in the following snippet, Navigation Services obtains and creates the preview automatically:

```
NavCreatePreview(theObject,0,NULL,0)
```

If the specified file is image-based ('`PICT`', '`JPEG`', etc.), Navigation Services creates a thumbnail custom icon for the file. Navigation Services does not create a custom icon if you pass in your own preview data.

Availability

Available in CarbonLib 1.0 and later when Navigation Services 2.0 or later is present.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Navigation.h`

NavCreatePutFileDialog

Creates a Save dialog, which prompts the user for the name and location of a file to be saved.

```
OSStatus NavCreatePutFileDialog (
   const NavDialogCreationOptions *inOptions,
   OSType inFileType,
   OSType inFileCreator,
   NavEventUPP inEventProc,
   void *inClientData,
   NavDialogRef *outDialog
);
```

Parameters

inOptions

A pointer to a structure specifying options that control the appearance and behavior of the dialog.

inFileType

A four-character code specifying a file type for the file to be saved.

inFileCreator

A four-character code specifying a creator signature for the file to be saved. If you want to change or remove the top default item in the Format menu, pass `kNavGenericSignature`.

inEventProc

A Universal Procedure Pointer (UPP) to your application's event-handling function. You are strongly advised to create and register an event-handling function, as described in `NavEventProcPtr` (page 1448). You must have an event-handling function in order to create modeless or window-modal (sheet) dialogs. Specify `NULL` in this parameter if you do not implement an event-handling function.

inClientData

A pointer to an application-defined value that is passed back to all callback functions. You can use this value to provide context information, for example. You may pass `NULL` in this parameter.

outDialog

On successful creation of a Save dialog, a pointer to a Navigation Services dialog reference that you can pass to the `NavDialogRun` (page 1433) function.

Return Value

A result code. See "Navigation Services Result Codes" (page 1486). A result code.

Discussion

Once you have successfully created the Save dialog, you display it by calling the `NavDialogRun` (page 1433) function. After the user interacts with the dialog, you can obtain information about the dialog session by calling the `NavDialogGetReply` (page 1430) function. When you are finished with the dialog, you should dispose of it by calling the `NavDialogDispose` (page 1430) function.

This function replaces the `NavPutFile` function and adds support for Unicode and new window modalities.

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X 10.0 and later.

Related Sample Code

CarbonSketch

QTCarbonShell

Declared In

`Navigation.h`

NavCustomAskSaveChanges

Displays a Save Changes alert box with a custom alert message. (Deprecated in Mac OS X v10.5.)

Not recommended

```
OSErr NavCustomAskSaveChanges (
   NavDialogOptions *dialogOptions,
   NavAskSaveChangesResult *reply,
   NavEventUPP eventProc,
   void *callBackUD
);
```

Parameters

dialogOptions

A pointer to a structure of type `NavDialogOptions` (page 1463). Before calling `NavCustomAskSaveChanges`, set up this structure to specify dialog box settings. When calling `NavCustomAskSaveChanges`, the `message` field is the only field you must supply with a value.

reply

A pointer to a value of type `NavAskSaveChangesResult`. On return, the value describes the user's response to the Save Changes alert box. For a description of the constants used to represent possible responses, see "Save Changes Actions" (page 1481).

eventProc

A Universal Procedure Pointer (UPP) to your application-defined event-handling function. You obtain this UPP by calling the function `NewNavEventUPP`. Implementing an event-handling function allows your application to update windows after the user moves or resizes the dialog box. If you pass `NULL` in this parameter, the Save Changes alert box is not movable. For more information on event-handling functions, see `NavEventProcPtr` (page 1448).

callBackUD

A pointer to a value set by your application. When the `NavCustomAskSaveChanges` function calls your event-handling function, the `callBackUD` value is passed back to your application.

Return Value

A result code. See "Navigation Services Result Codes" (page 1486).

Discussion

This function is similar to the function `NavAskSaveChanges` (page 1408), except that you provide a custom alert message. This function is useful when you need to post a Save Changes alert box at times other than quitting or closing a file. Your application can display this alert box if a specified time interval has passed since the user last saved changes, for example.

Availability

Available in CarbonLib 1.0 and later when Navigation Services 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Carbon Porting Notes

Apple recommends you use the function `NavCreateAskSaveChangesDialog` in order to take advantage of Mac OS X features like Unicode, long filenames and enhanced modality. In order to provide a customized alert message, pass a non-null message string in the `NavDialogCreationOptions` structure passed to `NavCreateAskSaveChangesDialog`.

Declared In

`Navigation.h`

NavCustomControl

Allows your application to control various settings in Navigation Services dialogs.

```
OSErr NavCustomControl (
   NavDialogRef dialog,
   NavCustomControlMessage selector,
   void *parms
);
```

Parameters

dialog

A Navigation Services dialog reference. You can obtain this value from the `context` field of the structure of type `NavCBRec` (page 1451) specified in the `callBackParms` parameter of your event-handling function.

selector

A value of type `NavCustomControlMessage`. Pass one or more of the constants representing the possible values used to control various aspects of the active dialog. For a description of these constants, see "Custom Control Settings" (page 1465).

parms

A pointer to a configuration value. Some of the control setting constants passed in the `selector` parameter require that you provide an additional configuration value. For a description of which constants require configuration values, see "Custom Control Settings" (page 1465).

Return Value

A result code. See "Navigation Services Result Codes" (page 1486).

Discussion

If you provide an event-handling function and an event occurs in a Navigation Services dialog, Navigation Services calls your event-handling function and specifies one of the constants described in "Event Messages" (page 1475) in the `param` field of a `NavCBRec` (page 1451) structure. Navigation Services specifies this structure in the `callBackParms` parameter of your event-handling function. When Navigation Services supplies the `kNavCBStart` constant in the `param` field, your application can call the `NavCustomControl` function and pass one of the constants described in "Custom Control Settings" (page 1465) to control various aspects of the active Navigation Services dialog. For example, your application can tell Navigation Services to sort the browser list by date by calling the `NavCustomControl` function and passing the `kNavCtlSortBy` constant in the `selector` parameter and a pointer to the `kNavSortDateField` configuration constant in the `parms` parameter. (Some of the `NavCustomControlMessage` constants do not require a corresponding configuration constant.)

Note that your application can call the `NavCustomControl` function from within its event-handling function or its preview-drawing function.

Special Considerations

Navigation Services does not accept calls to the `NavCustomControl` function until an appropriate dialog box is fully initialized and displayed. Always check for the `kNavCBStart` constant, described in "Event Messages" (page 1475), in the `param` field of the `NavCBRec` structure before calling the `NavCustomControl` function.

Availability

Available in CarbonLib 1.0 and later when Navigation Services 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In
`Navigation.h`

NavDialogDispose

Disposes of a dialog reference.

```
void NavDialogDispose (
   NavDialogRef inDialog
);
```

Parameters

inDialog
A Navigation Services dialog reference previously obtained by your application.

Discussion
Use this function to dispose of a dialog reference when you are completely finished with its associated dialog. You may call `NavDialogDispose` from within your application-defined event-handling function.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X 10.0 and later.

Related Sample Code
CarbonSketch
QTCarbonShell

Declared In
`Navigation.h`

NavDialogGetReply

Reports the results of a dialog session (unless cancelled or programmatically terminated).

```
OSStatus NavDialogGetReply (
   NavDialogRef inDialog,
   NavReplyRecord *outReply
);
```

Parameters

inDialog
A reference to a previously created dialog.

outReply
A pointer to a reply record you allocate to be filled out by Navigation Services.

Return Value
A result code. See "Navigation Services Result Codes" (page 1486).

Discussion
Call this function when you obtain a value other than `kNavUserActionCancel` or `kNavUserActionNone` from the `NavDialogGetUserAction` (page 1432) function. Upon completion of the `NavDialogGetReply` (page 1430) function, Navigation Services fills out the specified reply record with information about the dialog session. When you are finished with the reply record, remember to dispose of it by calling the `NavDisposeReply` function.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X 10.0 and later.

Related Sample Code
CarbonSketch
QTCarbonShell

Declared In
`Navigation.h`

NavDialogGetSaveFileExtensionHidden

Gets the current state of extension hiding in a Save dialog.

```
Boolean NavDialogGetSaveFileExtensionHidden (
   NavDialogRef inPutFileDialog
);
```

Parameters
`inPutFileDialog`
A reference to the Save dialog. You can create a Save dialog using the `NavCreatePutFileDialog` (page 1426) function.

Return Value
`True` if the extension is hidden; `false` if the extension is visible or if there is no extension.

Discussion
This function can be called at any time to determine if a Save dialog is hiding the file extension—if any—of the file to be saved.

Availability
Not available in CarbonLib.
Available in Mac OS X 10.1 and later.

Declared In
`Navigation.h`

NavDialogGetSaveFileName

Obtains the current value of the filename text field in a Save dialog.

```
CFStringRef NavDialogGetSaveFileName (
   NavDialogRef inPutFileDialog
);
```

Parameters

inPutFileDialog

A reference to a previously created dialog.

Return Value

A reference to the string containing the save filename. You should retain this string reference if you need the information after the dialog is dismissed. On Mac OS X, the full filename is returned, including any extension that may be hidden from the user. See the CFString documentation for a description of the `CFStringRef` data type.

Discussion

This function provides a Unicode-based replacement for using the `kNavGetEditFileName` selector with the `NavCustomControl` (page 1429) function.

Special Considerations

Note that you cannot use `NavDialogGetSaveFileName` with a Save dialog created using the `NavPutFile` function. You should instead create your Save dialog using the `NavCreatePutFileDialog` function.

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X 10.0 and later.

Declared In

`Navigation.h`

NavDialogGetUserAction

Reports the user action taken to dismiss a dialog.

```
NavUserAction NavDialogGetUserAction (
   NavDialogRef inDialog
);
```

Parameters

inDialog

A reference to a previously created dialog.

Return Value

One of the constants defined by the `NavUserAction` enumeration. This value indicates the user action that dismissed the dialog. See "User Actions" (page 1484) for a description of the values that may be returned here.

Discussion

If the dialog has not been dismissed or if the dialog was terminated by using the `kNavCtlTerminate` selector with the `NavCustomControl` (page 1429) function, the `NavDialogGetUserAction` (page 1432) function returns the `kNavUserActionNone` constant. When you obtain a value other than `kNavUserActionCancel` or `kNavUserActionNone` after returning from a file-handling dialog, Navigation Services fills out a reply record that you can obtain with the `NavDialogGetReply` (page 1430) function.

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X 10.0 and later.

Related Sample Code
CarbonSketch

Declared In
`Navigation.h`

NavDialogGetWindow

Obtains a window reference for a dialog.

```
WindowRef NavDialogGetWindow (
   NavDialogRef inDialog
);
```

Parameters

inDialog

A reference to a previously created dialog.

Return Value
A window reference for the specified dialog. Note that a valid dialog reference may not have a window associated with it until the `NavDialogRun` (page 1433) function is called. If no window is associated with the specified dialog, the `NavDialogGetWindow` (page 1433) function returns `NULL`.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X 10.0 and later.

Declared In
`Navigation.h`

NavDialogRun

Displays a previously created dialog.

```
OSStatus NavDialogRun (
   NavDialogRef inDialog
);
```

Parameters

inDialog

A reference to a previously created Navigation Services dialog.

Return Value
A result code. See "Navigation Services Result Codes" (page 1486).

Discussion
You must create a dialog before displaying it. To create a dialog, call one of the `NavCreate...Dialog` functions described in "Choosing Files, Folders and Volumes" (page 1401) and "Saving Files" (page 1402). If you specify an application-modal or system-modal dialog, the `NavDialogRun` (page 1433) function returns after the dialog is dismissed. If you specify a window-modal dialog (sheet) or a modeless dialog, the `NavDialogRun` (page 1433) function returns immediately; in order to know when the dialog has been dismissed, you must supply an event-handling function and watch for the `kNavCBUserAction` event.

After the user interacts with the dialog, you can obtain information about the dialog session by calling the `NavDialogGetReply` (page 1430) function.

Version Notes
On Mac OS 9 and earlier, all Navigation Services dialogs are modal, even if a window-modal or modeless dialog is requested. However, the `kNavCBUserAction` event is still sent to your event-handling function. It is possible to use a single programming model on both Mac OS 9 and on Mac OS X, provided you assume that the `NavDialogRun` function returns immediately after displaying the dialog.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X 10.0 and later.

Related Sample Code
CarbonSketch
QTCarbonShell

Declared In
`Navigation.h`

NavDialogSetFilterTypeIdentifiers

Sets UTI filtering criteria for "get file" and "choose file" dialogs.

```
OSStatus NavDialogSetFilterTypeIdentifiers (
   NavDialogRef inGetFileDialog,
   CFArrayRef inTypeIdentifiers
);
```

Parameters

inGetFileDialog

A Navigation Services dialog reference obtained from calling `NavCreateChooseFileDialog` (page 1419) or `NavCreateGetFileDialog` (page 1423).

inTypeIdentifiers

A Core Foundation array of uniform type identifiers. This array specifies the file types that you want your dialog to enable. If you pass an empty array, all files are filtered (and will appear dimmed in the dialog). If you pass `NULL`, all files are enabled.

The file types you specify here also appear in the popup menu (displayed using the localized name associated with the UTI), allowing the user to filter by a specific file type. The "All readable documents" selection displays all the types specified in the UTI array.

Return Value
A result code. See "Navigation Services Result Codes" (page 1486).

Discussion
For simple filtering by file type, you should use this function instead of writing a custom filter callback function. However, you can also use this call in conjunction with a filter callback; your custom filter callback is called after `NavDialogSetFilterTypeIdentifiers` performs the initial filtering.

This function supersedes the list of `OSType` values you can pass in the `inTypeList` parameter in dialog creation functions.

You can call this function at any time, even while the dialog is displayed. For example, say your dialog contained a custom menu item to filter by a specific type. When the user selects your menu item, you could call `NavDialogSetFilterTypeIdentifiers` with the UTI corresponding to that type, and the dialog will automatically update with the new filtering criteria.

For more information about uniform type identifiers, see *Uniform Type Identifiers Overview*

Availability
Available in Mac OS X v10.4 and later.

Declared In
`Navigation.h`

NavDialogSetSaveFileExtensionHidden

Sets the current state of extension hiding in a Save dialog.

```
OSStatus NavDialogSetSaveFileExtensionHidden (
   NavDialogRef inPutFileDialog,
   Boolean inHidden
);
```

Parameters

`inPutFileDialog`

A reference to the Save dialog. You can create a Save dialog using the `NavCreatePutFileDialog` (page 1426) function.

`inHidden`

A Boolean value indicating whether the file extension should be hidden. Pass `true` to hide the file extension; `false` to make any extension visible.

Return Value
A result code. See "Navigation Services Result Codes" (page 1486).

Discussion
This function can be called at any time to hide or show the extension of the file to be saved in a Save dialog. If the current filename has no extension, hiding the extension has no effect.

Availability
Not available in CarbonLib.
Available in Mac OS X 10.1 and later.

Declared In
`Navigation.h`

NavDialogSetSaveFileName

Specifies the current value of the filename text field in a Save dialog.

```
OSStatus NavDialogSetSaveFileName (
   NavDialogRef inPutFileDialog,
   CFStringRef inFileName
);
```

Parameters

inPutFileDialog

A reference to a previously created dialog.

inFileName

The filename to specify.

Return Value
A result code. See "Navigation Services Result Codes" (page 1486).

Discussion
This function may be called at any time to set the current filename for a save operation. You may use it to set an initial filename before calling `NavDialogRun`, or to change the filename dynamically while a dialog is running.

This function provides a Unicode-based replacement for using the `kNavSetEditFileName` selector with the `NavCustomControl` (page 1429) function.

Special Considerations

Note that you cannot use `NavDialogSetSaveFileName` with a Save dialog created using the `NavPutFile` function. You should instead create your Save dialog using the `NavCreatePutFileDialog` function.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X 10.0 and later.

Declared In
`Navigation.h`

NavDisposeReply

Releases the memory allocated for a `NavReplyRecord` structure after your application has finished using the structure.

```
OSErr NavDisposeReply (
   NavReplyRecord *reply
);
```

Parameters

reply

A pointer to a structure of type `NavReplyRecord` (page 1458) that your application has created.

Return Value
A result code. See "Navigation Services Result Codes" (page 1486).

Discussion
If your application calls a Navigation Services function that uses a structure of type `NavReplyRecord` (page 1458), you must use the `NavDisposeReply` function afterward to release the memory allotted for the `NavReplyRecord` structure.

Availability
Available in CarbonLib 1.0 and later when Navigation Services 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Related Sample Code
CarbonSketch
QTCarbonShell
QTMetaData

Declared In
`Navigation.h`

NavGetDefaultDialogCreationOptions

Determines the default attributes or behavior for dialogs.

```
OSStatus NavGetDefaultDialogCreationOptions (
   NavDialogCreationOptions *outOptions
);
```

Parameters

outOptions
A pointer to a `NavDialogCreationOptions` (page 1452) structure that you provide. On return, Navigation Services fills out the structure with default dialog configuration values.

Return Value
A result code. See "Navigation Services Result Codes" (page 1486).

Discussion
This function gives you a simple way to initialize a `NavDialogCreationOptions` (page 1452) structure and set default options before creating a Navigation Services dialog. After you create the `NavDialogCreationOptions` structure, you can change the configuration options before you call one of the dialog creation functions.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X 10.0 and later.

Related Sample Code
CarbonSketch
QTCarbonShell

Declared In
`Navigation.h`

NavGetDefaultDialogOptions

Determines the default attributes or behavior for dialog boxes. (Deprecated in Mac OS X v10.5.)

Not recommended

```
OSErr NavGetDefaultDialogOptions (
   NavDialogOptions *dialogOptions
);
```

Parameters

dialogOptions

A pointer to a structure of type `NavDialogOptions` (page 1463). On return, Navigation Services fills out the structure with default option values that your application can change as needed.

Return Value

A result code. See "Navigation Services Result Codes" (page 1486).

Discussion

This function gives you a simple way to initialize a structure of type `NavDialogOptions` (page 1463) and set the default dialog box options before calling one of the dialog box display functions. After you create the `NavDialogOptions` structure, you can supply it with the `NavDialogOptions` constants, described in "Dialog Configuration Options" (page 1471), to change the configuration options.

Availability

Available in CarbonLib 1.0 and later when Navigation Services 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Carbon Porting Notes

Apple recommends that you adopt the dialog creation functions (`NavCreate...Dialog`); you pass these functions a `NavDialogCreationOptions` (page 1452) structure rather than a `NavDialogOptions` (page 1463) structure.

Related Sample Code

QTMetaData

Declared In

`Navigation.h`

NavGetFile

Displays a dialog box that prompts the user to select a file or files to be opened. (Deprecated in Mac OS X v10.5.)

Not recommended

```
OSErr NavGetFile (
   AEDesc *defaultLocation,
   NavReplyRecord *reply,
   NavDialogOptions *dialogOptions,
   NavEventUPP eventProc,
   NavPreviewUPP previewProc,
   NavObjectFilterUPP filterProc,
   NavTypeListHandle typeList,
   void *callBackUD
);
```

Parameters

defaultLocation

A pointer to an Apple event descriptor structure (AEDesc). Before calling NavGetFile, you can set up a structure of AEDesc type 'typeFSS' to specify a default location to be viewed. If you pass NULL in this parameter, Navigation Services defaults to the last location visited during a call to the NavGetFile function. If the file system specification in the AEDesc structure does not describe a directory or volume, Navigation Services uses the desktop as the default location.

reply

A pointer to a structure of type NavReplyRecord (page 1458). Upon return, Navigation Services uses this structure to provide data to your application about the results of your NavGetFile call.

dialogOptions

A pointer to a structure of type NavDialogOptions (page 1463). Before calling NavGetFile, set up this structure to specify dialog box settings. If you pass NULL in this parameter, Navigation Services uses the defaults for all options. See "Dialog Configuration Options" (page 1471) for a description of the default settings.

eventProc

A Universal Procedure Pointer (UPP) to your application-defined event-handling function. You obtain this UPP by calling the function NewNavEventUPP. Implementing an event-handling function allows your application to update windows after the user moves or resizes the dialog box. If you pass NULL in this parameter, the Open dialog box is not movable or resizable. For more information on event-handling functions, see NavEventProcPtr (page 1448).

previewProc

A Universal Procedure Pointer (UPP) to your application-defined preview function. Obtain this UPP by calling the function NewNavPreviewUPP. A preview function allows your application to draw previews or to override Navigation Services previews. For more information on preview functions, see NavPreviewProcPtr (page 1450).

filterProc

A Universal Procedure Pointer (UPP) to your application-defined filter function. Obtain this UPP by calling the function NewNavObjectFilterUPP. An application-defined filter function determines if a volume, directory, or file should be displayed in the browser list or pop-up menus. For more information on filter functions, see NavObjectFilterProcPtr (page 1449).

typeList

A handle to a structure of type NavTypeList (page 1460). Before calling, set up this structure to declare file types that your application can open.

callBackUD

A pointer to a value set by your application. When the NavGetFile function calls your event-handling function, the callBackUD value is passed back to your application.

Return Value

A result code. See "Navigation Services Result Codes" (page 1486).

Discussion
After your application calls the `NavGetFile` function to display an Open dialog box and the user selects one or more files and clicks the Open button, `NavGetFile` closes the dialog box and returns references to the files to be opened in the `NavReplyRecord` structure. Your application should check the `validRecord` field of the `NavReplyRecord` structure; if this field is set to `true`, your application should open the files specified in the `selection` field of the `NavReplyRecord` structure.

Always dispose of the `NavReplyRecord` structure after completing the file opening operation by calling the function `NavDisposeReply` (page 1436). If you fail to use the `NavDisposeReply` function, memory used for the `NavReplyRecord` structure remains allocated and unavailable.

If you use the Show pop-up menu in an Open dialog box, your application must provide adequate kind strings to describe its native file types. For more information on `kind` strings, see Inside Macintosh: More Macintosh Toolbox.

Availability
Available in CarbonLib 1.0 and later when Navigation Services 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.

Carbon Porting Notes

Apple recommends you use the function `NavCreateGetFileDialog` (page 1423) in order to take advantage of Mac OS X features like Unicode, long filenames and enhanced modality.

Related Sample Code
QTMetaData

Declared In
`Navigation.h`

NavLibraryVersion

Reports the currently installed version of the Navigation Services shared library. (Deprecated in Mac OS X v10.5.)

```
UInt32 NavLibraryVersion (
   void
);
```

Return Value
An unsigned 32-bit integer. This value represents the version number (in binary-coded decimal) of Navigation Services installed on the user's system.

Discussion
If you want to use features that are present only in a specific version of Navigation Services, use the `NavLibraryVersion` function to determine which version of Navigation Services is installed.

Availability
Available in CarbonLib 1.0 and later when Navigation Services 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`Navigation.h`

NavLoad

Pre-loads the Navigation Services shared library.

Unsupported

```
OSErr NavLoad (
   void
);
```

Return Value
A result code. See "Navigation Services Result Codes" (page 1486).

Discussion
Use this function to pre-load the Navigation Services library. Pre-loading increases the memory used by your application, but it provides the best performance when using Navigation Services functions. If you don't use the `NavLoad` function, the Navigation Services shared library may not be loaded until your application calls one of the Navigation Services functions. If you use the `NavLoad` function, you must call the function `NavUnload` (page 1446) if you want to release reserved memory prior to quitting.

Availability
Available in CarbonLib 1.0 and later when Navigation Services 1.0 or later is present. Not available in Mac OS X.

Declared In
`Navigation.h`

NavNewFolder

Displays a dialog box that prompts the user to create a new folder. (Deprecated in Mac OS X v10.5.)

Not recommended

```
OSErr NavNewFolder (
   AEDesc *defaultLocation,
   NavReplyRecord *reply,
   NavDialogOptions *dialogOptions,
   NavEventUPP eventProc,
   void *callBackUD
);
```

Parameters
`defaultLocation`
A pointer to an Apple event descriptor structure (`AEDesc`). Before calling `NavNewFolder`, you can set up a structure of `AEDesc` type `'typeFSS'` to specify a default location to be viewed. If you pass `NULL` in this parameter, Navigation Services displays the last location visited during a call to the `NavNewFolder` function. If the file system specification in the `AEDesc` structure does not describe a directory or volume, Navigation Services uses the desktop as the default location.

reply

A pointer to a structure of type `NavReplyRecord` (page 1458). Upon return, Navigation Services uses this structure to provide data to your application about the results of the `NavNewFolder` function call.

dialogOptions

A pointer to a structure of type `NavDialogOptions` (page 1463). Before calling `NavNewFolder`, set up this structure to specify dialog box settings. If you pass `NULL` in this parameter, Navigation Services uses the defaults for all options. See "Dialog Configuration Options" (page 1471) for a description of the default settings.

eventProc

A Universal Procedure Pointer (UPP) to your application-defined event-handling function. You obtain this UPP by calling the function `NewNavEventUPP`. Implementing an event-handling function allows your application to update windows after the user moves or resizes the dialog box. If you pass `NULL` in this parameter, the dialog box is not movable or resizable. For more information on event-handling functions, see `NavEventProcPtr` (page 1448).

callBackUD

A pointer to a value set by your application. When the `NavNewFolder` function calls your event-handling function, the `callBackUD` value is passed back to your application.

Return Value
A result code. See "Navigation Services Result Codes" (page 1486).

Discussion
This function provides a way for your application to prompt the user to create a new folder. This might be useful for creating a project folder, for example.

Availability
Available in CarbonLib 1.0 and later when Navigation Services 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.

Carbon Porting Notes

Apple recommends you use the function `NavCreateNewFolderDialog` (page 1424) in order to take advantage of Mac OS X features like Unicode, long filenames and enhanced modality.

Declared In
`Navigation.h`

NavPutFile

Displays a Save dialog box. (Deprecated in Mac OS X v10.5.)

Not recommended

```
OSErr NavPutFile (
   AEDesc *defaultLocation,
   NavReplyRecord *reply,
   NavDialogOptions *dialogOptions,
   NavEventUPP eventProc,
   OSType fileType,
   OSType fileCreator,
   void *callBackUD
);
```

Parameters

defaultLocation

A pointer to an Apple event descriptor structure (`AEDesc`). Before calling `NavPutFile`, you can set up a structure of `AEDesc` type `'typeFSS'` to specify a default location to be viewed. If you pass `NULL` in this parameter, Navigation Services displays the last location visited during a call to the `NavPutFile` function. If the file system specification in the `AEDesc` structure does not describe a directory or volume, Navigation Services uses the desktop as the default location.

reply

A pointer to a structure of type `NavReplyRecord` (page 1458). Upon return, Navigation Services uses this structure to provide data to your application about the results of your `NavPutFile` call.

dialogOptions

A pointer to a structure of type `NavDialogOptions` (page 1463). Before calling `NavPutFile`, you can set up this structure to specify dialog box settings. If you pass `NULL` in this parameter, Navigation Services uses the defaults for all options. See "Dialog Configuration Options" (page 1471) for a description of the default settings.

eventProc

A Universal Procedure Pointer (UPP) to your application-defined event-handling function. You obtain this UPP by calling the function `NewNavEventUPP`. Implementing an event-handling function allows your application to update windows after the user moves or resizes the dialog box. If you pass `NULL` in this parameter, the Save dialog box is not movable or resizable. For more information on event-handling functions, see `NavEventProcPtr` (page 1448).

fileType

A four-character code. Pass the file type code for the document to be saved.

fileCreator

A four-character code. Pass the file creator code for the document to be saved. Under Navigation Services 2.0 or later, you may pass the "Generic File Signature Constant" (page 1479) constant if you want to override the types of files appearing in the Format popup.

callBackUD

A pointer to a value set by your application. When the `NavPutFile` function calls your event-handling function, the `callBackUD` value is passed back to your application.

Return Value

A result code. See "Navigation Services Result Codes" (page 1486). Note: If you specify the `kNavDontResolveAliases` constant as a dialog box option, as described in "Dialog Configuration Options" (page 1471), before calling the `NavPutFile` function, Navigation Services returns a `paramErr (-50)`.

Discussion

After your application calls the `NavPutFile` function to display a Save dialog box and the user selects a location, enters a filename, and clicks OK, `NavPutFile` closes the dialog box and returns references to the file to be saved in the `NavReplyRecord` structure. Your application should check the `validRecord` field of the `NavReplyRecord` structure; if this field is set to `true`, your application should save the file and call the function `NavCompleteSave` (page 1415).

If you specify the Format pop-up menu in a dialog box displayed by the `NavPutFile` function, your application must provide adequate kind strings to describe the file types available. If the user uses the Format menu to save a file to a format other than the file's native format, Navigation Services translates the file automatically. If you wish to turn off automatic translation, set to `false` the value of the `translationNeeded` field of the `NavReplyRecord` structure you pass in the `reply` parameter. If you turn off automatic translation, your application is responsible for any required translation.

Availability
Available in CarbonLib 1.0 and later when Navigation Services 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.

Carbon Porting Notes

Apple recommends you use the function `NavCreatePutFileDialog` (page 1426) in order to take advantage of Mac OS X features like Unicode, long filenames and enhanced modality.

Related Sample Code
QTMetaData

Declared In
`Navigation.h`

NavServicesAvailable

Reports whether the Navigation Services library is available on the user's system.

```
pascal Boolean NavServicesAvailable
```

Return Value
A Boolean value. This function returns `true` if Navigation Services is available, `false` if not.

Discussion
Use this function before attempting to use Navigation Services on Mac OS 8 and Mac OS 9. It is not necessary to call this function on Mac OS X, as Navigation Services is always available.

Special Considerations

There is a known problem with Navigation Services 1.0 that occurs if you call `NavServicesAvailable` more than once without the Appearance Manager being installed. Make sure that you check for the presence of the Appearance Manager before calling `NavServicesAvailable`.

Version Notes
Available in Navigation Services 1.0 and later.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Navigation.h`

NavServicesCanRun

Unsupported

```
Boolean NavServicesCanRun (
   void
);
```

Availability
Available in CarbonLib 1.0 and later when Navigation Services 1.0 or later is present. Not available in Mac OS X.

Declared In
`Navigation.h`

NavTranslateFile

Provides a means for files opened through Navigation Services to be read from different file formats. (Deprecated in Mac OS X v10.5.)

```
OSErr NavTranslateFile (
   const NavReplyRecord *reply,
   NavTranslationOptions howToTranslate
);
```

Parameters

reply

A pointer to a structure of type `NavReplyRecord` (page 1458). Upon return, Navigation Services uses this structure to provide translation information about the selected files.

howToTranslate

A value of type `NavTranslationOptions`. Pass one of these constants to tell Navigation Services how to perform the translation: either in-place or by making a copy of the file. For a description of the constants, see "Translation Options" (page 1483).

Return Value
A result code. See "Navigation Services Result Codes" (page 1486).

Discussion
Under automatic file translation, Navigation Services calls the `NavTranslateFile` function as necessary before returning from a file-opening function.

Your application can perform its own translation using the `NavReplyRecord` structure you specified in the `translateInfo` parameter. The `NavReplyRecord` structure contains a list of descriptors for the file or files to be opened and a corresponding list of translation specification records that can be passed to the Translation Manager. To determine if your application has to translate a file, your application can examine the `NavReplyRecord` structure to see if Navigation Services set the `translationNeeded` field to `true`. (The `translationNeeded` field of the `NavReplyRecord` structure is also set to `true` after returning from a `NavGetFile` call during which automatic translation was performed.) If you want to turn off automatic file translation, set the constant `kNavDontAutoTranslate` in the `dialogOptionFlags` field of the structure of type `NavDialogOptions` (page 1463) that you pass in the `dialogOptions` parameter of the file-opening function.

If your application uses the `NavTranslateFile` function after opening a file without automatic translation, Navigation Services checks to see if the source location can accept a new file. If the source location is not available (as occurs when the volume is locked or there is insufficient space), Navigation Services prompts the user to select a location in which to save the translated file. The same prompt may occur when automatic translation is enabled in an Open dialog box.

Availability
Available in CarbonLib 1.0 and later when Navigation Services 1.0 or later is present.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`Navigation.h`

NavUnload

Unloads the Navigation Services shared library.

Unsupported

```
OSErr NavUnload (
   void
);
```

Return Value
A result code. See "Navigation Services Result Codes" (page 1486).

Discussion
This function allows your application to unload the Navigation Services library and release the memory reserved for it. If you use the function `NavLoad` (page 1441) to load the Navigation Services library, you must call the `NavUnload` function if you want to release reserved memory prior to quitting.

Availability
Available in CarbonLib 1.0 and later when Navigation Services 1.0 or later is present. Not available in Mac OS X.

Declared In
`Navigation.h`

NewNavEventUPP

Creates a new universal procedure pointer to your application-defined event–handling function.

```
NavEventUPP NewNavEventUPP (
   NavEventProcPtr userRoutine
);
```

Parameters
`userRoutine`
A pointer to your event–handling function.

Return Value
On return, a universal procedure pointer (UPP) to the event–handling function. See the description of the `NavEventUPP` data type.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.

Related Sample Code
CarbonSketch
QTMetaData

Declared In
`Navigation.h`

NewNavObjectFilterUPP

Creates a new universal procedure pointer to your application-defined filter function.

```
NavObjectFilterUPP NewNavObjectFilterUPP (
   NavObjectFilterProcPtr userRoutine
);
```

Parameters

`userRoutine`
A pointer to your filter function.

Return Value
On return, a universal procedure pointer (UPP) to the filter function. See the description of the `NavObjectFilterUPP` data type.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.

Declared In
`Navigation.h`

NewNavPreviewUPP

Creates a new universal procedure pointer to your application-defined preview function.

```
NavPreviewUPP NewNavPreviewUPP (
   NavPreviewProcPtr userRoutine
);
```

Parameters

`userRoutine`
A pointer to your preview function.

Return Value
On return, a universal procedure pointer (UPP) to the preview function. See the description of the `NavPreviewUPP` data type.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.

Declared In
`Navigation.h`

Callbacks

NavEventProcPtr

A pointer to an event-handling function that handles events such as window updating and resizing.

```
typedef void (*NavEventProcPtr) (
                NavEventCallbackMessage callBackSelector,
                NavCBRecPtr callBackParms,
                void * callBackUD);
```

If you name your function `MyNavEventProc`, you would declare it like this:

```
void MyNavEventProc (
            NavEventCallbackMessage callBackSelector,
            NavCBRecPtr callBackParms,
            void * callBackUD);
```

Parameters

callBackSelector

One of the values specified by the `NavEventCallbackMessage` data type. These values indicate which type of event your function must respond to. For a description of the constants that represent these values, see "Event Messages" (page 1475).

callBackParms

A pointer to a `NavCBRec` (page 1451) structure. Your application uses the data supplied in this structure to process the event.

callBackUD

A pointer to a value set by your application when it calls a Navigation Services dialog creation function. When Navigation Services calls your event-handling function, the `callBackUD` value is passed back to your application in this parameter.

Discussion

Register your event-handling function by passing a Universal Procedure Pointer (UPP) in the `eventProc` parameter of a Navigation Services dialog creation function. You obtain this UPP by calling the function `NewNavEventUPP` and passing a pointer to your event-handling function. If you determine that an event is appropriate for your event-handling function, you can call other functions to handle custom control drawing.

When events involve controls, your event-handling function must respond to events only for your application-defined controls. To determine which control is affected by an event, pass the `kNavCtlGetFirstControlID` constant, described in "Custom Control Settings" (page 1465), in the `selector` parameter of the function `NavCustomControl` (page 1429).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Navigation.h`

NavObjectFilterProcPtr

A pointer to a filter function that determines whether file objects should be displayed in the browser list and navigation menus.

```
typedef Boolean (*NavObjectFilterProcPtr)
(
                AEDesc * theItem,
                void * info,
                void * callBackUD,
                NavFilterModes filterMode);
```

If you name your function `MyNavObjectFilterProc`, you would declare it like this:

```
Boolean MyNavObjectFilterProc (
            AEDesc * theItem,
            void * info,
            void * callBackUD,
            NavFilterModes filterMode);
```

Parameters

theItem

A pointer to an Apple event descriptor structure (`AEDesc`). Navigation Services uses this structure to provide information about the object being passed to your filter function. Always check the Apple event descriptor type before deciding if an object needs to be filtered. Never assume that an object is a file specification, because the browser or pop-up menus may contain objects of other types. Make sure that your function only returns `true` if it recognizes the object.

info

A pointer to a `NavFileOrFolderInfo` (page 1456) structure. Navigation Services uses this structure to provide file or folder information about the item being passed to your filter function. This information is only valid for objects of descriptor types `'typeFSS'` or `'typeFSRef'`.

callBackUD

A pointer to a value set by your application when it calls a Navigation Services dialog creation function. When Navigation Services calls your filter function, the `callBackUD` value is passed back to your application in this parameter.

filterMode

A value representing which list of objects is currently being filtered. For a description of the constants used to represent these values, see "Object Filtering Constants" (page 1480).

Return Value

A Boolean value. If your application returns `true`, Navigation Services displays the object. If your application returns `false`, Navigation Services displays the object as dimmed.

Discussion

Register your filter function by passing a Universal Procedure Pointer (UPP) in the `filterProc` parameter of a dialog creation function. You obtain this UPP by calling the function `NewNavObjectFilterUPP` and passing a pointer to your filter function. Navigation Services calls your filter function to determine whether a file object should be displayed in the browser list or the pop-up menus.

If you use a filter function in conjunction with built-in translation, you should provide a list of file types to inform Navigation Services which document types your application can open. You can do so using the following methods:

- Call the `NavDialogSetFilterTypeIdentifiers` (page 1434) function on an existing dialog to filter by uniform type identifiers. This method is preferable in Mac OS X v10.4 and later.
- Provide a list of allowable `OSType` file types in the `inTypeList` parameter of a file-opening function such as `NavCreateGetFileDialog`

If you provide a list of file types, your filter callback is called only for the files that match the specified type list. For example, if you wanted to enable only text files below a certain size, you could use `NavDialogSetFilterTypeIdentifiers` (page 1434) to enable only text files, and then use a filter callback to screen for file size. You should make sure that your filter callback doesn't automatically eliminate a document type in the filter list (for example, if the list allows JPEG files and the callback eliminates everything but PICT files). This is to ensure that the user can always see some files when a particular file type is selected from the Enable popup menu.

If your filter function returns a result of `true`, Navigation Services displays the object. Note that this is the opposite of Standard File filter functions.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Navigation.h`

NavPreviewProcPtr

A pointer to a preview function that displays custom file previews.

```
typedef Boolean (*NavPreviewProcPtr)
(
                NavCBRecPtr callBackParms,
                void * callBackUD);
```

If you name your function `MyNavPreviewProc`, you would declare it like this:

```
Boolean MyNavPreviewProc (
        NavCBRecPtr callBackParms,
        void * callBackUD);
```

Parameters

`callBackParms`

A pointer to a `NavCBRec` (page 1451) structure. Navigation Services uses this structure to provide data needed for your function to draw the preview.

`callBackUD`

A pointer to a value set by your application when it calls a Navigation Services function such as `NavCreateGetFileDialog`. When Navigation Services calls your preview function, the `callBackUD` value is passed back to your application in this parameter.

Return Value
A Boolean value. Your application returns `true` if your preview function successfully draws the custom file preview. If your preview function returns `false`, Navigation Services displays the preview if the file contains a valid `'pnot'` resource. If your preview function returns `false` and a `'pnot'` resource is not available, Navigation Services displays a blank preview area.

Discussion
Register your preview function by passing the resulting Universal Procedure Pointer (UPP) in the `previewProc` parameter of a Navigation Services dialog creation function. You obtain this UPP by calling the function `NewNavPreviewUPP` and passing a pointer to your preview-drawing function. When the user selects a file, Navigation Services calls your preview-drawing function. Your preview function, in turn, calls the function `NavCustomControl` (page 1429) to determine if the preview area is visible and, if so, what its dimensions are.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Navigation.h`

Data Types

NavDialogRef

An opaque reference to an instance of a Navigation Services dialog.

```
typedef struct __NavDialog * NavDialogRef;
```

Discussion
Your application obtains a `NavDialogRef` by calling one of the dialog creation functions described in "Choosing Files, Folders and Volumes" (page 1401) and "Saving Files" (page 1402). Once you obtain a valid reference, you pass it to other functions in order to display and process dialogs. When you are completely finished using the reference, dispose of it by calling the function `NavDialogDispose` (page 1430). This data type is available in CarbonLib 1.1 and later and in Mac OS X. It replaces the `NavContext` data type previously used by Navigation Services.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Navigation.h`

NavCBRec

Provides information you can use for event-handling and customization.

```
struct NavCBRec {
    UInt16 version;
    NavDialogRef context;
    WindowRef window;
    Rect customRect;
    Rect previewRect;
    NavEventData eventData;
    NavUserAction userAction;
    char reserved[218];
};
typedef struct NavCBRec NavCBRec;
typedef NavCBRec * NavCBRecPtr;
```

Fields

`version`
: Identifies the version of this structure. This value is defined by the `kNavCBRecVersion` constant.

`context`
: An opaque object identifying the dialog instance.

`window`
: An opaque object identifying the dialog's window.

`customRect`
: A local coordinate rectangle describing the customization area available to your application. This determines how much room your application has to install custom controls.

`previewRect`
: A local coordinate rectangle describing the preview area available to your application's preview function. The minimum size is 145 pixels wide by 118 pixels high.

`eventData`
: A structure of type `NavEventData` (page 1454). This structure provides event-specific data to your `NavEventProcPtr` (page 1448) function.

`userAction`
: A constant specifying the action taken by the user, generating a `kNavCBUserAction` event. See "User Actions" (page 1484) for a description of the possible values for this field.

  This field is available in CarbonLib 1.1 and later or in Mac OS X version 10.0 and later.

`reserved`
: Reserved.

Discussion

The `NavCBRec` structure is passed to your application-defined event-handling and preview functions. For more information on event-handling and preview functions, see `NavEventProcPtr` (page 1448) and `NavPreviewProcPtr` (page 1450), respectively.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Navigation.h`

NavDialogCreationOptions

Contains dialog configuration settings you can pass to Navigation Services dialog creation functions.

```
struct NavDialogCreationOptions {
    UInt16 version;
    NavDialogOptionFlags optionFlags;
    Point location;
    CFStringRef clientName;
    CFStringRef windowTitle;
    CFStringRef actionButtonLabel;
    CFStringRef cancelButtonLabel;
    CFStringRef saveFileName;
    CFStringRef message;
    UInt32 preferenceKey;
    CFArrayRef popupExtension;
    WindowModality modality;
    WindowRef parentWindow;
    char reserved[16];
};
typedef struct NavDialogCreationOptions NavDialogCreationOptions;
```

Fields

`version`

Identifies the version of this structure. The structure version is represented by the `kNavDialogCreationOptionsVersion` constant.

`optionFlags`

One of several constants defined by the `NavDialogOptionFlags` data type as described in "Dialog Configuration Options" (page 1471).

`location`

A point describing the location of the upper-left corner of the dialog window, in global coordinates. If you set this field to (-1,-1), then the dialog window appears in the same location as when it was last closed. The size and location of the dialog window is persistent, but defaults to opening in the middle of the main screen if any portion is not visible when opened at the persistent location and size.

This field is ignored for sheet dialogs.

`clientName`

A string that identifies your application in the window title of file dialogs and in the message displayed for the Save Changes, Review Changes, and Ask Discard changes alerts.

On Mac OS 8 and 9, Navigation Services cannot maintain persistence information for your application if you do not provide this string.

`windowTitle`

A string that you can provide to override the default window title. If you pass `NULL`, the default window title is used.

`actionButtonLabel`

An alternative button title for the dialog's default button. If you pass `NULL`, the button uses the default label (Open or Save, for example.

`cancelButtonLabel`

An alternative button title for the dialog's Cancel button. If you pass `NULL`, the default button title is used.

`saveFileName`

The default filename for a file to be saved (Save dialog only). If you pass `NULL`, the filename field is blank.

`message`

For the file dialogs, a string for the banner, or prompt, below the browser list. This message can provide more descriptive instructions for the user. If you pass `NULL`, no banner appears and the browser list expands to fill that area.

For the Save Changes, Review Changes and Ask Discard Changes alerts, a string specifying a custom message that replaces the default message.

`preferenceKey`

An application-defined value that identifies which set of dialog preferences Navigation Services should use. If your application maintains multiple sets of preferences for a particular type of dialog, you can determine which set is active by specifying the appropriate value in the `preferenceKey` field. For example, an application may provide one set of preferences when it calls the function to open text files and a different set of preferences when opening movie files. If you do not wish to provide a preference key, specify 0 for the `preferenceKey` value.

`popupExtension`

A reference to an array of menu item strings. These strings are used to add extra menu items to the Show pop-up menu in an Open dialog or to the Format pop-up menu in a Save dialog. Your application can use this array to add additional document types to be opened or saved, or different ways of saving a file (with or without line breaks, for example).

`modality`

This value allows you to specify the modality of the dialog. The default modality for all dialogs is `kWindowModalityAppModal`. If you specify the `kWindowModalityWindowModal` constant to make a dialog appear as a sheet, you must provide a valid window reference in the `parentWindow` field. If you specify the `kWindowModalityWindowModal` constant on Mac OS 8 or 9, the modality is set to `kWindowModalityAppModal`.

This field is available in CarbonLib 1.1 and later or in Mac OS X version 10.0 and later.

`parentWindow`

A reference to the parent window for a sheet.

This field is available in CarbonLib 1.1 and later or in Mac OS X version 10.0 and later.

`reserved`

Reserved.

Discussion

When you create a Navigation Services dialog, using one of the `NavCreate...Dialog` creation functions, you must supply a `NavDialogCreationOptions` structure to specify the appearance and behavior of the dialog. You can initialize a `NavDialogCreationOptions` structure using the `NavGetDefaultDialogCreationOptions` (page 1437) function; this fills out the structure with the default dialog creation settings.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Navigation.h`

NavEventData

Contains event data for Navigation Services dialogs.

```
struct NavEventData {
    NavEventDataInfo eventDataParms;
    SInt16 itemHit;
};
typedef struct NavEventData NavEventData;
```

Fields

`eventDataParms`

A structure of type `NavEventDataInfo` (page 1455).

`itemHit`

A signed integer value. On return, this value represents the item number of the dialog item last clicked by the user. If the user clicks something other than a valid Navigation Services-generated control item, this value is -1.

Discussion

The `NavEventData` structure is passed to your application-defined event-handling or preview function in the `eventData` field of the `NavCBRec` structure.

The `NavEventData` structure contains a structure of type `NavEventDataInfo` (page 1455). In Navigation Services 1.1 or later, the `NavEventData` structure also contains a field describing the dialog item last clicked by the user.

Version Notes

`itemHit` field added in Navigation Services 1.1.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Navigation.h`

NavEventDataInfo

Provides event–handling data to your application.

```
union NavEventDataInfo {
    EventRecord * event;
    void * param;
};
typedef union NavEventDataInfo NavEventDataInfo;
```

Fields

`event`

A pointer to the `EventRecord` structure describing an event to be handled by your event-handling function.

`param`

A pointer to additional event data. In most cases, this data consists of an Apple event descriptor list (`AEDescList`) for the file or files affected by the event described in the `event` field. For example, if the event consists of the user making a selection in the browser list, the `AEDescList` specifies the file or files selected.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`Navigation.h`

NavFileOrFolderInfo

Contains file or folder information for use by your application-defined filter function.

```
struct NavFileOrFolderInfo {
    UInt16 version
    Boolean isFolder
    Boolean visible
    UInt32 creationDate
    UInt32 modificationDate
    union {
        struct {
            Boolean locked;
            Boolean resourceOpen;
            Boolean dataOpen;
            Boolean reserved1;
            UInt32 dataSize;
            UInt32 resourceSize;
            FInfo finderInfo;
            FXInfo finderXInfo;
        } fileInfo;
        struct {
            Boolean shareable;
            Boolean sharePoint;
            Boolean mounted;
            Boolean readable;
            Boolean writeable;
            Boolean reserved2;
            UInt32 numberOfFiles;
            DInfo finderDInfo;
            DXInfo finderDXInfo;
            OSType folderType;
            OSType folderCreator;
            char reserved3[206];
        } folderInfo;
    } fileAndFolder;
};
typedef struct NavFileOrFolderInfo NavFileOrFolderInfo;
```

Fields

`version`
Identifies the version of this structure.

`isFolder`
A Boolean value. If this value is set to `true`, the object being described is a folder or volume; otherwise, the value is set to `false`. An alias to a folder or volume returns `true`. Check for the `kIsAlias` constant in the `fileAndFolder.folderInfo.finderDInfo` field to determine whether an object is an alias.

`visible`
A Boolean value. If this value is set to `true`, the object being described is visible in the browser list; otherwise, the value is set to `false`.

`creationDate`
The creation date of the object being described.

`modificationDate`
The modification date of the object being described.

`fileAndFolder.fileInfo.locked`
If `isFolder` is `false`, a Boolean value indicating whether the file is locked.

`fileAndFolder.fileInfo.resourceOpen`
If `isFolder` is `false`, a Boolean value indicating whether the resource fork of the file is open.

`fileAndFolder.fileInfo.dataOpen`
If `isFolder` is `false`, a Boolean value indicating whether the data fork of the file is open.

`fileAndFolder.fileInfo.reserved1`
Reserved.

`fileAndFolder.fileInfo.dataSize`
If `isFolder` is `false`, the size of the file's data fork.

`fileAndFolder.fileInfo.resourceSize`
If `isFolder` is `false`, the size of the file's resource fork.

`fileAndFolder.fileInfo.finderInfo`
If `isFolder` is `false`, a structure specifying further information about the file. See the Finder Interface documentation for more information on the `FInfo` structure.

`fileAndFolder.fileInfo.finderXInfo`
If `isFolder` is `false`, a structure specifying extended Finder information for the file. See the Finder Interface documentation for more information on the `FXInfo` structure.

`fileAndFolder.folderInfo.shareable`
If `isFolder` is `true`, a Boolean value indicating whether the folder is shareable.

`fileAndFolder.folderInfo.sharePoint`
If `isFolder` is `true`, a Boolean value indicating whether the folder is a share point.

`fileAndFolder.folderInfo.mounted`
If `isFolder` is `true`, a Boolean value indicating whether the folder is mounted.

`fileAndFolder.folderInfo.readable`
If `isFolder` is `true`, a Boolean value indicating whether the folder is readable.

`fileAndFolder.folderInfo.writeable`
If `isFolder` is `true`, a Boolean value indicating whether the folder is writeable.

`fileAndFolder.folderInfo.reserved2`
Reserved.

`fileAndFolder.folderInfo.numberOfFiles`
If `isFolder` is `true`, the number of files in the folder.

`fileAndFolder.folderInfo.finderDInfo`
If `isFolder` is `true`, a directory information structure describing the folder. See the Finder Interface documentation for further information on the `DInfo` structure.

`fileAndFolder.folderInfo.finderDXInfo`
If `isFolder` is `true`, an extended directory information structure describing the folder. See the Finder Interface documentation for further information on the `DXInfo` structure.

`fileAndFolder.folderInfo.folderType`
If `isFolder` is `true`, the package type (for structure version 1 or greater).

`fileAndFolder.folderInfo.folderCreator`
If `isFolder` is `true`, the creator code for the package (for structure version 1 or greater).

Discussion

The `NavFileOrFolderInfo` structure contains file or folder information for use by your application-defined filter function. Your filter function can determine whether the currently selected object is a file by checking the `isFolder` field of the `NavFileOrFolderInfo` structure for the value `false`. After making this determination, you can obtain more information about the object from the structure specified in the `fileAndFolder` field.

Special Considerations

The information in this structure is valid only for HFS file objects.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Navigation.h`

NavReplyRecord

Contains information about user interaction with a dialog.

```
struct NavReplyRecord {
    UInt16 version;
    Boolean validRecord;
    Boolean replacing;
    Boolean isStationery;
    Boolean translationNeeded;
    AEDescList selection;
    ScriptCode keyScript;
    FileTranslationSpecArrayHandle fileTranslation;
    UInt32 reserved1;
    CFStringRef saveFileName;
    Boolean saveFileExtensionHidden;
    UInt8 reserved2;
    char reserved[225];
};
typedef struct NavReplyRecord NavReplyRecord;
```

Fields

`version`
: Identifies the version of this structure. The structure version is represented by the constant `kNavReplyRecordVersion`.

`validRecord`
: A Boolean value of `true` if the user closes a dialog by pressing Return or Enter, or by clicking the default button in an Open or Save dialog. If this field is `false`, all other fields are unused and do not contain valid data.

`replacing`
: A Boolean value of `true` if the user chooses to save a file by replacing an existing file (thereby necessitating the removal or renaming of the existing file).

`isStationery`
: A Boolean value informing your application whether the file about to be saved should be saved as a stationery document.

`translationNeeded`
: A Boolean value indicating whether translation was or will be needed for files selected in Open and Save dialogs.

`selection`
: For a file-opening or file-choosing dialog, this is an Apple event descriptor list (`AEDescList`) containing references to items selected by the user. Navigation Services creates this list, which is automatically disposed of when your application calls the `NavDisposeReply` function. Some dialogs may return one or more items; you can determine the number of items in the list by calling the Apple Event Manager function `AECountItems`. Each selected HFS file object is described in an `AEDesc` structure of type `'typeFSS'` or `'typeFSRef'`. You can coerce this descriptor into a file reference to perform operations such as opening the file. If you use one of the Carbon-compliant dialog creation functions described in"Choosing Files, Folders and Volumes" (page 1401) and "Saving Files" (page 1402), the descriptor is of type `'typeFSS'` on Mac OS 8 or 9; on Mac OS X systems, this descriptor is of type `'typeFSRef'`.File-saving dialogs always return a single descriptor in the list. If you use the `NavCreatePutFileDialog` (page 1426) function, this descriptor specifies the directory where the file is to be saved. You can obtain the name for the save file from the `saveFileName` field.

`keyScript`
: The keyboard script system used for the filename.

`fileTranslation`
: A handle to a Translation Manager structure of type `FileTranslationSpec`. This structure contains a corresponding translation array for each file reference returned in the `selection` field. When opening files, Navigation Services performs the translation automatically unless you set the `kNavDontAutoTranslate` flag in the `dialogOptionFlags` field of the `NavDialogCreationOptions` (page 1452) structure. When Navigation Services performs an automatic translation, the `FileTranslationSpec` structure is strictly for the Translation Manager's use. If you turn off automatic translation, your application may use the `FileTranslationSpec` structure for your own translation scheme. If the user chooses a translation for a saved file, the `FileTranslationSpec` structure contains a single translation reference for the saved file and the `translationNeeded` field of the `NavReplyRecord` structure is set to `true`. The handle to the `FileTranslationSpec` structure is locked, so you can safely use dereferenced pointers.

`reserved1`
: Reserved.

`saveFileName`
: If the reply record is filled out by a dialog created with the `NavCreatePutFileDialog` (page 1426) function, this field contains a string specifying the name of a file to be saved. This field contains the entire name of the file, regardless of whether or not any file extension is visible to the user. You can identify the directory in which the file is to be saved by checking the `selection` field.

  This field was added in structure version 1.

`saveFileExtensionHidden`
: A Boolean value indicating whether the extension on the name of the saved file should be hidden. Once the file has been saved, the client call the `NavCompleteSave` function. `NavCompleteSave` hides the extension on the file. However, the client needs to know that the extension is hidden so that it can display the document name correctly in the user interface, such as in window titles and menus. This field is only used if the client has requested extension preservation using the `kNavPreserveSaveFileExtension` dialog option flag. This field was added in structure version 2.

`reserved2`
: Reserved.

`reserved`
: Reserved.

Discussion
Navigation Services uses the `NavReplyRecord` structure to provide your application with information about the user's interactions with a Navigation Services dialog. If the dialog is created with the Carbon-compliant `NavCreate...Dialog` functions, you obtain the `NavReplyRecord` by calling the `NavDialogGetReply` (page 1430) function after your event-handling function receives the `kNavCBUserAction` event. If you create the dialog using one of the older functions, you pass the address of a `NavReplyRecord` directly to the function that invokes the dialog. When your application is through using the structure, remember to dispose of it by calling the function `NavDisposeReply` (page 1436).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Navigation.h`

NavTypeList

Defines a list of file types that your application is capable of opening.

```
struct NavTypeList {
    OSType componentSignature;
    short reserved;
    short osTypeCount;
    OSType osType[1];
};
typedef struct NavTypeList NavTypeList;
typedef NavTypeList * NavTypeListPtr;
```

Fields

`componentSignature`
A four character code specifying your application signature. If you want your application to be able to open all files of the types you specify in the `osType` field (regardless of which application created them), specify the `kNavGenericSignature` constant in this field.

`reserved`
Reserved.

`osTypeCount`
A number indicating how many file types are defined in the `osType` field.

`osType`
A list of file types your application can open.

Discussion
Your application uses the `NavTypeList` structure to define a list of file types that your application is capable of opening. Your application passes a pointer to this list to Navigation Services functions that display Open or Save dialogs. You may create this list dynamically or reference a Translation Manager `'open'` resource.

For more information on the `'open'` resource and the Translation Manager, see the "Translation Manager" chapter in *Inside Macintosh: More Macintosh Toolbox*.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Navigation.h`

NavEventUPP

Defines a universal procedure pointer (UPP) to an application-defined event–handling function.

```
typedef NavEventProcPtr NavEventUPP;
```

Discussion
For more information, see the description of the `NavEventProcPtr` (page 1448) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Navigation.h`

NavObjectFilterUPP

Defines a universal procedure pointer (UPP) to an application-defined filter function.

```
typedef NavObjectFilterProcPtr NavObjectFilterUPP;
```

Discussion
For more information, see the description of the `NavObjectFilterProcPtr` (page 1449) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Navigation.h`

NavPreviewUPP

Defines a universal procedure pointer (UPP) to an application-defined preview function.

```
typedef NavPreviewProcPtr NavPreviewUPP;
```

Discussion
For more information, see the description of the `NavPreviewProcPtr` (page 1450) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Navigation.h`

NavMenuItemSpec

Defines additional items in an Open dialog's Show pop-up menu or a Save dialog's Format pop-up menu.

```
struct NavMenuItemSpec {
    UInt16 version;
    OSType menuCreator;
    OSType menuType;
    Str255 menuItemName;
    char reserved[245];
};
typedef struct NavMenuItemSpec          NavMenuItemSpec;
typedef NavMenuItemSpec *               NavMenuItemSpecArrayPtr;
typedef NavMenuItemSpecArrayPtr *       NavMenuItemSpecArrayHandle;
typedef NavMenuItemSpecArrayPtr         NavMenuItemSpecPtr;
typedef NavMenuItemSpecArrayHandle      NavMenuItemSpecHandle;
```

Fields

`version`

Identifies the version of this structure. Be sure to specify the `kNavMenuItemSpecVersion` constant in this field.

`menuCreator`

A unique value set by your application. Navigation Services passes this value back to your application to identify the application type of the selected menu item.

`menuType`

A unique value set by your application. Navigation Services passes this value back to your application to identify the type of the selected menu item. Values from -1 to 10 are reserved for Navigation Services.

`menuItemName`

The item name that appears in the pop-up menu.

`reserved`

Reserved for future use.

Discussion

For information about file creators and file types, see *Inside Macintosh: Macintosh Toolbox Essentials*.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Navigation.h`

NavContext

An old name for `NavDialogRef`.

Not recommended

```
typedef NavDialogRef NavContext;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Navigation.h`

NavDialogOptions

Contains dialog box configuration settings.

Not recommended

```
struct NavDialogOptions {
    UInt16 version;
    NavDialogOptionFlags dialogOptionFlags;
    Point location;
    Str255 clientName;
    Str255 windowTitle;
    Str255 actionButtonLabel;
    Str255 cancelButtonLabel;
    Str255 savedFileName;
    Str255 message;
    UInt32 preferenceKey;
    NavMenuItemSpecArrayHandle popupExtension;
    char reserved[494];
};
typedef struct NavDialogOptions NavDialogOptions;
```

Fields

`version`
: Identifies the version of this structure. Be sure to specify the `kNavDialogOptionsVersion` constant in this field.

`dialogOptionFlags`
: One of several constants defined by the `NavDialogOptionFlags` data type as described in "Dialog Configuration Options" (page 1471).

`location`
: The upper-left location of the dialog box (in global coordinates). If you set the `dialogOptionFlags` field to `NULL` or set this field to (-1,-1), then the dialog box appears in the same location as when last closed. The size and location of the dialog box is persistent, but defaults to opening in the middle of the main screen if any portion is not visible when opened at the persistent location and size.

`clientName`
: A string that identifies your application in the dialog box window title.

`windowTitle`
: A string that you can provide to override the default window title.

`actionButtonLabel`
: An alternative button title for the dialog box's action button. If you do not specify a title, the button will use the default label (Open or Save, for example.

`cancelButtonLabel`
: An alternative button title for the Cancel button in dialog boxes.

`savedFileName`
: The default filename for a saved file.

`message`
: The string for the banner, or prompt, below the browser list. This message can provide more descriptive instructions for the user. If you don't provide a message string, the browser list expands to fill that area.

`preferenceKey`

An application-defined value that identifies which set of dialog box preferences Navigation Services should use. If your application maintains multiple sets of preferences for a particular type of dialog box, you can determine which set is active by specifying the appropriate value in the `preferenceKey` field. For example, an application may allow one set of preferences when it calls the function `NavGetFile` (page 1438) to open text files and a different set of preferences when opening movie files. If you do not wish to provide a preference key, specify `NULL` for the `preferenceKey` value.

`popupExtension`

A handle to one or more structures of type `NavMenuItemSpec` (page 1461) used to add extra menu items to the Show pop-up menu in an Open dialog box or the Format pop-up menu in Save dialog boxes. Using `NavMenuItemSpec` structures allows your application to add additional document types to be opened or saved, or different ways of saving a file (with or without line breaks, for example).

`reserved`

Reserved for future use.

Carbon Porting Notes

The `NavDialogCreationOptions` (page 1452) structure is the recommended replacement for the `NavDialogOptions` structure. *`NavDialogCreationOptions`* uses CFString objects instead of Pascal strings, thereby adding support for Unicode. In addition, `NavDialogCreationOptions` adds fields for controlling window modality and setting the parent window (necessary for sheets on OS X).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Navigation.h`

Constants

Action State Constants

Let you block certain actions in dialogs.

```
typedef UInt32 NavActionState;
enum {
    kNavNormalState = 0x00000000,
    kNavDontOpenState = 0x00000001,
    kNavDontSaveState = 0x00000002,
    kNavDontChooseState = 0x00000004,
    kNavDontNewFolderState = 0x00000010
};
```

Constants

`kNavNormalState`

Allows all user actions. This is the default state.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavDontOpenState`
Prevents Navigation Services from opening files.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavDontSaveState`
Prevents Navigation Services from saving files.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavDontChooseState`
Prevents Navigation Services from choosing files.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavDontNewFolderState`
Prevents Navigation Services from creating new folders.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

Discussion
The `NavActionState` enumeration defines constants you can specify in the `parms` parameter of the function `NavCustomControl` (page 1429) in order to block certain actions in Navigation Services dialogs. When you specify these constants, you must also specify the `kNavSetActionState` constant in the `selector` parameter of the `NavCustomControl` function.

Version Notes
These constants are only available in Navigation Services 2.0 or later.

Custom Control Settings

Provide constants that allow you to control various aspects of the active dialog.

```
typedef SInt32 NavCustomControlMessage;
enum {
    kNavCtlShowDesktop = 0,
    kNavCtlSortBy = 1,
    kNavCtlSortOrder = 2,
    kNavCtlScrollHome = 3,
    kNavCtlScrollEnd = 4,
    kNavCtlPageUp = 5,
    kNavCtlPageDown = 6,
    kNavCtlGetLocation = 7,
    kNavCtlSetLocation = 8,
    kNavCtlGetSelection = 9,
    kNavCtlSetSelection = 10,
    kNavCtlShowSelection = 11,
    kNavCtlOpenSelection = 12,
    kNavCtlEjectVolume = 13,
    kNavCtlNewFolder = 14,
    kNavCtlCancel = 15,
    kNavCtlAccept = 16,
    kNavCtlIsPreviewShowing = 17,
    kNavCtlAddControl = 18,
    kNavCtlAddControlList = 19,
    kNavCtlGetFirstControlID = 20,
    kNavCtlSelectCustomType = 21,
    kNavCtlSelectAllType = 22,
    kNavCtlGetEditFileName = 23,
    kNavCtlSetEditFileName = 24,
    kNavCtlSelectEditFileName = 25,
    kNavCtlBrowserSelectAll = 26,
    kNavCtlGotoParent = 27,
    kNavCtlSetActionState = 28,
    kNavCtlBrowserRedraw = 29,
    kNavCtlTerminate = 30
};
```

Constants

`kNavCtlShowDesktop`

Tells Navigation Services to change the browser list location to the desktop.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavCtlSortBy`

Alerts Navigation Services that your application is setting a sort key in the browser list. In addition to the `kNavCtlSortBy` constant, your application passes one of the `NavSortKeyField` constants in the `parms` parameter of the function `NavCustomControl` (page 1429). For a description of the `NavSortKeyField` constants, see "File Sorting Constants" (page 1478).

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavCtlSortOrder`

Alerts Navigation Services that your application is setting sort order, either ascending or descending, in the browser list. In addition to passing the `kNavCtlSortOrder` constant, your application must pass one of the `NavSortOrder` constants in the `parms` parameter of the `NavCustomControl` function. For a description of the `NavSortOrder` constants, see "Sort Order Constants" (page 1482).

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavCtlScrollHome`
Tells Navigation Services to scroll the browser to the top of the file list.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavCtlScrollEnd`
Tells Navigation Services to scroll the browser to the bottom of the file list.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavCtlPageUp`
Tells Navigation Services to scroll the browser up one page length as a result of the user clicking the scroll bar above the scroll box.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavCtlPageDown`
Tells Navigation Services to scroll the browser down one page length as a result of the user clicking the scroll bar below the scroll box.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavCtlGetLocation`
Tells Navigation Services to return the current location. Navigation Services reports the current location by setting a pointer to an `AEDesc` structure in the `param` field of the structure of type `NavCBRec` (page 1451) that you specified in your event-handling function.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavCtlSetLocation`
Tells Navigation Services that your application wishes to set the location being viewed in the browser list. In addition to specifying the `kNavCtlSetLocation` constant, your application passes a pointer to an `AEDesc` structure describing the new location in the `parms` parameter of the `NavCustomControl` function.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavCtlGetSelection`
Tells Navigation Services to return the selected item or items in the browser. When you specify this constant, Navigation Services returns a pointer to an `AEDesc` structure describing the selected item(s) in the `param` field of the structure of type `NavCBRec` (page 1451) that you specified in your event-handling function. If the user deselects the current selection, the `AEDescList` returned by Navigation Services contains an empty reference. You can account for this case by using the function `AECountItems` and checking for a zero count.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavCtlSetSelection`

Tells Navigation Services to change the browser list selection. In addition to specifying the `kNavCtlSetSelection` constant, your application must pass a pointer to an `AEDescList` structure describing the selection in the `parms` parameter of the `NavCustomControl` function. If you want to deselect the current selection without making a new selection, pass `NULL` for the pointer. Note: If you specify this constant, Navigation Services notifies your event-handling function by setting the `kNavCBSelectEntry` constant twice; once when the previous selection is deselected, and once when the new selection is made.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavCtlShowSelection`

Tells Navigation Services to make the current selection visible in the browser list if the selection has been scrolled out of sight by the user.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavCtlOpenSelection`

Tells Navigation Services to open the current selection.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavCtlEjectVolume`

Tells Navigation Services to eject a volume. In addition to specifying this constant, you pass a pointer to the volume reference number (`vRefNum`) of the volume to be ejected in the `parms` parameter of the `NavCustomControl` function.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavCtlNewFolder`

Tells Navigation Services to create a new folder in the current browser location. In addition to specifying the `kNavCtlNewFolder` constant, your application passes a string representing the name of the new folder in the `parms` parameter of the `NavCustomControl` function.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavCtlCancel`

Tells Navigation Services to dismiss the Open or Save dialog as if the user had pressed the Cancel button.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavCtlAccept`

Tells Navigation Services to close the Open or Save dialog as if the user had pressed the Open or Save button. Navigation Services does not act on this constant if there is no current selection.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavCtlIsPreviewShowing`

Asks Navigation Services if the preview area is currently available. If you specify this constant, Navigation Services sets a pointer to a Boolean value in the `param` field of the `NavCBRec` (page 1451) structure that you specified in your event-handling function. This value is `true` if the preview area is available, `false` otherwise.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavCtlAddControl`

Tells Navigation Services to add one application-defined control to Open or Save dialogs. In addition to sending this message, your application passes a control handle in the `parms` parameter of the `NavCustomControl` function. Design the control in local coordinates.

Note: To avoid any unnecessary flickering or redrawing, ensure the control is initially invisible before specifying this constant. You may set the control to visible after Navigation Services supplies the `kNavCBStart` constant, described in "Event Messages" (page 1475), in the `param` field of the `NavCBRec` (page 1451) structure. If the user resizes the dialog, your application must move the control because it is not maintained by Navigation Services. If you use the `kNavCtlAddControlList` constant (described next) and you supply a `'DITL'` resource, you avoid the need to move the control yourself.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavCtlAddControlList`

Tells Navigation Services to add a list of application-defined dialog items to Open or Save dialogs. In addition to specifying this constant, your application passes a handle to a dialog item list or `'DITL'` resource in the `parms` parameter of the `NavCustomControl` function. Design the `'DITL'` resource in local coordinates. Navigation Services adds the custom items relative to the upper left corner of the customization area. If the user resizes the dialog, your custom items are moved automatically.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavCtlGetFirstControlID`

Asks Navigation Services to help you identify the first custom control in the dialog, in order to determine which custom control item was selected by the user. Navigation Services returns a pointer to a 16-bit integer that indicates the item number of the first custom control in the param field of the structure of type `NavCBRec` (page 1451) that you specified in your event-handling function. In your event-handling function, use the Dialog Manager function `FindDialogItem` to find out which item was selected. The `FindDialogItem` function returns 0 for the first item, 1 for the second and so on. To get the proper item number, add 1 to the `FindDialogItem` function result. The Open or Save dialog's standard controls precede yours, so use the formula `(itemHit - yourFirstItem + 1)` to determine which of your items was selected. Your application should not depend on any hardcoded value for the number of items, since this value is likely to change in the future.

Be sure to test the result from `FindDialogItem` to ensure that it describes a control that you defined. Your application must not respond to any controls that do not belong to it.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavCtlSelectCustomType`
Tells Navigation Services to set one of your custom menu items in the Show pop-up menu or the Format pop-up menu as the default selection. This is useful if you want to override the default pop-up menu selection. In addition to specifying this constant, pass a pointer to a `NavMenuItem` structure in the `parms` parameter of the `NavCustomControl` function. This structure describes the item you wish to have selected.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavCtlSelectAllType`
Tells Navigation Services to override the default menu item in the Type pop-up menu. By specifying one of the `NavPopupMenuItem` constants, described in "Menu Item Selection Constants" (page 1479), in the `parms` parameter of the `NavCustomControl` function, you can set the default item to All [AppName] Documents, All Readable Documents or All Documents.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavCtlGetEditFileName`
Tells Navigation Services to return the name of the file to be saved by a file-saving function. This would be useful if you wanted to automatically add an extension to the filename, for example. When you send this message, the `parms` parameter of the `NavCustomControl` function returns a `StringPtr` to a Pascal string containing the filename. Note that in Carbon, you can use the `NavDialogGetSaveFileName` (page 1431) function to obtain a Unicode string containing the filename.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavCtlSetEditFileName`
Tells Navigation Services that your application wishes to set the name of the file to be saved by a file-saving function. Your application normally specifies the `KNavCtlSetEditFileName` constant after modifying the filename obtained by specifying the `kNavCtlGetEditFileName` constant. In addition to specifying the `kNavCtlSetEditFileName` constant, your application passes a `StringPtr` to a Pascal string containing the filename in the `parms` parameter of the `NavCustomControl` function. Note that you can set the filename with a Unicode string by calling the `NavDialogSetSaveFileName` (page 1435) function.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavCtlSelectEditFileName`
(Navigation Services 1.1 or later) Tells Navigation Services to display the name of the file to be saved by the function with some or all of the filename string highlighted for selection. In addition to specifying the `kNavCtlSelectEditFileName` constant, your application passes a Control Manager structure of type `ControlEditTextSelectionRec` in the `parms` parameter of the `NavCustomControl` function in order to specify which part of the filename string to highlight. For more information on the `ControlEditTextSelectionRec` structure, see *Inside Mac OS X: Control Manager Reference*.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavCtlBrowserSelectAll`
(Navigation Services 2.0 or later.) Tells Navigation Services to select all files in the browser list.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavCtlGotoParent`
(Navigation Services 2.0 or later.) Tells Navigation Services to navigate to the parent folder or volume of the current selection.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavCtlSetActionState`
(Navigation Services 2.0 or later.) Prevents Navigation Services from handling certain user actions, such as opening or saving files. This is useful if you want to prevent the dismissal of a dialog until certain conditions are met, for example. Specify which actions to prevent by passing one or more of the constants defined by the `NavActionState` enumeration, described in "Action State Constants" (page 1464).

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavCtlBrowserRedraw`
(Navigation Services 2.0 or later.) Tells Navigation Services to refresh the browser list.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavCtlTerminate`
(Navigation Services 2.0 or later.) Tells Navigation Services to dismiss the current dialog. This constant is similar to `kNavCtlCancel`, except that using `kNavCtlTerminate` does not return an error code.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

Discussion
The `NavCustomControlMessage` data type defines constants that your application can pass in the `selector` parameter of the function `NavCustomControl` (page 1429) to control various aspects of the active dialog.

Dialog Configuration Options

Specify dialog configuration options.

```
typedef UInt32 NavDialogOptionFlags;
enum {
    kNavDefaultNavDlogOptions = 0x000000E4,
    kNavNoTypePopup = 0x00000001,
    kNavDontAutoTranslate = 0x00000002,
    kNavDontAddTranslateItems = 0x00000004,
    kNavAllFilesInPopup = 0x00000010,
    kNavAllowStationery = 0x00000020,
    kNavAllowPreviews = 0x00000040,
    kNavAllowMultipleFiles = 0x00000080,
    kNavAllowInvisibleFiles = 0x00000100,
    kNavDontResolveAliases = 0x00000200,
    kNavSelectDefaultLocation = 0x00000400,
    kNavSelectAllReadableItem = 0x00000800,
    kNavSupportPackages = 0x00001000,
    kNavAllowOpenPackages = 0x00002000,
    kNavDontAddRecents = 0x00004000,
    kNavDontUseCustomFrame = 0x00008000,
    kNavDontConfirmReplacement = 0x00010000,
    kNavPreserveSaveFileExtension = 0x00020000
};
```

Constants

`kNavDefaultNavDlogOptions`

Tells Navigation Services to use default configuration options. These default options include:

- no custom control titles
- no banner or prompt message
- automatic resolution of aliases
- support for file previews
- no display of invisible file objects
- support for multiple file selection
- support for stationery
- no package support
- all chosen items added to Recent list
- customization area is framed

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavNoTypePopup`

Tells Navigation Services not to display the Show pop-up menu in the Open dialog or the Format pop-up menu in the Save dialog.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavDontAutoTranslate`

Tells Navigation Services not to do automatic file translation. Normally a file chosen in an Open dialog that requires translation is automatically translated. Navigation Services informs your application that a file needs translating by setting the `translationNeeded` field of the `NavReplyRecord` (page 1458) structure to `true`. A translation specification array specified in the `fileTranslation` field of the `NavReplyRecord` structure contains the associated translation specification records. When you set the `kNavDontAutoTranslate` flag, your application is responsible for translation, either by calling the function `NavTranslate` or by performing the translation itself.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavDontAddTranslateItems`

Tells Navigation Services not to display file translation options in the Show pop-up menu.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavAllFilesInPopup`

Tells Navigation Services to add a pop-up menu item called All Documents, so the user can see a display of all files in the current directory.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavAllowStationery`

Tells Navigation Services to display a Stationery Option command in the Format pop-up menu of Save dialogs, so users can choose to save a file as a document or as stationery. This is a default option.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavAllowPreviews`

Tells Navigation Services to provide previews, when available, of selected files. This is a default option.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavAllowMultipleFiles`

Tells Navigation Services to allow users to select and open multiple files in the browser list by shift-clicking or using the Select All command. If you don't specify this constant, users can select multiple files for drag-and-drop operations, but the default button (normally titled Open) is disabled when multiple items are selected. Note that the user cannot add folders or volumes to a multiple selection.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavAllowInvisibleFiles`

Tells Navigation Services to show invisible file objects in the browser list.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavDontResolveAliases`

Tells Navigation Services not to resolve any alias selected by the user. If the user selects an alias with this option set, the file system specification returned by Navigation Services designates the alias file instead of its referenced original.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavSelectDefaultLocation`

Tells Navigation Services to select the default location in the browser list. By default, Navigation Services will open the browser list with the default location displayed, not selected. For example, if you define the System Folder as the default location and specify the `kNavSelectDefaultLocation` constant, the System Folder appears as the current selection in the browser list. Without this constant, the browser list displays the contents of the System Folder.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavSelectAllReadableItem`

Tells Navigation Services to show All Readable Documents as the default selection in the Show pop-up menu when the Open dialog is first displayed. If you do not specify this constant, Navigation Services shows the All [AppName] Documents menu item as the default selection in the Show pop-up menu when the Open dialog is first displayed.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavSupportPackages`

(Available in Navigation Services 2.0 and later.) Tells Navigation Services to allow packages to be displayed in the browser list.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavAllowOpenPackages`

(Available in Navigation Services 2.0 and later.) Tells Navigation Services to allow packages to be opened and navigated in the browser list.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavDontAddRecents`

(Available in Navigation Services 2.0 and later.) Tells Navigation Services not to add file objects to the Recent list after a dialog is closed. This is useful if you want to allow users to choose long lists of items without cluttering up the Recent list.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavDontUseCustomFrame`

(Available in Navigation Services 2.0 and later.) Tells Navigation Services not to draw a bevelled border around the customization area. The border is drawn by default, so you must specify this constant if you want to turn it off.

Note: Keep in mind that turning off the border may affect the placement of any controls you create in the customization area. This means your controls may appear differently in different versions of Navigation Services.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavDontConfirmReplacement`
(Mac OS X only.) Tells Navigation Services not to display an alert when the user attempts to save a file over another file with the same name.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavPreserveSaveFileExtension`
(Available in Navigation Services 3.1 and later.) Tells Navigation Services that the extension in the default filename should be preserved between dialog invocations and is initially hidden.

Available in Mac OS X v10.1 and later.

Declared in `Navigation.h`.

Discard Changes Actions

Describe the user response to a Discard Changes dialog.

```
typedef UInt32 NavAskDiscardChangesResult;
enum {
    kNavAskDiscardChanges = 1,
    kNavAskDiscardChangesCancel = 2
};
```

Constants

`kNavAskDiscardChanges`
User clicked the Okay button.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavAskDiscardChangesCancel`
User clicked the Cancel button.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

Discussion

After you display a dialog created using the `NavCreateAskDiscardChangesDialog` (page 1416) function, your application determines the result of the function call by checking the `eventData` field of a structure of type `NavCBRec` (page 1451) for one of the constants defined by the `NavAskDiscardChangesResult` data type.

Event Messages

Define messages sent to your event-handling function.

```
typedef SInt32 NavEventCallbackMessage;
enum {
    kNavCBEvent = 0,
    kNavCBCustomize = 1,
    kNavCBStart = 2,
    kNavCBTerminate = 3,
    kNavCBAdjustRect = 4,
    kNavCBNewLocation = 5,
    kNavCBShowDesktop = 6,
    kNavCBSelectEntry = 7,
    kNavCBPopupMenuSelect = 8,
    kNavCBAccept = 9,
    kNavCBCancel = 10,
    kNavCBAdjustPreview = 11,
    kNavCBUserAction = 12,
    kNavCBOpenSelection = (long)0x80000000
};
```

Constants

`kNavCBEvent`

Tells your application that an event has occurred (including an idle event), which provides an opportunity for your application to track controls, update other windows, and so forth. Your application can obtain the event record describing this event from the `event` field of the `NavCBRec` (page 1451) structure. The `kNavCBEvent` constant is the only message that needs to be processed by most applications that do not customize Open and Save dialogs.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavCBCustomize`

Tells your application to supply a dialog customization request. The `customRect` field of the `NavCBRec` (page 1451) structure defines a rectangle in the local coordinates of the dialog; the top-left coordinates define the anchor point for a customization rectangle. If you want to customize the dialog, your application responds to the `kNavCBCustomize` message by setting a value in the `customRect` field that completes the dimensions of the customization rectangle. After your application responds, Navigation Services inspects the `customRect` field to determine if the requested dimensions result in a dialog that can fit on the screen. If the dimensions are too large, then Navigation Services responds by setting the rectangle to the largest size that the screen can accommodate. Your application can continue to "negotiate" by examining the `customRect` field and requesting a different size until Navigation Services provides an acceptable rectangle value, at which time you should create your custom control or item list. The minimum size for the customization area is 400 pixels wide by 40 pixels high.

Note: Don't add new dialog items until your application receives the `kNavCBStart` event message constant.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavCBStart`

Tells your application that a dialog is ready to be displayed. After receiving the `kNavCBCustomize` event message constant, your event-handling function should wait for the `kNavCBStart` event message constant to ensure that your application can safely add dialog items. No additional data is provided to your application with this constant.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavCBTerminate`

Tells your application that the dialog is about to be closed, which means you must remove any user-interface items that were created in response to the `kNavCBStart` message. You can determine which user action closed the dialog by checking the `userAction` field of the `NavCBRec` (page 1451) structure.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavCBAdjustRect`

Tells your application that the dialog has been resized and the customization rectangle has been accordingly resized. Use the `customRect` field from the `NavCBRec` (page 1451) structure to determine the new customization rectangle size. Your application does not need to offset the controls; Navigation Services moves them automatically. Your application is responsible for any redrawing of the controls or handling events beyond moving the controls, however.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavCBNewLocation`

Tells your application that a new location is being viewed in the dialog. The `param` field of the `NavCBRec` (page 1451) structure contains a pointer to an `AEDesc` structure of type `'typeFSS'` describing the new location. This pointer is valid only during the execution of your event-handling function.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavCBShowDesktop`

Tells your application that the Open or Save dialog is showing the desktop view, consisting of the composite of all desktop folders from all mounted volumes. The `param` field of the `NavCBRec` (page 1451) structure contains a pointer to an `AEDescList` structure identifying the desktop location. This pointer is valid only during the execution of your event-handling function.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavCBSelectEntry`

Tells your application that an entry in the browser list has been selected or deselected by the user. The `param` field of the `NavEventDataInfo` structure contains a pointer to an `AEDescList` record of type `'typeFSS'` identifying the current selection. If the user deselects the current selection, the `AEDescList` record contains an empty reference. This pointer is valid only during the execution of your event-handling function.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavCBPopupMenuSelect`

Tells your application that a selection was made from the Open dialog's Show pop-up menu or Save dialog's Format pop-up menu. The`NavCBRec.eventData.eventDataParms.param` field contains a pointer to a `NavMenuItemSpec` structure describing the pop-up menu item selected. If the dialog was created using the Carbon-only `NavCreateXXXDialog` APIs, then the `menuType` field of the `NavMenuItemSpec` structure is set to the index into the client's CFArray of `popupExtension` strings in the `NavDialogCreationOptions` (page 1452) structure. This data is valid only during the execution of your event-handling function.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavCBAccept`
Tells your application that the user has pressed the Accept button.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavCBCancel`
Tells your application that the user has pressed the Cancel button.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavCBAdjustPreview`
Tells your application that the user has toggled the preview area on or off. The `param` field of the `NavCBRec` (page 1451) structure contains a pointer to a Boolean value of `true` if the preview area is toggled on and `false` if toggled off. This information is useful if your application creates custom controls in the preview area.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavCBUserAction`
(Available only in CarbonLib 1.1 and later and in Mac OS X version 10.0 and later.) Tells your application that the user has taken an action. To determine which action the user took, call the `NavDialogGetUserAction` (page 1432) function.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavCBOpenSelection`
(Navigation Services 2.0 or later.) Tells your application that the user has opened a file or chosen a file object. After detecting this constant, you can call the `NavCustomControl` (page 1429) function and specify one of the `NavActionState` constants, described in "Action State Constants" (page 1464), in order to block the opening or choosing action.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

File Sorting Constants

Let you specify a sort key for how files display in browser.

```
typedef UInt16 NavSortKeyField;
enum {
    kNavSortNameField = 0,
    kNavSortDateField = 1
};
```

Constants

`kNavSortNameField`
Sort by filename.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavSortDateField`
Sort by modification date.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

Discussion
Your application can determine the sort key for displayed files by passing the `kNavCtlSortBy` constant, described in "Custom Control Settings" (page 1465), in the `selector` parameter of the function `NavCustomControl` (page 1429), and passing one of the constants defined in the `NavSortKeyField` data type in the `parms` parameter of the `NavCustomControl` function.

Generic File Signature Constant

Defines a generic creator code.

```
enum {
    kNavGenericSignature = '****'
};
```

Constants
`kNavGenericSignature`
Tells Navigation Services to display all files of a specified type, regardless of the file's creator code.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

Discussion
To specify that your application can open all files of a given type (such as `'TEXT'`, for example), specify the `kNavGenericSignature` constant in the `componentSignature` field of the structure of type `NavTypeList` (page 1460) that you pass to a function that creates a file-opening dialog, such as `NavCreateGetFileDialog` (page 1423). You can also pass this constant in the `inFileCreator` parameter of the function `NavCreatePutFileDialog` (page 1426) in order to override the types of files appearing in the Format pop-up menu.

Version Notes
Added in Navigation Services 2.0

Menu Item Selection Constants

Let you specify the default selection in the Show pop-up menu.

```
typedef UInt16 NavPopupMenuItem;
enum {
    kNavAllKnownFiles = 0,
    kNavAllReadableFiles = 1,
    kNavAllFiles = 2
};
```

Constants
`kNavAllKnownFiles`
Tells Navigation Services to display all files identified as readable by your application.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavAllReadableFiles`
Tells Navigation Services to display all files identified as readable or translatable by your application.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavAllFiles`
Tells Navigation Services to display all files.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

Discussion
To set the default selection for the Show pop-up menu of an Open dialog box, your application passes the `kNavCtlSelectAllType` constant, described in "Custom Control Settings" (page 1465), in the `selector` parameter of the function `NavCustomControl` (page 1429) and passes one of the constants defined in the `NavPopupMenuItem` data type in the `parms` parameter of the `NavCustomControl` function.

Object Filtering Constants

Inform you which part of a dialog contains object being filtered.

```
typedef SInt16 NavFilterModes;
enum {
    kNavFilteringBrowserList = 0,
    kNavFilteringFavorites = 1,
    kNavFilteringRecents = 2,
    kNavFilteringShortCutVolumes = 3,
    kNavFilteringLocationPopup = 4
};
```

Constants
`kNavFilteringBrowserList`
The browser list contains the object being filtered.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavFilteringFavorites`
The Favorites pop-up menu contains the object being filtered.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavFilteringRecents`
The Recent pop-up menu contains the object being filtered.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavFilteringShortCutVolumes`
The Shortcuts pop-up menu contains the object being filtered.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavFilteringLocationPopup`
The object being filtered is the path described by the Location menu.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

Discussion
Navigation Services passes one of the constants defined by the `NavFilterModes` data type to the `filterMode` parameter of your application-defined filter function to tell your application whether the browser list or one of the navigation option pop-up menus contains the object currently being filtered.

Save Changes Actions

Describe the user response to a Save Changes dialog.

```
typedef UInt32 NavAskSaveChangesResult;
enum {
    kNavAskSaveChangesSave = 1,
    kNavAskSaveChangesCancel = 2,
    kNavAskSaveChangesDontSave = 3
};
```

Constants
`kNavAskSaveChangesSave`
User clicked the Save button.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavAskSaveChangesCancel`
User clicked the Cancel button.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavAskSaveChangesDontSave`
User clicked the Don't Save button.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

Discussion
When you create a Save dialog using the function `NavCreatePutFileDialog` (page 1426), you obtain the user's response by calling `NavDialogGetReply` (page 1430), specifying a `NavReplyRecord` (page 1458) structure in the `outReply` parameter. On completion, this structure contains a value represented by one of the constants defined by the `NavAskSaveChangesResult` data type.

Save Changes Requests

Describe the condition that prompts a Save Changes dialog.

```
typedef UInt32 NavAskSaveChangesAction;
enum {
    kNavSaveChangesClosingDocument = 1,
    kNavSaveChangesQuittingApplication = 2,
    kNavSaveChangesOther = 0
};
```

Constants

`kNavSaveChangesClosingDocument`
Requests a Save Changes alert that asks the user whether to save changes when closing a document.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavSaveChangesQuittingApplication`
Requests a Save Changes alert that asks the user whether to save changes when quitting your application.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavSaveChangesOther`
Requests a Save Changes alert that asks the user whether to save changes at some time other than closing or quitting. This is useful when your application prompts the user to save documents at timed intervals, for example.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

Discussion

Your application requests a Save Changes alert by specifying one of the following constants, defined by the `NavAskSaveChangesAction` data type, in the `inAction` parameter of the function `NavCreatePutFileDialog` (page 1426).

Sort Order Constants

Let you specify ascending or descending sort order in the browser.

```
typedef UInt16 NavSortOrder;
enum {
    kNavSortAscending = 0,
    kNavSortDescending = 1
};
```

Constants

`kNavSortAscending`
Sort in ascending order.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavSortDescending`
Sort in descending order.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

Discussion
Your application can specify the sort order for displayed files by passing the `kNavCtlSortOrder` constant in the `selector` parameter of the function `NavCustomControl` (page 1429) and passing one of the constants defined in the `NavSortOrder` data type in the `parms` parameter of the `NavCustomControl` function.

Translation Options

Let you specify how files are translated.

```
typedef UInt32 NavTranslationOptions;
enum {
    kNavTranslateInPlace = 0,
    kNavTranslateCopy = 1
};
```

Constants

`kNavTranslateInPlace`
Tells Navigation Services to replace the source file with the translation. This setting is the default for Save dialogs.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavTranslateCopy`
Tells Navigation Services to create a translated copy of the source file. This setting is the default for Open dialogs. The `NavCompleteSave` function always uses this setting under automatic translation.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

Discussion
Your application passes one of the `NavTranslationOptions` constants to the `howToTranslate` parameter to specify how files are to be translated by the function `NavTranslateFile` (page 1445).

User Actions

```
typedef UInt32 NavUserAction;
enum {
    kNavUserActionNone = 0,
    kNavUserActionCancel = 1,
    kNavUserActionOpen = 2,
    kNavUserActionSaveAs = 3,
    kNavUserActionChoose = 4,
    kNavUserActionNewFolder = 5,
    kNavUserActionSaveChanges = 6,
    kNavUserActionDontSaveChanges = 7,
    kNavUserActionDiscardChanges = 8,
    kNavUserActionReviewDocuments = 9,
    kNavUserActionDiscardDocuments = 10
};
```

Constants

`kNavUserActionNone`
The dialog is still running or was terminated programmatically.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavUserActionCancel`
The user pressed the Cancel button.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavUserActionOpen`
The user pressed the Open button in a file-opening dialog.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavUserActionSaveAs`
The user pressed the Save button in a file-saving dialog.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavUserActionChoose`
The user pressed the Choose button in a Choose dialog.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavUserActionNewFolder`
The user pressed the New Folder button in a New Folder dialog and Navigation Services created a folder.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavUserActionSaveChanges`
The user pressed the Save button in a Save Changes dialog.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavUserActionDontSaveChanges`
The user pressed the Don't Save button in a Save Changes dialog.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavUserActionDiscardChanges`
The user pressed the Discard button in a Discard Changes dialog.

Available in Mac OS X v10.0 and later.

Declared in `Navigation.h`.

`kNavUserActionReviewDocuments`
The user clicked the Review Unsaved button in the Review Documents dialog (used only on Mac OS X).

Available in Mac OS X v10.1 and later.

Declared in `Navigation.h`.

`kNavUserActionDiscardDocuments`
The user clicked the Discard Changes button in the Review Documents dialog (used only on Mac OS X).

Available in Mac OS X v10.1 and later.

Declared in `Navigation.h`.

Discussion
Navigation Services passes one of the constants defined by the `NavUserAction` enumeration to your application in the `userAction` field of a `NavCBRec` (page 1451) structure in order to indicate which action was taken by the user during a Navigation Services dialog.

NavDialogCreationOptions Version Constant

Represents the current version of the `NavDialogCreationOptions` structure.

```
enum {
        kNavDialogCreationOptionsVersion = 0
};
```

NavCBRec Version Constant

Represents the current version of the `NavCBRec` structure.

```
enum {
    kNavCBRecVersion = 1
};
```

NavFileOrFolder Version Constant

Represents the current version of the `NavFileOrFolder` structure.

```
enum {
    kNavFileOrFolderVersion = 1
};
```

NavMenuItemSpec Version Constant

Represents the current version of the `NavMenuItemSpec` structure.

```
enum {
    kNavMenuItemSpecVersion = 0
};
```

NavReplyRecord Version Constant

Represents the current version of the `NavReplyRecord` structure.

```
enum {
    kNavReplyRecordVersion = 2
};
```

Result Codes

The table below shows the most common result codes returned by Navigation Services.

Result Code	Value	Description
`kNavWrongDialogStateErr`	-5694	Dialog is not in correct state for requested operation (it must be running but is not, or vice versa. Available in Mac OS X v10.0 and later.
`kNavWrongDialogClassErr`	-5695	Requested operation is not valid for this type of dialog. Available in Mac OS X v10.0 and later.
`kNavInvalidSystemConfigErr`	-5696	One or more Navigation Services–required system components is missing or out of date. Available in Mac OS X v10.0 and later.
`kNavCustomControlMessageFailedErr`	-5697	Navigation Services did not accept a control message sent by your application. Available in Mac OS X v10.0 and later.
`kNavInvalidCustomControlMessageErr`	-5698	Your application sent an invalid custom control message. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
kNavMissingKindStringErr	-5699	No kind strings were provided to describe your application's native file types. Available in Mac OS X v10.0 and later.

Notification Manager Reference

Framework: Carbon/Carbon.h

Declared in Notification.h

Overview

The Notification Manager provides a notification service. It allows software running in the background (or otherwise unseen by the user) to communicate information to the user. For example, applications that manage lengthy background tasks (such as printing many documents or transferring large amounts of data to other machines) might need to inform the user that the operation is complete. These applications cannot use the standard methods of communicating with the user, such as alert or dialog boxes, because such windows might easily be obscured by the windows of other applications. Moreover, even if those windows are visible, the background application cannot be certain that the user is aware of the change. A more reliable method is needed to manage the communication between a background application and the user, who might be awaiting the completion of the background task while running some other application in the foreground.

In the same way, relatively invisible operations such as Time Manager tasks, VBL tasks, or device drivers might need to inform the user that some previously started routine is complete or perhaps that some error has rendered further execution undesirable or impossible.

In all these cases, the communication generally needs to occur in one direction only, from the background application (or task, or driver) to the user. The Notification Manager, included in system software versions 6.0 and later, allows you to alert the user by posting a notification, which is an audible or visible indication that your application (or other piece of software) requires the user's attention. You post a notification by issuing a notification request to the Notification Manager, which places your request in a queue. When your request reaches the top of the queue, the Notification Manager posts a notification to the user.

You can request three types of notification:

- **Polite notification.** A small icon blinks, by periodically alternating with the Apple menu icon (the Apple logo) or the Application menu icon in the menu bar.
- **Audible notification.** The Sound Manager plays the system alert sound or a sound contained in an 'snd ' resource.
- **Alert notification.** An alert box containing a short message appears on the screen. The user must dismiss the alert box (by clicking the OK button) before foreground processing can continue.

These types of notification are not mutually exclusive. For example, an application can request both audible and alert notifications. Moreover, if the requesting software is listed in the Application menu (and hence represents a process that is loaded into memory), you can instruct the Notification Manager to place a

diamond-shaped mark next to the name of the requesting process. The mark is usually intended to prompt the user to switch the marked application into the foreground. Finally, you can request that the Notification Manager execute a notification response procedure, which is executed as the final step in a notification.

Functions

DisposeNMUPP

Releases a universal procedure pointer (UPP) to a response callback function.

```
void DisposeNMUPP (
   NMUPP userUPP
);
```

Parameters

userUPP

A valid UPP, created by calling the function `NewNMUPP` (page 1490).

Availability

CarbonLib 1.0 and later. Mac OS X version 10.0 and later.

Declared In

`Notification.h`

InvokeNMUPP

Invokes a response callback function.

```
void InvokeNMUPP (
   NMRecPtr nmReqPtr,
   NMUPP userUPP
);
```

Parameters

nmReqPtr

A pointer to a notification request. For information about defining a notification request, see `NMRec` (page 1492).

userUPP

A response callback UPP, created by calling the function `NewNMUPP` (page 1490).

Availability

CarbonLib 1.0 and later. Mac OS X version 10.0 and later.

Declared In

`Notification.h`

NewNMUPP

Creates a universal procedure pointer (UPP) to a response callback function.

```
NMUPP NewNMUPP (
   NMProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to a response callback function. For more information about response callbacks, see `NMProcPtr` (page 1492).

Return Value

A new UPP. When you no longer need this UPP, you should call `DisposeNMUPP` (page 1490) to release it.

Availability

CarbonLib 1.0 and later. Mac OS X version 10.0 and later.

Declared In

`Notification.h`

NMInstall

Installs a notification request in the notification queue.

```
OSErr NMInstall (
   NMRecPtr nmReqPtr
);
```

Parameters

request

A pointer to a notification request record that the caller provides.

Return Value

A result code. If the result is non-zero, the Notification Manager cannot install the request because it contains invalid information. See "Notification Manager Result Codes" (page 1494).

Availability

CarbonLib 1.0 and later. Mac OS X version 10.0 and later.

Not available to 64-bit applications.

Declared In

`Notification.h`

NMRemove

Removes a notification request from the notification queue.

```
OSErr NMRemove (
   NMRecPtr nmReqPtr
);
```

Parameters

nmReqPtr

A pointer to a notification request record.

Return Value
A result code. If the result is non-zero, the Notification Manager cannot remove the request. See "Notification Manager Result Codes" (page 1494).

Availability
CarbonLib 1.0 and later. Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`Notification.h`

Callbacks

NMProcPtr

Defines a pointer to a response callback function. Your response callback function is executed as the final stage of a notification.

```
typedef void (*NMProcPtr) (
    NMRecPtr request
);
```

If you name your function `MyNMProc`, you would declare it like this:

```
void MyNMProc (
    NMRecPtr request
);
```

Parameters

request
A pointer to a notification request record.

Discussion
The `nmResp` field of the notification record defines a response function executed as the final stage of a notification. If no processing is necessary in response to the notification, then you can supply the value `NULL` in that field. If you supply a UPP to your own response function in the `nmResp` field, the Notification Manager passes it one parameter, a pointer to your notification request.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Notification.h`

Data Types

NMRec

Defines the record format of a notification request.

```
struct NMRec {
    QElemPtr qLink;
    short qType;
    short nmFlags;
    long nmPrivate;
    short nmReserved;
    short nmMark;
    Handle nmIcon;
    Handle nmSound;
    StringPtr nmStr;
    NMUPP nmResp;
    long nmRefCon;
};
typedef struct NMRec NMRec;
typedef NMRec * NMRecPtr;
```

Fields

`qLink`
: This field is used internally by the Notification Manager. You do not need to assign a value to this field.

`qType`
: Indicates the type of queue. You should assign the value `nmType` to this field.

`nmFlags`
: Reserved.

`nmPrivate`
: Reserved.

`nmReserved`
: Reserved.

`nmMark`
: In Mac OS, the item to mark in the Application menu. If you do not want to display a mark, assign 0 to this field.

  For additional information about the use of this field in Mac OS X, see the discussion below.

`nmIcon`
: In Mac OS, a handle to a small icon that is to blink periodically in the menu bar. If you do not want to display an icon, assign `NULL` to this field.

`nmSound`
: In Mac OS, a handle to a sound resource to be played. If you do not want to play a sound, assign `NULL` to this field.

`nmStr`
: A string to appear in the notification window. In Mac OS, if you do not want a notification window you should assign `NULL` to this field.

`nmResp`
: A UPP to a response callback function. If no processing is necessary in response to the notification, assign the value `NULL` to this field.

`nmRefCon`
: A long integer for private use by your application.

Discussion

In Mac OS X version 10.0, only the alert notification is supported.

In Mac OS X version 10.1 and later, mark and alert notifications are both supported. If you set the `nmMark` field to a non-zero value, your process icon will bounce in the dock.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Notification.h`

Result Codes

The most common result codes returned by the Notification Manager are listed in the table below.

Result Code	Value	Description
`nmTypErr`	-299	The notification request contains an invalid queue type. Available in Mac OS X v10.0 and later.

Gestalt Constants

You can check for version and feature availability information by using the Notification Manager selectors defined in the Gestalt Manager. For more information, see *Inside Mac OS X: Gestalt Manager Reference*.

Scrap Manager Reference (Not Recommended)

Framework: Carbon/Carbon.h

Declared in Scrap.h

Overview

Important: The Scrap Manager is deprecated in Mac OS X version 10.5 and later. The replacement API is the Pasteboard Manager. For more information, see *Pasteboard Manager Programming Guide*.

In Mac OS 9 and earlier, the Scrap Manager allowed applications to copy and paste data using the Clipboard. The Scrap Manager was included in Carbon to facilitate the porting of legacy applications to Mac OS X. Only the `LoadScrap` and `UnloadScrap` functions were retained from the original Scrap Manager. However, the Carbon Scrap Manager provided new features, including support for promises.

The Pasteboard Manager supersedes the Scrap Manager and the drag flavor functionality in the Drag Manager, adding greater flexibility in the type and quantity of data to be transferred. Pasteboard Manager pasteboards are also fully compatible with Cocoa pasteboards.

You should not use the Scrap Manager in new application development.

Functions by Task

Getting Information About the Scrap

`GetScrapFlavorCount` (page 1500) Deprecated in Mac OS X v10.5
Obtains a scrap flavor count to use with the `GetScrapFlavorInfoList` function. (Deprecated. Use `PasteboardCopyItemFlavors` instead.)

`GetScrapFlavorFlags` (page 1502) Deprecated in Mac OS X v10.5
Obtains information about a specified scrap flavor. (Deprecated. Use `PasteboardGetItemFlavorFlags` instead.)

`GetScrapFlavorInfoList` (page 1502) Deprecated in Mac OS X v10.5
Fills an array with items which each describe a corresponding flavor in the scrap. (Deprecated. Use `PasteboardCopyItemFlavors` instead.)

Reading Information From the Scrap

`GetCurrentScrap` (page 1499) Deprecated in Mac OS X v10.5
Obtains a reference to the current scrap. (Deprecated. Use `PasteboardCreate` instead.)

`GetScrapByName` (page 1499) Deprecated in Mac OS X v10.5
Obtains a reference to a named scrap. (Deprecated. Use `PasteboardCreate` instead.)

`GetScrapFlavorData` (page 1501) Deprecated in Mac OS X v10.5
Obtains the data for the specified flavor from the specified scrap. (Deprecated. Use `PasteboardCopyItemFlavorData` instead.)

`GetScrapFlavorSize` (page 1503) Deprecated in Mac OS X v10.5
Obtains the size of the data for a specified flavor from a scrap. (Deprecated. Use `PasteboardCopyItemFlavorData` instead.)

Writing Information to the Scrap

`ClearCurrentScrap` (page 1497) Deprecated in Mac OS X v10.5
Clears the current scrap. (Deprecated. Use `PasteboardClear` instead.)

`ClearScrap` (page 1498) Deprecated in Mac OS X v10.5
Clears a given scrap. (Deprecated. Use `PasteboardClear` instead.)

`PutScrapFlavor` (page 1505) Deprecated in Mac OS X v10.5
Puts data on or promises data to the specified scrap. (Deprecated. Use `PasteboardPutItemFlavor` instead.)

Transferring Data Between the Scrap in Memory and the Scrap on Disk

`LoadScrap` (page 1504) Deprecated in Mac OS X v10.5
Reads the scrap from the scrap file into memory.

`UnloadScrap` (page 1507) Deprecated in Mac OS X v10.5
Writes the scrap from memory to the scrap file.

Working With Scrap Promise Keeper Functions

`CallInScrapPromises` (page 1497) Deprecated in Mac OS X v10.5
Forces all promised flavors to be supplied. (Deprecated. Use `PasteboardResolvePromises` instead.)

`DisposeScrapPromiseKeeperUPP` (page 1498) Deprecated in Mac OS X v10.5
Disposes of a universal procedure pointer to a function that provides promised scrap data.

`InvokeScrapPromiseKeeperUPP` (page 1504) Deprecated in Mac OS X v10.5
Calls a universal procedure pointer to a function that provides promised scrap data.

`NewScrapPromiseKeeperUPP` (page 1505) Deprecated in Mac OS X v10.5
Creates a new universal procedure pointer to a function that provides promised scrap data.

`SetScrapPromiseKeeper` (page 1507) Deprecated in Mac OS X v10.5
Associates an application-defined promise-keeper function with a scrap or removes an associated promise-keeper. (Deprecated. Use `PasteboardSetPromiseKeeper` instead.)

Functions

CallInScrapPromises

Forces all promised flavors to be supplied. (Deprecated in Mac OS X v10.5. Use `PasteboardResolvePromises` instead.)

```
OSStatus CallInScrapPromises (
   void
);
```

Return Value
A result code. See "Scrap Manager Result Codes" (page 1514).

Discussion
Before quitting, your application should call `CallInScrapPromises` in order to ensure the user's ability to paste into other applications. Your application should call `CallInScrapPromises` even if your application does not explicitly promise any flavors.

It doesn't hurt to call `CallInScrapPromises` more than once, though promise-keeper functions may be asked to keep promises they already tried and failed.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Scrap.h`

ClearCurrentScrap

Clears the current scrap. (Deprecated in Mac OS X v10.5. Use `PasteboardClear` instead.)

```
OSStatus ClearCurrentScrap (
   void
);
```

Return Value
A result code. See "Scrap Manager Result Codes" (page 1514).

Discussion
Call `ClearCurrentScrap` immediately when the user requests a Copy or Cut operation, even if you maintain a private scrap. You should not wait until receiving a suspend event to call `ClearCurrentScrap`. You don't need to put any data on the scrap immediately, although it's perfectly fine to do so. You do need to call `GetCurrentScrap` (page 1499) after `ClearCurrentScrap` so you'll have a valid scrap reference to pass to other functions.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Scrap.h`

ClearScrap

Clears a given scrap. (Deprecated in Mac OS X v10.5. Use `PasteboardClear` instead.)

```
OSStatus ClearScrap (
   ScrapRef *inOutScrap
);
```

Parameters

inOutScrap

A pointer to a scrap reference. On input, this parameter should refer to the scrap to clear. On output, `ClearScrap` returns a reference to the cleared scrap.

Return Value
A result code. See "Scrap Manager Result Codes" (page 1514).

Discussion
`ClearScrap` will clear the scrap passed in and return the incremented `ScrapRef` value. `ClearScrap` behaves similarly to the `GetScrapByName` function when called with the `kScrapClearNamedScrap` option, with the benefit of not requiring a name in the event one is not available.

Availability
Available in Mac OS X v10.1 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Scrap.h`

DisposeScrapPromiseKeeperUPP

Disposes of a universal procedure pointer to a function that provides promised scrap data. (Deprecated in Mac OS X v10.5.)

```
void DisposeScrapPromiseKeeperUPP (
   ScrapPromiseKeeperUPP userUPP
);
```

Parameters

userUPP

The UPP to dispose of.

Discussion
See `ScrapPromiseKeeperProcPtr` (page 1508) for more information on promise keeper functions.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`Scrap.h`

GetCurrentScrap

Obtains a reference to the current scrap. (Deprecated in Mac OS X v10.5. Use `PasteboardCreate` instead.)

```
OSStatus GetCurrentScrap (
   ScrapRef *scrap
);
```

Parameters

scrap

A pointer to a scrap reference. On return, this scrap reference refers to the current scrap.

Return Value

A result code. See "Scrap Manager Result Codes" (page 1514).

Discussion

Your application can determine if the scrap contents have changed by storing the scrap reference returned by `GetCurrentScrap` and comparing it against the scrap reference returned by `GetCurrentScrap` at a later time. If the two scrap references are different, the scrap has changed.

Carbon applications should use `GetCurrentScrap` instead of checking the `convertClipboardFlag` in the `EventRecord`.

The `ScrapRef` obtained via `GetCurrentScrap` becomes invalid and unusable after the scrap is cleared. That is, the scrap reference is valid until a Carbon client calls `ClearCurrentScrap` (page 1497), a Classic client calls `ZeroScrap`, or a Cocoa client calls `declareTypes:owner:`.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Scrap.h`

GetScrapByName

Obtains a reference to a named scrap. (Deprecated in Mac OS X v10.5. Use `PasteboardCreate` instead.)

```
OSStatus GetScrapByName (
   CFStringRef name,
   OptionBits options,
   ScrapRef *scrap
);
```

Parameters

name

A CFString containing the name of the scrap to obtain. You may specify a standard scrap by passing one of the constants described in "Named Scraps" (page 1513) in this parameter.

options

A value indicating whether the specified scrap should be cleared after the reference is returned. See "Options for the GetScrapByName Function" (page 1513) for more information.

scrap
A pointer to a scrap reference. On return, this scrap reference refers to the scrap named in the `name` parameter.

Return Value
A result code. See "Scrap Manager Result Codes" (page 1514).

Discussion
`GetScrapByName` allows access to an indefinite number of public or private scraps. The constant `kScrapClipboardScrap` refers to the "current" scrap. `kScrapFindScrap` allows Carbon applications to interact seamlessly with Cocoa's global find scrap. Note that calling:

```
GetScrapByName( kScrapClipboardScrap, kScrapGetNamedScrap, &scrap  );
```

is an exact match to the call:

```
GetCurrentScrap( &scrap );
```

Additionally, a call to:

```
 GetScrapByName( kScrapClipboardScrap, kScrapClearNamedScrap, &scrap  );
```

is a replacement for the sequence:

```
ClearCurrentScrap();
GetCurrentScrap( &scrap );
```

You can use this API to generate your own private scraps to use as a high level interprocess communication between your main and helper applications. Apple recommends using the Java naming convention for your scraps, for example, com.mycompany.scrap.secret.

Availability
Available in Mac OS X v10.1 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Scrap.h`

GetScrapFlavorCount

Obtains a scrap flavor count to use with the `GetScrapFlavorInfoList` function. (Deprecated in Mac OS X v10.5. Use `PasteboardCopyItemFlavors` instead.)

```
OSStatus GetScrapFlavorCount (
   ScrapRef scrap,
   UInt32 *infoCount
);
```

Parameters
scrap
A reference to the scrap to get the flavor count for.
infoCount
A pointer to a count variable. On return, specifies the flavor count for the specified scrap.

Return Value
A result code. See "Scrap Manager Result Codes" (page 1514).

Discussion
For related information, see `GetScrapFlavorInfoList` (page 1502).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Scrap.h`

GetScrapFlavorData

Obtains the data for the specified flavor from the specified scrap. (Deprecated in Mac OS X v10.5. Use `PasteboardCopyItemFlavorData` instead.)

```
OSStatus GetScrapFlavorData (
   ScrapRef scrap,
   ScrapFlavorType flavorType,
   Size *byteCount,
   void *destination
);
```

Parameters

scrap
A reference to the scrap to get data from.

flavorType
The flavor type to obtain data for. Some flavor types are described in "Scrap Flavor Types" (page 1511).

byteCount
A pointer to a size variable. Before calling `GetScrapFlavorData`, specify the maximum number of bytes your buffer can contain. On return, `GetScrapFlavorData` provides the number of bytes of data that were actually available, even if this is more than you requested.

destination
A pointer to a buffer, local variable, or other storage location created and disposed of by your application. The size, in bytes, must be at least as large as the value you pass in the `byteCount` parameter. On return, this buffer contains the specified flavor data. The amount of data returned will not exceed the value you passed in `byteCount`, even if the number of bytes of available data is more than you specified in `byteCount`.

Return Value
A result code. See "Scrap Manager Result Codes" (page 1514).

Discussion
This function blocks until the specified flavor data is available.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Scrap.h`

GetScrapFlavorFlags

Obtains information about a specified scrap flavor. (Deprecated in Mac OS X v10.5. Use `PasteboardGetItemFlavorFlags` instead.)

```
OSStatus GetScrapFlavorFlags (
   ScrapRef scrap,
   ScrapFlavorType flavorType,
   ScrapFlavorFlags *flavorFlags
);
```

Parameters

scrap
A reference to the scrap to check.

flavorType
The flavor type to check for. Some scrap flavor types are described in "Scrap Flavor Types" (page 1511).

flavorFlags
A pointer to a variable of type `ScrapFlavorFlags`; values for this type are described in "Scrap Flavor Flags" (page 1512). On return, this variable contains information about the flavor specified by the `flavorType` parameter.

Return Value
A result code. See "Scrap Manager Result Codes" (page 1514).

Discussion
The `GetScrapFlavorFlags` function tells you whether the scrap contains data for a particular flavor and, if it does, provides some information about that flavor. This function never blocks, and is useful for deciding whether to enable the Paste item in your Edit menu, among other things.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Scrap.h`

GetScrapFlavorInfoList

Fills an array with items which each describe a corresponding flavor in the scrap. (Deprecated in Mac OS X v10.5. Use `PasteboardCopyItemFlavors` instead.)

```
OSStatus GetScrapFlavorInfoList (
   ScrapRef scrap,
   UInt32 *infoCount,
   ScrapFlavorInfo info[]
);
```

Parameters

scrap

A reference to the scrap to get the flavor information for.

infoCount

A pointer to a count variable. Before calling `GetScrapFlavorInfoList`, set the value to the number of flavors to get information for. Your application typically obtains the flavor count by calling `GetScrapFlavorCount` (page 1500).

On return, specifies the number of flavors for which information was supplied, which may be smaller than the number requested.

info

An array of type `ScrapFlavorInfo` (page 1509), whose size is indicated by the `infoCount` parameter. The array is created and disposed of by your application. On return, the array elements contain the flavor information.

Return Value

A result code. See "Scrap Manager Result Codes" (page 1514).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Scrap.h`

GetScrapFlavorSize

Obtains the size of the data for a specified flavor from a scrap. (Deprecated in Mac OS X v10.5. Use `PasteboardCopyItemFlavorData` instead.)

```
OSStatus GetScrapFlavorSize (
   ScrapRef scrap,
   ScrapFlavorType flavorType,
   Size *byteCount
);
```

Parameters

scrap

A reference to the scrap to get the flavor data size from.

flavorType

The flavor type to obtain the size for. Some flavor types are described in "Scrap Flavor Types" (page 1511).

byteCount

A pointer to a size variable. On return, this variable contains the byte count for the data of the specified flavor.

Return Value
A result code. See "Scrap Manager Result Codes" (page 1514).

Discussion
This function will block until the size of the data is available. This may mean blocking until the data itself is available, since some scrap senders don't know how big a flavor will be until they've made the flavor data. `GetScrapFlavorSize` is intended as a prelude to allocating memory and calling `GetScrapFlavorData` (page 1501).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Scrap.h`

InvokeScrapPromiseKeeperUPP

Calls a universal procedure pointer to a function that provides promised scrap data. (Deprecated in Mac OS X v10.5.)

```
OSStatus InvokeScrapPromiseKeeperUPP (
   ScrapRef scrap,
   ScrapFlavorType flavorType,
   void *userData,
   ScrapPromiseKeeperUPP userUPP
);
```

Return Value
A result code. See "Scrap Manager Result Codes" (page 1514).

Discussion
You should not need to use the function `InvokeScrapPromiseKeeperUPP`, as the system calls your scrap promise keeper function for you.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`Scrap.h`

LoadScrap

Reads the scrap from the scrap file into memory. (Deprecated in Mac OS X v10.5.)

```
OSStatus LoadScrap (
   void
);
```

Return Value
A result code. See "Scrap Manager Result Codes" (page 1514).

Discussion
The function allocates memory in your application's heap to hold the scrap before reading the scrap into memory. The scrap file is located in the System Folder of the startup volume and has the filename as indicated by the `scrapName` field of the scrap information structure (usually "Clipboard"). If the scrap is already in memory, this function does nothing.

Special Considerations

In Mac OS X, this function does nothing and is no longer necessary.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`Scrap.h`

NewScrapPromiseKeeperUPP

Creates a new universal procedure pointer to a function that provides promised scrap data. (Deprecated in Mac OS X v10.5.)

```
ScrapPromiseKeeperUPP NewScrapPromiseKeeperUPP (
   ScrapPromiseKeeperProcPtr userRoutine
);
```

Parameters
`userRoutine`
A pointer to your scrap promise keeper function.

Return Value
A UPP to the scrap promise keeper function. See the description of the `ScrapPromiseKeeperUPP` data type.

Discussion
See `ScrapPromiseKeeperProcPtr` (page 1508) for more information on promise keeper functions.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`Scrap.h`

PutScrapFlavor

Puts data on or promises data to the specified scrap. (Deprecated in Mac OS X v10.5. Use `PasteboardPutItemFlavor` instead.)

```
OSStatus PutScrapFlavor (
   ScrapRef scrap,
   ScrapFlavorType flavorType,
   ScrapFlavorFlags flavorFlags,
   Size flavorSize,
   const void *flavorData
);
```

Parameters

scrap

A reference to the scrap to supply data or promises to.

flavorType

The flavor type to supply or promise the data for. Some flavor types are described in "Scrap Flavor Types" (page 1511).

flavorFlags

A variable of type `ScrapFlavorFlags` that you use to supply information about the flavor specified by the `flavorType` parameter. See "Scrap Flavor Flags" (page 1512) for a description of the values you can use in this parameter.

flavorSize

The size of the data you are supplying or promising, in bytes. If you don't know the size, pass `kScrapFlavorSizeUnknown` to place a promise for data of undetermined size on the scrap. If you pass 0 in this parameter, a flavor with no expected data—not a promise—is placed on the scrap, and the value of the *flavorData* parameter is ignored.

flavorData

A pointer to a buffer, local variable, or other storage location, created and disposed of by your application. Before calling `PutScrapFlavor` to put flavor data on the scrap, store the data in this buffer. For information on the number of bytes of data, see the description of the `flavorSize` parameter.

Pass `NULL` for this parameter to indicate you will provide data through a subsequent call to `PutScrapFlavor`, either later in the same code flow or during execution of your `ScrapPromiseKeeperProcPtr` (page 1508) callback.

The last time you can provide scrap flavor data is when your scrap promise-keeper function gets called. It is not possible to call `PutScrapFlavor` while handling a suspend event; suspend events under Carbon don't work the way they do under Mac OS 8 and 9.

Return Value

A result code. See "Scrap Manager Result Codes" (page 1514).

Discussion

`PutScrapFlavor` is different than `PutScrap` in that it includes a `ScrapRef` parameter and it supports promising a flavor for later delivery, rather than supplying it immediately.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Scrap.h`

SetScrapPromiseKeeper

Associates an application-defined promise-keeper function with a scrap or removes an associated promise-keeper. (Deprecated in Mac OS X v10.5. Use `PasteboardSetPromiseKeeper` instead.)

```
OSStatus SetScrapPromiseKeeper (
   ScrapRef scrap,
   ScrapPromiseKeeperUPP upp,
   const void *userData
);
```

Parameters

scrap

A reference to the scrap to set the promise-keeper function for.

upp

A universal procedure pointer to a scrap promise-keeper function. For more information, see `ScrapPromiseKeeperProcPtr` (page 1508)

You can remove a promise-keeper function from a scrap by passing `NULL` for this parameter.

userData

An untyped pointer to a buffer, local variable, or other storage location, created and disposed of by your application. This value is passed to the promise-keeper function specified by the `upp` parameter, which can do whatever it needs to do with the value. For example, you might pass a pointer or handle to some private scrap data that your promise-keeper function uses in fabricating one or more promised flavors.

If your promise-keeper function has no need for special user data, pass `NULL` for this parameter.

Return Value

A result code. See "Scrap Manager Result Codes" (page 1514).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Scrap.h`

UnloadScrap

Writes the scrap from memory to the scrap file. (Deprecated in Mac OS X v10.5.)

```
OSStatus UnloadScrap (
   void
);
```

Return Value

A result code. See "Scrap Manager Result Codes" (page 1514).

Discussion

This function releases the memory occupied by the scrap in your application's heap. The scrap file is located in the System Folder of the startup volume and has the filename as indicated by the `scrapName` field of the scrap information structure (usually "Clipboard"). If the scrap is already on the disk, this function does nothing.

Special Considerations

In Mac OS X, this function does nothing and is no longer necessary.

Availability

Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In

`Scrap.h`

Callbacks

ScrapPromiseKeeperProcPtr

Defines a pointer to a function the Scrap Manager calls to obtain promised scrap data for a specified flavor.

```
typedef OSStatus (*ScrapPromiseKeeperProcPtr)
(
                ScrapRef scrap,
                ScrapFlavorType flavorType,
                void * userData);
```

If you name your function `MyScrapPromiseKeeperCallback`, you would declare it like this:

```
OSStatus MyScrapPromiseKeeperCallback (

            ScrapRef scrap,
            ScrapFlavorType flavorType,
            void * userData);
```

Parameters

scrap

A reference to the scrap to which to supply the promised flavor data.

flavorType

The flavor type to supply the promised data for. Some scrap flavor types are described in "Scrap Flavor Types" (page 1511).

userData

An untyped pointer to a buffer, local variable, or other storage location, created and disposed of by your application, and supplied to the Scrap Manager by a previous call to `SetScrapPromiseKeeper` (page 1507). For example, this parameter might refer to a pointer or handle to some private scrap data which your promise-keeper function uses in fabricating one or more promised flavors.

This parameter may have a value of `NULL`.

Return Value

A result code. See "Scrap Manager Result Codes" (page 1514).

Discussion
To provide a pointer to your promise-keeper function, you create a universal procedure pointer (UPP), using the function `NewScrapPromiseKeeperUPP` (page 1505). You can do so with code similar to the following:

```
ScrapPromiseKeeperUPP MyPromiseKeeperUPP;
MyPromiseKeeperUPP = NewScrapPromiseKeeperUPP (&MyScrapPromiseKeeperCallback);
```

You can then associate the UPP with a scrap by passing `MyScrapPromiseKeeperCallback` as a parameter to the `SetScrapPromiseKeeper` (page 1507) function.

If a promised flavor is requested through `GetScrapFlavorData` (page 1501) or `GetScrapFlavorSize` (page 1503), the Scrap Manager calls the application's `ScrapPromiseKeeperProcPtr` function. The scrap promise-keeper function should call `PutScrapFlavor` (page 1505) as appropriate to fulfill its promises. Failure to do so—including returning an error or simply neglecting to keep a promise for a flavor—will result in errors being returned to corresponding callers of `GetScrapFlavorData` or `GetScrapFlavorSize`.

Under Mac OS X, the scrap promise-keeper function can be called during any call to `GetScrapFlavorData` or `GetScrapFlavorSize`. Under Mac OS 8 and 9, the Carbon Scrap Manager still must support non-Carbon callers of `GetScrap`, which does not know about promised flavors. As a result, the Carbon Scrap Manager must make sure all promises have been kept when the application is suspended.

It is okay to keep a promise without ever receiving a call to a scrap promise-keeper function. You should not call `WaitNextEvent` or any similar function from your promise-keeper function because the promise-keeper might be running at suspend event time.

After you are finished with a promise-keeper function, and have removed it by passing `NULL` to the `SetScrapPromiseKeeper` (page 1507) function, you can dispose of the UPP with the `DisposeScrapPromiseKeeperUPP` (page 1498) function. However, don't dispose of the UPP if any other scrap uses it or if you plan to use it again.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Scrap.h`

Data Types

ScrapFlavorInfo

Describes a single flavor within a scrap.

```
struct ScrapFlavorInfo {
    ScrapFlavorType flavorType;
    ScrapFlavorFlags flavorFlags;
};
typedef struct ScrapFlavorInfo ScrapFlavorInfo;
```

Fields
`flavorType`
The flavor type for the flavor. Some values for flavor types are described in "Scrap Flavor Types" (page 1511).

`flavorFlags`
The flavor flags for the flavor. Some values for flavor flags are described in "Scrap Flavor Flags" (page 1512).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Scrap.h`

ScrapRef

Defines a reference to a scrap.

```
typedef struct OpaqueScrapRef * ScrapRef;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Scrap.h`

ScrapPromiseKeeperUPP

Defines a universal procedure pointer to a scrap promise–keeper function.

```
typedef ScrapPromiseKeeperProcPtr ScrapPromiseKeeperUPP;
```

Discussion
See `ScrapPromiseKeeperProcPtr` (page 1508) for more information on scrap promise–keeper functions.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Scrap.h`

ScrapFlavorType

Defines a scrap flavor type.

```
typedef FourCharCode ScrapFlavorType;
```

Discussion
Some values for scrap flavor types are shown in "Scrap Flavor Types" (page 1511).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Scrap.h`

Constants

Scrap Flavor Types

Identify commonly used scrap flavor types.

```
enum {
    kScrapFlavorTypePicture = 'PICT',
    kScrapFlavorTypeText = 'TEXT',
    kScrapFlavorTypeTextStyle = 'styl',
    kScrapFlavorTypeMovie = 'moov',
    kScrapFlavorTypeSound = 'snd ',
    kScrapFlavorTypeUnicode = 'utxt',
    kScrapFlavorTypeUnicodeStyle = 'ustl'
};
```

Constants

`kScrapFlavorTypePicture`
Specifies the contents of a `PicHandle`.

Available in Mac OS X v10.0 and later.

Declared in `Scrap.h`.

`kScrapFlavorTypeText`
Specifies a stream of text characters.

Available in Mac OS X v10.0 and later.

Declared in `Scrap.h`.

`kScrapFlavorTypeTextStyle`
Specifies a style scrap structure. For more information, see `TEGetStyleScrapHandle` in *Inside Mac OS X: TextEdit Reference*.

Available in Mac OS X v10.0 and later.

Declared in `Scrap.h`.

`kScrapFlavorTypeMovie`
Specifies a reference to a QuickTime movie.

Available in Mac OS X v10.0 and later.

Declared in `Scrap.h`.

`kScrapFlavorTypeSound`
Specifies the contents of a `'snd '` resource. For more information, see the functions `SndRecord` and `SndPlay` in *Inside Mac OS X: Sound Manager Reference*.

Available in Mac OS X v10.0 and later.

Declared in `Scrap.h`.

`kScrapFlavorTypeUnicode`
Specifies a stream of 16-bit Unicode characters.

Available in Mac OS X v10.0 and later.

Declared in `Scrap.h`.

`kScrapFlavorTypeUnicodeStyle`
Specifies Unicode style information; defined by ATSUI and used by Textension.

Available in Mac OS X v10.0 and later.

Declared in `Scrap.h`.

Scrap Flavor Flags

Describe attributes of a scrap flavor.

```
enum {
    kScrapFlavorMaskNone = 0x00000000,
    kScrapFlavorMaskSenderOnly = 0x00000001,
    kScrapFlavorMaskTranslated = 0x00000002
};
typedef UInt32 ScrapFlavorFlags;
```

Constants

`kScrapFlavorMaskNone`
No flavors in the scrap.

Available in Mac OS X v10.0 and later.

Declared in `Scrap.h`.

`kScrapFlavorMaskSenderOnly`
The flavor is intended to be visible to the sender only. This is typically used to save a private flavor into the scrap so that other public promised flavors can be derived from it on demand. If another process put a flavor with this flag on the scrap, your process will never see the flavor, so you do not need to test for this flag.

Available in Mac OS X v10.0 and later.

Declared in `Scrap.h`.

`kScrapFlavorMaskTranslated`
The flavor has been translated (or will be translated when the promise is kept) from some other flavor in the scrap, either by the sender or by the Translation Manager.

Available in Mac OS X v10.0 and later.

Declared in `Scrap.h`.

Discussion

To determine the attributes of a scrap flavor, call the `GetScrapFlavorFlags` (page 1502) function.

Reserved Flavor Type

Identifies a flavor type reserved by the Scrap Manager.

```
enum {
    kScrapReservedFlavorType = 'srft'
};
```

Discussion

`kScrapReservedFlavorType` is a flavor type which is reserved for use by the Scrap Manager. If you pass it to the Scrap Manager, it will be rejected.

Unknown Flavor Data Size Constant

Indicates that the size of the scrap flavor data is unknown.

```
enum {
    kScrapFlavorSizeUnknown = -1
};
```

Discussion
When promising a scrap flavor, it is okay if you don't yet know how big the flavor data will be. In this case, pass `kScrapFlavorSizeUnknown` for the flavor data size.

Options for the GetScrapByName Function

Define options passed to the `GetScrapByName` function.

```
enum {
    kScrapGetNamedScrap = 0,
    kScrapClearNamedScrap = 0x00000001
};
```

Constants

`kScrapGetNamedScrap`
Get current named scrap without bumping or clearing the scrap.
Available in Mac OS X v10.1 and later.
Declared in `Scrap.h`.

`kScrapClearNamedScrap`
Acquire the named scrap, bumping and clearing the scrap.
Available in Mac OS X v10.1 and later.
Declared in `Scrap.h`.

Invalid Scrap Reference

Defines an invalid scrap reference.

```
#define kScrapRefNone ((ScrapRef)NULL)
```

Discussion
`kScrapRefNone` is guaranteed to be an invalid `ScrapRef`. This is convenient when initializing application variables.

Named Scraps

Specify Apple–defined scrap names for the `GetScrapByName` function.

```
#define kScrapClipboardScrap CFSTR("com.apple.scrap.clipboard")
#define kScrapFindScrap CFSTR("com.apple.scrap.find")
```

Constants

`kScrapClipboardScrap`

Traditional clipboard scrap.

Available in Mac OS X v10.1 and later.

Declared in `Scrap.h`.

`kScrapFindScrap`

Compatible with Cocoa's global find scrap.

Available in Mac OS X v10.1 and later.

Declared in `Scrap.h`.

Result Codes

The most common result codes returned by Scrap Manager are listed below.

Result Code	Value	Description
`needClearScrapErr`	-100	Scrap does not exist (same as `noScrapErr`) Available in Mac OS X v10.0 and later.
`noScrapErr`	-100	Scrap does not exist (not initialized) Available in Mac OS X v10.0 and later.
`noTypeErr`	-102	No data of the requested format type in scrap Available in Mac OS X v10.0 and later.
`scrapFlavorNotFoundErr`	-102	No data of the requested format type in scrap (same as `noTypeErr`) Available in Mac OS X v10.0 and later.
`internalScrapErr`	-4988	Scrap Manager has encountered an internal error Available in Mac OS X v10.0 and later.
`duplicateScrapFlavorErr`	-4989	Data (or a promise for data) of the specified format already exists in the scrap Available in Mac OS X v10.0 and later.
`badScrapRefErr`	-4990	Invalid scrap reference Available in Mac OS X v10.0 and later.
`processStateIncorrectErr`	-4991	Available in Mac OS X v10.0 and later.
`scrapPromiseNotKeptErr`	-4992	No data supplied for the promised flavor Available in Mac OS X v10.0 and later.

Result Code	Value	Description
noScrapPromiseKeeperErr	-4993	No scrap promise–keeper function set for the scrap Available in Mac OS X v10.0 and later.
nilScrapFlavorDataErr	-4994	Available in Mac OS X v10.0 and later.
scrapFlavorFlagsMismatchErr	-4995	Available in Mac OS X v10.0 and later.
scrapFlavorSizeMismatchErr	-4996	Available in Mac OS X v10.0 and later.
illegalScrapFlavorFlagsErr	-4997	Invalid scrap flavor flags Available in Mac OS X v10.0 and later.
illegalScrapFlavorTypeErr	-4998	Invalid scrap flavor Available in Mac OS X v10.0 and later.
illegalScrapFlavorSizeErr	-4999	Invalid size for scrap flavor data Available in Mac OS X v10.0 and later.

Gestalt Constants

You can check for version and feature availability information by using the Scrap Manager selectors defined in the Gestalt Manager. For more information see *Gestalt Manager Reference*.

Speech Recognition Manager Reference

Framework: Carbon/Carbon.h

Declared in SpeechRecognition.h

Overview

The Speech Recognition Manager provides speech recognition support in applications.

Functions by Task

Opening and Closing Recognition Systems

`SRCloseRecognitionSystem` (page 1524)
Closes a recognition system when your application is finished using it (for example, just before your application quits).

`SROpenRecognitionSystem` (page 1537)
Opens a recognition system.

Creating and Manipulating Recognizers

`SRCancelRecognition` (page 1522)
Cancels the attempt to recognize the current utterance.

`SRContinueRecognition` (page 1524)
Causes a recognizer to continue recognizing speech.

`SRGetLanguageModel` (page 1528)
Gets a recognizer's active language model.

`SRIdle` (page 1531)
Grants processing time to the Speech Recognition Manager if your application does not call `WaitNextEvent` frequently.

`SRNewRecognizer` (page 1535)
Creates a new recognizer.

`SRSetLanguageModel` (page 1543)
Sets a recognizer's active language model.

`SRStartListening` (page 1546)
Starts a recognizer listening and reporting results to your application.

`SRStopListening` (page 1547)
Stops a recognizer listening and reporting results to your application.

Managing Speech Objects

`SRGetProperty` (page 1529)
Gets the current value of a property of a speech object.

`SRGetReference` (page 1530)
Obtains an extra reference to a speech object.

`SRReleaseObject` (page 1540)
Releases a speech object.

`SRSetProperty` (page 1543)
Sets the value of a property of a speech object.

Traversing Speech Objects

`SRCountItems` (page 1525)
Determines the number of subitems in a container object.

`SRGetIndexedItem` (page 1527)
Gets a subitem in a container object.

`SRRemoveIndexedItem` (page 1541)
Removes a subitem from a container object.

`SRSetIndexedItem` (page 1542)
Replaces a subitem in a container object with some other object.

Creating Language Objects

`SRNewLanguageModel` (page 1531)
Creates a new language model.

`SRNewPath` (page 1534)
Creates a new path.

`SRNewPhrase` (page 1535)
Creates a new phrase.

`SRNewWord` (page 1536)
Creates a new word.

Manipulating Language Objects

`SRAddLanguageObject` (page 1521)
Adds a language object to some other language object.

`SRAddText` (page 1522)
Adds text to the contents of a language object.

SRChangeLanguageObject (page 1523)
Changes the contents of a language object.

SREmptyLanguageObject (page 1527)
Empties the contents of a language object.

SRNewLanguageObjectFromHandle (page 1533)
Creates a language object from the handle previously created by the SRPutLanguageObjectIntoHandle function.

SRNewLanguageObjectFromDataFile (page 1532)
Reads a language object from a data file.

SRPutLanguageObjectIntoHandle (page 1540)
Puts a language object (and any embedded languages objects it contains) into a handle.

SRPutLanguageObjectIntoDataFile (page 1539)
Puts a language object (and any embedded language objects it contains) into a data file.

SRRemoveLanguageObject (page 1541)
Removes a language object from another language object that contains it.

Using the System Feedback Window

SRDrawRecognizedText (page 1525)
Draws recognized text in the feedback window.

SRDrawText (page 1526)
Draws output text in the feedback window.

SRProcessBegin (page 1537)
Indicates that a recognition result is being processed.

SRProcessEnd (page 1538)
Indicates that a recognition result is done being processed.

SRSpeakAndDrawText (page 1544)
Draws output text in the feedback window and causes the feedback character in the feedback window to speak that text.

SRSpeakText (page 1545)
Causes the feedback character in the feedback window to speak a text string.

SRSpeechBusy (page 1546)
Determines if the feedback character in a feedback window is currently speaking.

SRStopSpeech (page 1547)
Terminates speech by the feedback character in a feedback window.

Creating, Invoking and Disposing UPPs

NewSRCallBackUPP (page 1520)
Creates a new universal procedure pointer (UPP) to a speech recognition callback function.

DisposeSRCallBackUPP (page 1520)
Disposes of a universal procedure pointer (UPP) to a speech recognition callback function.

InvokeSRCallBackUPP (page 1520)
Invokes your speech recognition callback function.

Functions

DisposeSRCallBackUPP

Disposes of a universal procedure pointer (UPP) to a speech recognition callback function.

```
void DisposeSRCallBackUPP (
   SRCallBackUPP userUPP
);
```

Parameters

userUPP

The UPP to dispose of.

Availability
Available in CarbonLib 1.0.2 and later.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechRecognition.h`

InvokeSRCallBackUPP

Invokes your speech recognition callback function.

```
void InvokeSRCallBackUPP (
   SRCallBackStruct *param,
   SRCallBackUPP userUPP
);
```

Discussion
You should not have to call the `InvokeSRCallbackUPP` function, as the system calls your speech recognition callback function for you.

Availability
Available in CarbonLib 1.0.2 and later.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechRecognition.h`

NewSRCallBackUPP

Creates a new universal procedure pointer (UPP) to a speech recognition callback function.

```
SRCallBackUPP NewSRCallBackUPP (
   SRCallBackProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your speech recognition callback function.

Return Value
A UPP to the speech recognition callback function. See the description of the `SRCallBackUPP` data type.

Availability
Available in CarbonLib 1.0.2 and later.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechRecognition.h`

SRAddLanguageObject

Adds a language object to some other language object.

```
OSErr SRAddLanguageObject (
   SRLanguageObject base,
   SRLanguageObject addon
);
```

Parameters

`base`

The language object to which to add the language object specified by the `addon` parameter.

`addon`

The language object to add on to the language object specified in the `base` parameter. For example, if `addon` specifies a word and `base` specifies a phrase, then `SRAddLanguageObject` appends that word to the end of that phrase.

Return Value
A result code. See "Speech Recognition Manager Result Codes" (page 1568).

Discussion
The `SRAddLanguageObject` function is useful for adding language objects to phrases, paths, and language models. For a phrase or a path, `SRAddLanguageObject` appends the specified object to the end of the phrase or path. For a language model, `SRAddLanguageObject` adds the specified object to the list of alternative recognizable utterances.

The language object to which you add an object acquires a new reference to that object. Accordingly, any changes you subsequently make to the added object are reflected in any object to which you added it. The base object releases its reference to the added object when the base object is disposed of.

`SRAddLanguageObject` does not alter the value of the reference constant property of the language object specified by the `base` parameter.

See `SRAddText` (page 1522) for a useful shortcut function.

Availability
Available in CarbonLib 1.0 and later when Speech Recognition 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechRecognition.h`

SRAddText

Adds text to the contents of a language object.

```
OSErr SRAddText (
   SRLanguageObject base,
   const void *text,
   SInt32 textLength,
   SRefCon refCon
);
```

Parameters

base

A language object to which to add the text.

text

A pointer to a buffer that contains the words or phrase to add to the contents of the specified language object.

textLength

The size, in bytes, of the specified text.

refCon

An application-defined reference constant. The value of the reference constant property of the new word or phrase representing the specified text is set to this value.

Return Value

A result code. See "Speech Recognition Manager Result Codes" (page 1568).

Discussion

The `SRAddText` function is useful for phrases, paths, and language models. If the `base` parameter specifies a path or language model, `SRAddText` is equivalent to calling `SRNewPhrase`, `SRAddLanguageObject`, and `SRReleaseObject` for the phrase specified by the `text` parameter and calling `SRSetProperty` to reset the value of the reference constant property of the new phrase.

If the `base` parameter specifies a phrase, `SRAddText` is equivalent to calling `SRNewPhrase`, `SRAddLanguageObject`, and `SRReleaseObject` for each distinguishable word in the `text` parameter and calling `SRSetProperty` to set the value of the reference constant property of the new words.

Availability

Available in CarbonLib 1.0 and later when Speech Recognition 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In

`SpeechRecognition.h`

SRCancelRecognition

Cancels the attempt to recognize the current utterance.

```
OSErr SRCancelRecognition (
   SRRecognizer recognizer
);
```

Parameters

recognizer

A recognizer.

Return Value
A result code. See "Speech Recognition Manager Result Codes" (page 1568).

Discussion
The `SRCancelRecognition` function instructs the recognizer specified by the `recognizer` parameter to stop recognizing speech. You need to call either `SRContinueRecognition` or `SRCancelRecognition` each time your application is notified that the user has started speaking (using Apple events or through an application-defined callback routine).

Availability
Available in CarbonLib 1.0 and later when Speech Recognition 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechRecognition.h`

SRChangeLanguageObject

Changes the contents of a language object.

```
OSErr SRChangeLanguageObject (
   SRLanguageObject languageObject,
   const void *text,
   SInt32 textLength
);
```

Parameters

languageObject
A language object.

text
A pointer to a buffer that contains the words or phrase to which the contents of the specified language object are to be changed.

textLength
The size, in bytes, of the specified text.

Return Value
A result code. See "Speech Recognition Manager Result Codes" (page 1568).

Discussion
`SRChangeLanguageObject` is a convenient shortcut for calling `SREmptyLanguageObject` and then `SRAddText`.

`SRChangeLanguageObject` does not alter the value of the reference constant property of the language object specified by the `languageObject` parameter.

If there are no other references to the language object specified by the `languageObject` parameter, calling `SRChangeLanguageObject` causes that object to be released.

If you want to swap rapidly among several language models, you should use the `SRSetLanguageObject` function instead of `SRChangeLanguageObject`. Or, you could use the `kSREnabled` property to rapidly enable and disable parts of the current language model to reflect the current context.

Availability
Available in CarbonLib 1.0 and later when Speech Recognition 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In
`SpeechRecognition.h`

SRCloseRecognitionSystem

Closes a recognition system when your application is finished using it (for example, just before your application quits).

```
OSErr SRCloseRecognitionSystem (
   SRRecognitionSystem system
);
```

Parameters

system
A recognition system.

Return Value
A result code. See "Speech Recognition Manager Result Codes" (page 1568).

Discussion
The `SRCloseRecognitionSystem` function closes the recognition system specified by the `system` parameter. If any speech objects are still attached to that recognition system, they are disposed of and any references you have to those objects are thereby rendered invalid.

Availability
Available in CarbonLib 1.0 and later when Speech Recognition 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechRecognition.h`

SRContinueRecognition

Causes a recognizer to continue recognizing speech.

```
OSErr SRContinueRecognition (
   SRRecognizer recognizer
);
```

Parameters

recognizer
A recognizer.

Return Value
A result code. See "Speech Recognition Manager Result Codes" (page 1568).

Discussion
You need to call either `SRContinueRecognition` or `SRCancelRecognition` each time your application is notified that the user has started speaking (using Apple events or through an application-defined callback routine).

Availability
Available in CarbonLib 1.0 and later when Speech Recognition 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechRecognition.h`

SRCountItems

Determines the number of subitems in a container object.

```
OSErr SRCountItems (
   SRSpeechObject container,
   long *count
);
```

Parameters

container
A speech object.

count
On return, a pointer to a long containing the number of subitems in the specified speech object.

Return Value
A result code. See "Speech Recognition Manager Result Codes" (page 1568).

Discussion
This function is useful only for speech objects that have distinguishable subitems, such as phrases (which contain words), paths (which contain words, phrases, and language models), and language models (which contain words, phrases, paths, and possibly other language models).

Version Notes
In Speech Recognition Manager version 1.5, this function is useful only for operating on language objects (of type `SRLanguageObject`), although it is defined for all speech objects.

Availability
Available in CarbonLib 1.0 and later when Speech Recognition 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechRecognition.h`

SRDrawRecognizedText

Draws recognized text in the feedback window.

```
OSErr SRDrawRecognizedText (
   SRRecognizer recognizer,
   const void *dispText,
   SInt32 dispLength
);
```

Parameters

recognizer

A recognizer.

dispText

A pointer to a buffer that contains the text to be drawn.

dispLength

The size, in bytes, of the specified text.

Return Value

A result code. See "Speech Recognition Manager Result Codes" (page 1568).

Discussion

The `SRDrawRecognizedText` function draws the text specified by the `dispText` and `dispLength` parameters in the transcript portion of the feedback window associated with the recognizer specified by the `recognizer` parameter. The text is drawn in the style characteristic of all recognized text. You might want to use this function to display a recognized phrase using a different spelling than the one used in the language model.

If the value of the `kSRWantsResultTextDrawn` property of the specified recognizer is `TRUE` (which is the default value), a transcript of the text of a recognition result is automatically sent directly to the feedback window. As a result, you should call `SRDrawRecognizedText` only when the value of the recognizer's `kSRWantsResultTextDrawn` property is `FALSE`.

Availability

Available in CarbonLib 1.0 and later when Speech Recognition 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`SpeechRecognition.h`

SRDrawText

Draws output text in the feedback window.

```
OSErr SRDrawText (
   SRRecognizer recognizer,
   const void *dispText,
   SInt32 dispLength
);
```

Parameters

recognizer

A recognizer.

dispText

A pointer to a buffer that contains the text to be drawn.

dispLength

The size, in bytes, of the specified text.

Return Value
A result code. See "Speech Recognition Manager Result Codes" (page 1568).

Discussion
The `SRDrawText` function draws the text specified by the `dispText` and `dispLength` parameters in the transcript portion of the feedback window associated with the recognizer specified by the `recognizer` parameter. The text is drawn in the style characteristic of all output text.

Availability
Available in CarbonLib 1.0 and later when Speech Recognition 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechRecognition.h`

SREmptyLanguageObject

Empties the contents of a language object.

```
OSErr SREmptyLanguageObject (
   SRLanguageObject languageObject
);
```

Parameters

`languageObject`
A language object.

Return Value
A result code. See "Speech Recognition Manager Result Codes" (page 1568).

Discussion
The `SREmptyLanguageObject` function empties the contents of the language object specified by the `languageObject` parameter. (For example, if `languageObject` specifies a phrase containing two words, calling `SREmptyLanguageObject` would result in a phrase that contains no words.) Any properties of that object that are not related to its contents are unchanged. In particular, `SREmptyLanguageObject` does not alter the value of the reference constant property of that language object.

If there are no other references to the words, phrases, and paths that were contained in the language object, calling `SREmptyLanguageObject` causes them to be disposed of.

Availability
Available in CarbonLib 1.0 and later when Speech Recognition 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechRecognition.h`

SRGetIndexedItem

Gets a subitem in a container object.

```
OSErr SRGetIndexedItem (
   SRSpeechObject container,
   SRSpeechObject *item,
   long index
);
```

Parameters

container

A speech object.

item

On return, a reference to the subitem in the specified speech object that has the specified index.

index

An integer ranging from 0 to one less than the number of subitems in the specified speech object. (You can call the `SRCountItems` function to determine the number of subitems contained in a speech object.) If the index you specify is not in this range, `SRGetIndexedItem` returns the result code `kSRParamOutOfRange`.

Return Value

A result code. See "Speech Recognition Manager Result Codes" (page 1568).

Discussion

This function is useful for iterating through all subitems in a container object.

`SRGetIndexedItem` increases the reference count of the specified speech object. You should call the `SRReleaseObject` function to release the object reference returned by `SRGetIndexedItem` when you are done using it. For example, you can get a reference to the third word in a phrase by executing this code:

```
myErr = SRGetIndexedItem(myPhrase, &myWord, 2)
```

Then, when you are finished using the word, you should execute this code:

```
myErr = SRReleaseObject(myWord);
```

Version Notes

In Speech Recognition Manager version 1.5, this function is useful only for operating on language objects (of type `SRLanguageObject`), although it is defined for all speech objects.

Availability

Available in CarbonLib 1.0 and later when Speech Recognition 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`SpeechRecognition.h`

SRGetLanguageModel

Gets a recognizer's active language model.

```
OSErr SRGetLanguageModel (
   SRRecognizer recognizer,
   SRLanguageModel *languageModel
);
```

Parameters

recognizer

A recognizer.

languageModel

On return, a reference to the language model currently active for the specified recognizer.

Return Value

A result code. See "Speech Recognition Manager Result Codes" (page 1568).

Discussion

`SRGetLanguageModel` increases the reference count of the specified language model. You should call the `SRReleaseObject` function to release the language model reference returned by `SRGetLanguageModel` when you are done using it.

Availability

Available in CarbonLib 1.0 and later when Speech Recognition 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`SpeechRecognition.h`

SRGetProperty

Gets the current value of a property of a speech object.

```
OSErr SRGetProperty (
   SRSpeechObject srObject,
   OSType selector,
   void *property,
   Size *propertyLen
);
```

Parameters

srObject

A speech object.

selector

A property selector. See "Recognizer Properties" (page 1564), "Recognizer Listen Key Properties" (page 1563), "Language Object Properties" (page 1557), and "Recognition System Properties" (page 1562) for lists of the available property selectors.

property

A pointer to a buffer into which the value of the specified property is to be copied.

propertyLen

On entry, a pointer to the length, in bytes, of the specified buffer. If the value is of a fixed size, then `propertyLen` should point to a variable of type `Size` that specifies that size. If the size of the value can vary (for example, if the value is a string), then `propertyLen` should point to a variable of type `Size` that specifies the number of bytes in the buffer pointed to by the `property` parameter.

On return, if the buffer is large enough to hold the returned property value and no error occurs, `SRGetProperty` sets `propertyLen` to the total number of bytes in the value of the specified property. If the buffer is not large enough to hold the returned property value, `SRGetProperty` sets `propertyLen` to the number of bytes required to store the requested property and returns the `kSRBufferTooSmall` error code.

Return Value
A result code. See "Speech Recognition Manager Result Codes" (page 1568).

Discussion
Not all selectors are valid for all types of speech objects. If the selector you specify does not specify a property of the specified speech object, `SRGetProperty` returns the result code `kSRCantGetProperty`.

If `SRGetProperty` returns an object reference, you must make sure to release that object reference (by calling `SRReleaseObject`) when you are finished using it. Most selectors do not cause `SRGetProperty` to return object references. For example, passing the selector `kSRSpelling` causes `SRGetProperty` to return a buffer of text, not an object reference.

Availability
Available in CarbonLib 1.0 and later when Speech Recognition 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechRecognition.h`

SRGetReference

Obtains an extra reference to a speech object.

```
OSErr SRGetReference (
   SRSpeechObject srObject,
   SRSpeechObject *newObjectRef
);
```

Parameters

srObject

A speech object.

newObjectRef

On return, a new reference to the specified speech object.

Return Value
A result code. See "Speech Recognition Manager Result Codes" (page 1568).

Discussion
The original object reference (contained in `srObject`) and the new reference (returned in `newObjectRef`) may have different values. Accordingly, you cannot simply compare two object references to determine whether they are references to the same speech object.

SRGetReference increases the reference count of the specified speech object. You should call the SRReleaseObject function to release the object reference returned by SRGetReference when you are done using it.

Availability
Available in CarbonLib 1.0 and later when Speech Recognition 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
SpeechRecognition.h

SRIdle

Grants processing time to the Speech Recognition Manager if your application does not call WaitNextEvent frequently.

```
OSErr SRIdle (
   void
);
```

Return Value
A result code. See "Speech Recognition Manager Result Codes" (page 1568).

Discussion
The SRIdle function grants processing time to the Speech Recognition Manager, thereby allowing it to process incoming sound and send recognition results.

Most applications do not need to call the SRIdle function. You need to call it only if your application does a significant amount of processing without periodically calling WaitNextEvent. If you do use the SRIdle function, you should call it often enough that the Speech Recognition Manager can perform its work.

Note, however, that if you call SRIdle and not WaitNextEvent, you give time to the recognizer but not to the feedback window. You must call WaitNextEvent periodically to have the feedback animations work correctly if your recognizer is using the standard feedback window.

Availability
Available in CarbonLib 1.0 and later when Speech Recognition 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
SpeechRecognition.h

SRNewLanguageModel

Creates a new language model.

```
OSErr SRNewLanguageModel (
   SRRecognitionSystem system,
   SRLanguageModel *model,
   const void *name,
   SInt32 nameLength
);
```

Parameters

system

A recognition system.

model

On return, a reference to a new empty language model associated with the specified recognition system.

name

A pointer to a buffer that contains the name of the language model. The name of the language model should be unique among all the language models your application creates, and it should be comprehensible to users. (For example, a language model that defined a list of names might be called "«Names»").

The convention that language model names begin with the character "«" and end with the character "»" is adopted to support future utilities that display the names of language models to the user (perhaps as part of showing the user what he or she can say).

nameLength

The size, in bytes, of the specified name.

Return Value

A result code. See "Speech Recognition Manager Result Codes" (page 1568).

Discussion

You can add language objects (that is, words, phrases, paths, and other language models) to a language model by calling the `SRAddText` and `SRAddLanguageObject` functions.

`SRNewLanguageModel` sets the reference count of the specified language model to 1. You should call the `SRReleaseObject` function to release the language model reference returned by `SRNewLanguageModel` when you are done using it.

You can get or set the name of an existing language model by calling the `SRGetProperty` or `SRSetProperty` functions with the `kSRSpelling` property selector.

Availability

Available in CarbonLib 1.0 and later when Speech Recognition 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`SpeechRecognition.h`

SRNewLanguageObjectFromDataFile

Reads a language object from a data file.

```
OSErr SRNewLanguageObjectFromDataFile (
   SRRecognitionSystem system,
   SRLanguageObject *languageObject,
   short fRefNum
);
```

Parameters

system

A recognition system.

languageObject

On return, a reference to a new language object whose description is stored in the open data file that has the file reference number specified by the `fRefNum` parameter.

fRefNum

A file reference number of an open data file.

Return Value

A result code. See "Speech Recognition Manager Result Codes" (page 1568).

Discussion

`SRNewLanguageObjectFromDataFile` reads data beginning at the current file mark.

If the language object is successfully created and initialized, the file mark is left at the byte immediately following the language object description. Otherwise, if the language object data is not appropriately formatted, `SRNewLanguageObjectFromDataFile` returns the result code `kSRCantReadLanguageObject` as its function result and the file mark is not moved.

You should call the `SRReleaseObject` function to release the language object reference returned by `SRNewLanguageObjectFromDataFile` when you are done using it.

Availability

Available in CarbonLib 1.0 and later when Speech Recognition 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`SpeechRecognition.h`

SRNewLanguageObjectFromHandle

Creates a language object from the handle previously created by the `SRPutLanguageObjectIntoHandle` function.

```
OSErr SRNewLanguageObjectFromHandle (
   SRRecognitionSystem system,
   SRLanguageObject *languageObject,
   Handle lObjHandle
);
```

Parameters

system

A recognition system.

languageObject

On return, a reference to a new language object created and initialized using the private data to which the `lobjHandle` parameter is a handle.

lObjHandle

A handle to a language object. The data specified by `lobjHandle` should have been created by a previous call to the `SRPutLanguageObjectIntoHandle` function; if that data is not appropriately formatted, `SRNewLanguageObjectFromHandle` returns the result code `kSRCantReadLanguageObject` as its function result.

Return Value
A result code. See "Speech Recognition Manager Result Codes" (page 1568).

Discussion
You can use this function to load language objects from resources (for example, by using the Resource Manager function `GetResource`).

You should call the `SRReleaseObject` function to release the language object reference returned by `SRNewLanguageObjectFromHandle` when you are done using it.

Availability
Available in CarbonLib 1.0 and later when Speech Recognition 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechRecognition.h`

SRNewPath

Creates a new path.

```
OSErr SRNewPath (
   SRRecognitionSystem system,
   SRPath *path
);
```

Parameters

system

A recognition system.

path

On return, a reference to a new empty path associated with the specified recognition system.

Return Value
A result code. See "Speech Recognition Manager Result Codes" (page 1568).

Discussion
You can then add objects to a path by calling the `SRAddText` or `SRAddLanguageObject` functions.

You should call the `SRReleaseObject` function to release the path reference returned by `SRNewPath` when you are done using it.

Availability
Available in CarbonLib 1.0 and later when Speech Recognition 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechRecognition.h`

SRNewPhrase

Creates a new phrase.

```
OSErr SRNewPhrase (
   SRRecognitionSystem system,
   SRPhrase *phrase,
   const void *text,
   SInt32 textLength
);
```

Parameters

system

A recognition system.

phrase

On return, a reference to a new phrase associated with the specified recognition system.

text

A pointer to a buffer that contains the words that comprise the phrase.

textLength

The size, in bytes, of the specified text.

Return Value

A result code. See "Speech Recognition Manager Result Codes" (page 1568).

Discussion

The phrase's contents (that is, the words that comprise the phrase) is specified by the `text` and `textLength` parameters. You can, if you wish, create a new empty phrase and then add words to it by calling the `SRAddText` or `SRAddLanguageObject` functions.

You should call the `SRReleaseObject` function to release the phrase reference returned by `SRNewPhrase` when you are done using it.

Availability

Available in CarbonLib 1.0 and later when Speech Recognition 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`SpeechRecognition.h`

SRNewRecognizer

Creates a new recognizer.

```
OSErr SRNewRecognizer (
   SRRecognitionSystem system,
   SRRecognizer *recognizer,
   OSType sourceID
);
```

Parameters

system

A recognition system.

recognizer
On return, a reference to a new recognizer associated with the specified recognition system and using the specified speech source.

sourceID
A speech source ID. See "Speech Source Constants" (page 1567).

Return Value
A result code. See "Speech Recognition Manager Result Codes" (page 1568).

Discussion
`SRNewRecognizer` may need to load substantial amounts of data from disk into memory. As a result, you might want to change the cursor to the watch cursor before you call `SRNewRecognizer`.

You should call the `SRReleaseObject` function to release the object reference returned by `SRNewRecognizer` when you are done using it.

Availability
Available in CarbonLib 1.0 and later when Speech Recognition 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechRecognition.h`

SRNewWord

Creates a new word.

```
OSErr SRNewWord (
   SRRecognitionSystem system,
   SRWord *word,
   const void *text,
   SInt32 textLength
);
```

Parameters
system
A recognition system.

word
On return, a reference to a new word associated with the specified recognition system.

text
A pointer to a buffer that contains the characters that comprise the word.

textLength
The size, in bytes, of the specified text.

Return Value
A result code. See "Speech Recognition Manager Result Codes" (page 1568).

Discussion
You should call the `SRReleaseObject` function to release the word reference returned by `SRNewWord` when you are done using it.

Availability
Available in CarbonLib 1.0 and later when Speech Recognition 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In
SpeechRecognition.h

SROpenRecognitionSystem

Opens a recognition system.

```
OSErr SROpenRecognitionSystem (
   SRRecognitionSystem *system,
   OSType systemID
);
```

Parameters

system
On return, a reference to the recognition system having the specified system ID.

systemID
A recognition system ID.

Return Value
A result code. See "Speech Recognition Manager Result Codes" (page 1568).

Discussion
Generally, you should open a single recognition system when your application starts up and close it (by calling the function SRCloseRecognitionSystem) before your application exits.

Availability
Available in CarbonLib 1.0 and later when Speech Recognition 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
SpeechRecognition.h

SRProcessBegin

Indicates that a recognition result is being processed.

```
OSErr SRProcessBegin (
   SRRecognizer recognizer,
   Boolean failed
);
```

Parameters

recognizer
A recognizer.

failed
A Boolean value that determines how the feedback gestures are to be altered and whether the response sound is to be played (FALSE) or not (TRUE).

Return Value
A result code. See "Speech Recognition Manager Result Codes" (page 1568).

Discussion
The `SRProcessBegin` function causes the Speech Recognition Manager to provide the relevant feedback (in the feedback window associated with the recognizer specified by the `recognizer` parameter) indicating that the application is in the process of responding to a spoken command. Currently, the gestures of the feedback character are changed to indicate that processing is occurring.

If you set the value of the recognizer's `kSRWantsAutoFBGestures` property to `FALSE`, you should call `SRProcessBegin` at the beginning of your response to a recognition result and `SRProcessEnd` at the end of your response. During the interval separating the two calls, the feedback character displays an appropriate set of gestures showing the user that the task is being processed. If you pass the value `TRUE` in the `failed` parameter (indicating that the recognition result cannot successfully be processed), the feedback character displays frowns, shrugs, or other appropriate gestures. In addition, when `failed` is `TRUE`, you do not need to call `SRProcessEnd` to end the processing. If you pass the value `FALSE` in the `failed` parameter but determine subsequently that the recognition result cannot successfully be processed, you should call `SRProcessEnd` with the `failed` parameter set to `TRUE`.

If the value of the `kSRWantsAutoFBGestures` property of the specified recognizer is `TRUE`, the Speech Recognition Manager calls `SRProcessBegin` internally before notifying your application of a recognition result, and it calls `SRProcessEnd` internally after your application is notified. As a result, you should call `SRProcessBegin` or `SRProcessEnd` only when the value of the recognizer's `kSRWantsAutoFBGestures` property is `FALSE`.

Because the default value of the `kSRWantsAutoFBGestures` property is `TRUE`, most applications do not need to call `SRProcessBegin`. Calling `SRProcessBegin` is useful, however, when you know the resulting action might take a significant amount of time.

Availability
Available in CarbonLib 1.0 and later when Speech Recognition 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechRecognition.h`

SRProcessEnd

Indicates that a recognition result is done being processed.

```
OSErr SRProcessEnd (
   SRRecognizer recognizer,
   Boolean failed
);
```

Parameters

`recognizer`
A recognizer.

`failed`
A Boolean value that determines how the feedback gestures are to be altered (`FALSE`) or not (`TRUE`).

Return Value
A result code. See "Speech Recognition Manager Result Codes" (page 1568).

Discussion
The `SRProcessEnd` function causes the Speech Recognition Manager to provide the relevant feedback (in the feedback window associated with the recognizer specified by the `recognizer` parameter) indicating that a recognition result is done being processed. Currently, the gestures of the feedback character are changed and a response sound is played.

If the value of the `kSRWantsAutoFBGestures` property of the specified recognizer is `TRUE`, the Speech Recognition Manager calls `SRProcessBegin` internally before notifying your application of a recognition result, and it calls `SRProcessEnd` internally after your application is notified. As a result, you should call `SRProcessBegin` or `SRProcessEnd` only when the value of the recognizer's `kSRWantsAutoFBGestures` property is `FALSE`.

Because the default value of the `kSRWantsAutoFBGestures` property is `TRUE`, most applications do not need to call `SRProcessBegin`. Calling `SRProcessBegin` is useful, however, when you know the resulting action might take a significant amount of time.

Availability
Available in CarbonLib 1.0 and later when Speech Recognition 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechRecognition.h`

SRPutLanguageObjectIntoDataFile

Puts a language object (and any embedded language objects it contains) into a data file.

```
OSErr SRPutLanguageObjectIntoDataFile (
   SRLanguageObject languageObject,
   short fRefNum
);
```

Parameters

`languageObject`
A language object.

`fRefNum`
A file reference number of an open data file into which the data describing the specified language object is to be put.

Return Value
A result code. See "Speech Recognition Manager Result Codes" (page 1568).

Discussion
The `SRPutLanguageObjectIntoDataFile` function puts a description of the language object specified by the `languageObject` parameter into the data file specified by the `fRefNum` parameter. Data are written starting at the current file mark, and the file mark is moved to the end of the written data.

Availability
Available in CarbonLib 1.0 and later when Speech Recognition 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechRecognition.h`

SRPutLanguageObjectIntoHandle

Puts a language object (and any embedded languages objects it contains) into a handle.

```
OSErr SRPutLanguageObjectIntoHandle (
   SRLanguageObject languageObject,
   Handle lobjHandle
);
```

Parameters

languageObject

A language object.

lobjHandle

A handle to a block of memory into which the data describing the specified language object is to be put. On entry, this handle can have a length of 0.

Return Value

A result code. See "Speech Recognition Manager Result Codes" (page 1568).

Discussion

The `SRPutLanguageObjectIntoHandle` function puts a description of the language object specified by the `languageObject` parameter into the block of memory specified by the `lobjHandle` parameter. This replaces the data in the handle and resizes the handle if necessary.

You can use Resource Manager functions (such as `AddResource`) to store language objects into resources.

Availability

Available in CarbonLib 1.0 and later when Speech Recognition 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In

`SpeechRecognition.h`

SRReleaseObject

Releases a speech object.

```
OSErr SRReleaseObject (
   SRSpeechObject srObject
);
```

Parameters

srObject

A speech object.

Return Value

A result code. See "Speech Recognition Manager Result Codes" (page 1568).

Discussion

If there are no other remaining references to the object, `SRReleaseObject` disposes of the memory occupied by the object.

Your application should balance every function call that returns an object reference with a call to `SRReleaseObject`. This means that every call to a function whose name begins with `SRNew` or `SRGet` that successfully returns an object reference must be balanced with a call to `SRReleaseObject`.

In addition, you should call `SRReleaseObject` to release references to `SRSearchResult` objects that are passed to your application (via an Apple event handler or a callback routine).

Availability
Available in CarbonLib 1.0 and later when Speech Recognition 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechRecognition.h`

SRRemoveIndexedItem

Removes a subitem from a container object.

```
OSErr SRRemoveIndexedItem (
   SRSpeechObject container,
   long index
);
```

Parameters

`container`
A speech object.

`index`
An integer ranging from 0 to one less than the number of subitems in the specified speech object. (You can call the `SRCountItems` function to determine the number of subitems contained in a speech object.) If the index you specify is not in this range, `SRRemoveIndexedItem` returns the result code `kSRParamOutOfRange`.

Return Value
A result code. See "Speech Recognition Manager Result Codes" (page 1568).

Discussion
The `SRRemoveIndexedItem` function removes from the speech object specified by the `container` parameter the subitem located at the position specified by the `index` parameter. If `SRRemoveIndexedItem` completes successfully, the number of subitems in the container object is reduced by 1, and the index of each subitem that follows the removed item is reduced by 1.

Version Notes
In Speech Recognition Manager version 1.5, this function is useful only for operating on language objects (of type `SRLanguageObject`), although it is defined for all speech objects.

Availability
Available in CarbonLib 1.0 and later when Speech Recognition 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechRecognition.h`

SRRemoveLanguageObject

Removes a language object from another language object that contains it.

```
OSErr SRRemoveLanguageObject (
   SRLanguageObject base,
   SRLanguageObject toRemove
);
```

Parameters

base

The language object containing the language object to remove.

toRemove

The language object to remove.

Return Value
A result code. See "Speech Recognition Manager Result Codes" (page 1568).

Discussion
The object specified by the `base` parameter should be a container one of whose subitems is the object specified by the `toRemove` parameter.

Availability
Available in CarbonLib 1.0 and later when Speech Recognition 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechRecognition.h`

SRSetIndexedItem

Replaces a subitem in a container object with some other object.

```
OSErr SRSetIndexedItem (
   SRSpeechObject container,
   SRSpeechObject item,
   long index
);
```

Parameters

container

A speech object.

item

A speech object.

index

An integer ranging from 0 to one less than the number of subitems in the specified speech object.

Return Value
A result code. See "Speech Recognition Manager Result Codes" (page 1568).

Discussion
The `SRSetIndexedItem` function replaces the subitem having the index specified by the `index` parameter in the container object specified by the `container` parameter with the speech object specified by the `item` parameter. A reference to the replacement item is maintained separately by the container; as a result, you can release any reference to that item if you no longer need it. The reference to the replaced item is removed from the container; if that reference was the last remaining reference to the object, the object is released.

Version Notes
In Speech Recognition Manager version 1.5, this function is useful only for operating on language objects (of type `SRLanguageObject`), although it is defined for all speech objects.

Availability
Available in CarbonLib 1.0 and later when Speech Recognition 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechRecognition.h`

SRSetLanguageModel

Sets a recognizer's active language model.

```
OSErr SRSetLanguageModel (
   SRRecognizer recognizer,
   SRLanguageModel languageModel
);
```

Parameters

recognizer
A recognizer.

languageModel
The language model you wish to become the active model for the specified recognizer.

Return Value
A result code. See "Speech Recognition Manager Result Codes" (page 1568).

Discussion
If no other references exist to the language model currently in use by the specified recognizer, calling `SRSetLanguageModel` with a different language model causes the current one to be released.

Availability
Available in CarbonLib 1.0 and later when Speech Recognition 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechRecognition.h`

SRSetProperty

Sets the value of a property of a speech object.

```
OSErr SRSetProperty (
   SRSpeechObject srObject,
   OSType selector,
   const void *property,
   Size propertyLen
);
```

Parameters

srObject

A speech object.

selector

A property selector. See "Recognizer Properties" (page 1564), "Recognizer Listen Key Properties" (page 1563), "Language Object Properties" (page 1557), and "Recognition System Properties" (page 1562) for lists of the available property selectors.

property

A pointer to a buffer containing the value to which the specified property is to be set.

propertyLen

The length, in bytes, of the specified buffer.

Return Value

A result code. See "Speech Recognition Manager Result Codes" (page 1568).

Discussion

The `SRSetProperty` function sets the value of the property of the speech object specified by the `srObject` parameter to the value specified through the `property` parameter. The `selector` parameter specifies which property is to be set and the `propertyLen` parameter specifies it size, in bytes.

Not all properties can be set. If you attempt to set a property that cannot be set, `SRSetProperty` returns the result code `kSRCantSetProperty` or `kSRBadSelector` as its function result.

Availability

Available in CarbonLib 1.0 and later when Speech Recognition 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`SpeechRecognition.h`

SRSpeakAndDrawText

Draws output text in the feedback window and causes the feedback character in the feedback window to speak that text.

```
OSErr SRSpeakAndDrawText (
   SRRecognizer recognizer,
   const void *text,
   SInt32 textLength
);
```

Parameters

recognizer

A recognizer.

text

A pointer to a buffer that contains the text to be drawn and spoken.

textLength
The size, in bytes, of the specified text.

Return Value
A result code. See "Speech Recognition Manager Result Codes" (page 1568).

Availability
Available in CarbonLib 1.0 and later when Speech Recognition 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechRecognition.h`

SRSpeakText

Causes the feedback character in the feedback window to speak a text string.

```
OSErr SRSpeakText (
   SRRecognizer recognizer,
   const void *speakText,
   SInt32 speakLength
);
```

Parameters
recognizer
A recognizer.

speakText
A pointer to a buffer that contains the text to be spoken. The text pointed to by the `speakText` parameter can contain embedded speech commands to enhance the prosody of the spoken string.

speakLength
The size, in bytes, of the specified text.

Return Value
A result code. See "Speech Recognition Manager Result Codes" (page 1568).

Discussion
While speaking, the feedback character lip-syncs the spoken string using the Speech Synthesis Manager's phoneme callback routines. `SRSpeakText` uses the default voice and rate selected in the Speech control panel. (The Speech Synthesis Manager was formerly called the Speech Manager. Its name has been changed to distinguish it from the Speech Recognition Manager and to describe its operation more clearly.)

You can use the `SRSpeechBusy` function to determine whether the feedback character is already speaking. If it is, you can call the `SRStopSpeech` function to stop that speaking immediately.

The `SRSpeakText` function speaks the specified text but does not display it. Use the `SRSpeakAndDrawText` function if you want to speak and display the text.

Availability
Available in CarbonLib 1.0 and later when Speech Recognition 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In
`SpeechRecognition.h`

SRSpeechBusy

Determines if the feedback character in a feedback window is currently speaking.

```
Boolean SRSpeechBusy (
   SRRecognizer recognizer
);
```

Parameters

recognizer

A recognizer.

Return Value

On return, `true` if the feedback character in the feedback window associated with the recognizer specified by the `recognizer` parameter is currently speaking; otherwise `false`.

Availability

Available in CarbonLib 1.0 and later when Speech Recognition 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`SpeechRecognition.h`

SRStartListening

Starts a recognizer listening and reporting results to your application.

```
OSErr SRStartListening (
   SRRecognizer recognizer
);
```

Parameters

recognizer

A recognizer.

Return Value

A result code. See "Speech Recognition Manager Result Codes" (page 1568).

Discussion

The `SRStartListening` function instructs the recognizer specified by the `recognizer` parameter to begin processing sound from its speech source and reporting its results to your application (either using Apple events or through a speech recognition callback routine).

You must already have built a language model and attached it to the recognizer (by calling the `SRSetLanguageModel` function) before you call `SRStartListening`.

Availability

Available in CarbonLib 1.0 and later when Speech Recognition 1.0 or later is present.

Available in Mac OS X 10.0 and later.

Declared In

`SpeechRecognition.h`

SRStopListening

Stops a recognizer listening and reporting results to your application.

```
OSErr SRStopListening (
   SRRecognizer recognizer
);
```

Parameters

recognizer
A recognizer.

Return Value

A result code. See "Speech Recognition Manager Result Codes" (page 1568).

Discussion

The `SRStopListening` function instructs the recognizer specified by the `recognizer` parameter to stop processing sound from its speech source and reporting its results to your application.

Availability

Available in CarbonLib 1.0 and later when Speech Recognition 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In

`SpeechRecognition.h`

SRStopSpeech

Terminates speech by the feedback character in a feedback window.

```
OSErr SRStopSpeech (
   SRRecognizer recognizer
);
```

Parameters

recognizer
A recognizer.

Return Value

A result code. See "Speech Recognition Manager Result Codes" (page 1568).

Discussion

The `SRStopSpeech` function immediately terminates any speaking by the feedback character in the feedback window associated with the recognizer specified by the `recognizer` parameter.

Availability

Available in CarbonLib 1.0 and later when Speech Recognition 1.0 or later is present.
Available in Mac OS X 10.0 and later.

Declared In

`SpeechRecognition.h`

Callbacks

SRCallBackProcPtr

Defines a pointer to a speech recognition callback function which is called whenever the recognizer encounters one of the events specified in its `kSRNotificationParam` property.

```
typedef void (*SRCallBackProcPtr) (
    SRCallBackStruct * param
);
```

If you name your function `MySRCallBackProc`, you would declare it like this:

```
void MySRCallBackProc (
    SRCallBackStruct * param
);
```

Parameters

param

A pointer to a speech recognition callback structure. See `SRCallBackStruct` (page 1549) for a description of this structure.

Discussion

You can receive notification of recognizer events either by installing an Apple event handler or by installing a speech recognition callback function. In general, you should use an Apple event handler to process recognition notifications. You should use callbacks only for executable code that cannot easily receive Apple events.

You can determine what event caused your function to be called by inspecting the `what` field of the speech recognition callback structure specified by the `param` parameter.

Because the Speech Recognition Manager is not fully reentrant, you should not call any of its functions other than `SRContinueRecognition` or `SRCancelRecognition` from within your speech recognition callback. Accordingly, your callback should simply queue the notification for later processing by your software (for instance, when it receives background processing time).

If the event is of type `kSRNotifyRecognitionBeginning` (which occurs only if you request speech-begun notifications), you must call either `SRContinueRecognition` or `SRCancelRecognition` before speech recognition can continue. A recognizer that has issued a recognition notification suspends activity until you call one of these two functions.

In general, when your speech recognition callback receives the `kSRNotifyRecognitionBeginning` notification, it should queue an indication for your main code both to adjust the current language model (if necessary) and to call the `SRContinueRecognition` function. When your callback receives the `kSRNotifyRecognitionDone` notification, it should queue an indication for your main code to handle the recognition result passed in the `message` field of the speech recognition callback structure specified by the `param` parameter. You should make sure, however, that the `message` field contains a valid reference to a recognition result by inspecting the `status` field of that structure; if `status` contains any value other than `noErr`, the contents of the `message` field are undefined.

When your callback is executed, your application is not the current process. As a result, some restrictions apply; for example, the current resource chain might not be that of your application.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`SpeechRecognition.h`

Data Types

SRCallBackParam

Defines a speech recognition callback parameter structure.

```
struct SRCallBackParam {
    SRCallBackUPP callBack;
    long refCon;
};
typedef struct SRCallBackParam SRCallBackParam;
```

Fields

`callBack`
: A UPP for a speech recognition callback function. You can use the function `NewSRCallBackUPP` to create this UPP.

`refCon`
: An application-defined reference constant. This value is passed to your callback routine in the `refcon` field of a speech recognition callback structure. You can pass any 4-byte value you wish.

Discussion
If you want to receive recognition notifications using a speech recognition callback routine instead of an Apple event handler, you must change the value of the `kSRCallBackParam` property of the current recognizer. The value of the `kSRCallBackParam` property is the address of a callback function parameter structure, defined by the `SRCallBackParam` data type.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`SpeechRecognition.h`

SRCallBackStruct

Defines a structure sent to your speech recognition callback function.

```
struct SRCallBackStruct {
    long what;
    long message;
    SRRecognizer instance;
    OSErr status;
    short flags;
    long refCon;
};
typedef struct SRCallBackStruct SRCallBackStruct;
```

Fields

`what`

A notification flag that indicates the kind of event that caused this notification to be issued. This field contains either `kSRNotifyRecognitionBeginning` or `kSRNotifyRecognitionDone`. See "Notification Flags" (page 1560) for complete details on the available notification flags.

`message`

If the value of the `status` field is `noErr` and the value of the `what` field is `kSRNotifyRecognitionDone`, this field contains a reference to a recognition result. Your callback routine can inspect the properties of this recognition result to determine what the user said.

Note that your callback routine must release this reference (by calling `SRReleaseObject`) when it is finished using it. If the value of the `status` field is not `noErr`, the value of this field is undefined.

`instance`

A reference to the recognizer that issued this notification. You should not call `SRReleaseObject` on this recognizer reference in response to a recognition notification.

`status`

An error code indicating the status of the recognition. If the value of this field is `noErr`, the `message` field contains a reference to a recognition result. If the value of this field is `kSRRecognitionDone` and the value of the `what` field is `kSRNotifyRecognitionDone`, the recognizer finished without error but nothing belonging to that recognizer was recognized; in this case, the `message` field does not contain a reference to a recognition result. If the value of this field is any other value, some other error occurred.

`flags`

Reserved for use by Apple Computer, Inc.

`refCon`

An application-defined reference constant. The value in this field is the value you passed in the `refcon` field of a callback function parameter structure (of type `SRCallBackParam`).

Discussion

When you receive a notification of recognition results through an application-defined callback function (instead of using an Apple event handler), the Speech Recognition Manager sends your callback function a pointer to a speech recognition callback structure, defined by the `SRCallBackStruct` data type.

For information on writing a speech recognition callback function, see `SRCallBackProcPtr` (page 1548).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`SpeechRecognition.h`

SRCallBackUPP

Defines a universal procedure pointer (UPP) to a speech recognition callback function.

```
typedef SRCallBackProcPtr SRCallBackUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`SpeechRecognition.h`

SRLanguageModel

Represents a language model.

```
typedef SRLanguageObject SRLanguageModel;
```

Discussion
A language model is a list of zero or more words, phrases, or paths.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`SpeechRecognition.h`

SRLanguageObject

Represents a language object.

```
typedef SRSpeechObject SRLanguageObject;
```

Discussion
A language model is built using four kinds of objects, collectively called language objects: words, phrases, paths, and language models.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`SpeechRecognition.h`

SRPath

Represents a language object identifying a path.

```
typedef SRLanguageObject SRPath;
```

Discussion
A path is a sequence of zero or more words, phrases, or language models.

Availability
Available in Mac OS X v10.0 and later.

Declared In
SpeechRecognition.h

SRPhrase

Represents a language object identifying a phrase.

```
typedef SRLanguageObject SRPhrase;
```

Discussion
A phrase is a sequence of zero or more words.

Availability
Available in Mac OS X v10.0 and later.

Declared In
SpeechRecognition.h

SRRecognitionResult

Represents a recognition result which contains information about a recognized utterance.

```
typedef SRSpeechSource SRRecognitionResult;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
SpeechRecognition.h

SRRecognitionSystem

Represents a speech object identifying a recognition system.

```
typedef SRSpeechObject SRRecognitionSystem;
```

Discussion
A speech object is an instance of a speech class, which defines a set of properties for objects in the class. The behavior of a speech object is determined by the set of properties associated with the object's class. Recognition systems have the properties associated with the `SRRecognitionSystem` class of speech objects.

Availability
Available in Mac OS X v10.0 and later.

Declared In
SpeechRecognition.h

SRRecognizer

Represents a speech object identifying a speech recognizer.

```
typedef SRSpeechObject SRRecognizer;
```

Discussion
A speech object is an instance of a speech class, which defines a set of properties for objects in the class. The behavior of a speech object is determined by the set of properties associated with the object's class. Speech recognizers have the properties associated with the `SRRecognizer` class of speech objects.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`SpeechRecognition.h`

SRRejectionLevel

```
typedef  SRRejectionLevel;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`SpeechRecognition.h`

SRSpeechObject

Defines a reference to a speech object.

```
typedef struct OpaqueSRSpeechObject * SRSpeechObject;
```

Discussion
The Speech Recognition Manager is object oriented in the sense that many of its capabilities are accessed by creating and manipulating speech objects. A speech object is an instance of a speech class, which defines a set of properties for objects in the class. The behavior of a speech object is determined by the set of properties associated with the object's class.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`SpeechRecognition.h`

SRSpeechSource

Represents a speech object identifying a speech source.

```
typedef SRSpeechObject SRSpeechSource;
```

Discussion
A speech object is an instance of a speech class, which defines a set of properties for objects in the class. The behavior of a speech object is determined by the set of properties associated with the object's class. Speech sources have the properties associated with the `SRSpeechSource` class of speech objects.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`SpeechRecognition.h`

SRSpeedSetting

```
typedef  SRSpeedSetting;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`SpeechRecognition.h`

SRWord

Represents a language object identifying a word.

```
typedef SRLanguageObject SRWord;
```

Discussion
A word represents a single speakable word.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`SpeechRecognition.h`

Constants

Feedback and Listening Modes

Identify the feedback and listening modes of the recognition system.

```
enum {
    kSRNoFeedbackNoListenModes = 0,
    kSRHasFeedbackHasListenModes = 1,
    kSRNoFeedbackHasListenModes = 2
};
```

Constants

`kSRNoFeedbackNoListenModes`

If the feedback and listening modes value of a recognition system is set to `kSRNoFeedbackNoListenModes`, the next created recognizer has no feedback window and does not use the listening modes selected by the user in the Speech control panel. (For example, push-to-talk is a listening mode.)

Available in Mac OS X v10.0 and later.

Declared in `SpeechRecognition.h`.

`kSRHasFeedbackHasListenModes`

If the feedback and listening modes value of a recognition system is set to `kSRHasFeedbackHasListenModes`, the next created recognizer opens a feedback window that uses the listening modes selected by the user in the Speech control panel.

Available in Mac OS X v10.0 and later.

Declared in `SpeechRecognition.h`.

`kSRNoFeedbackHasListenModes`

If the feedback and listening modes value of a recognition system is set to `kSRNoFeedbackHasListenModes`, the next created recognizer has no feedback window but does use the listening modes selected by the user in the Speech control panel.

Available in Mac OS X v10.0 and later.

Declared in `SpeechRecognition.h`.

Apple Event Selectors

Define selectors that you can use to handle recognition notifications in your Apple event handler.

```
enum {
    kAESpeechDone = 'srsd',
    kAESpeechDetected = 'srbd'
};
enum {
    kAESpeechSuite = 'sprc'
};
enum {
    keySRRecognizer = 'krec',
    keySRSpeechResult = 'kspr',
    keySRSpeechStatus = 'ksst'
};
enum {
    typeSRRecognizer = 'trec',
    typeSRSpeechResult = 'tspr'
};
```

Constants

`kAESpeechDone`
The message ID for a speech-done event.
Available in Mac OS X v10.0 and later.
Declared in `SpeechRecognition.h`.

`kAESpeechDetected`
The message ID for a speech-detected event.
Available in Mac OS X v10.0 and later.
Declared in `SpeechRecognition.h`.

`kAESpeechSuite`
The Apple event suite for speech recognition events.
Available in Mac OS X v10.0 and later.
Declared in `SpeechRecognition.h`.

`keySRRecognizer`
The ID for the recognizer parameter.
Available in Mac OS X v10.0 and later.
Declared in `SpeechRecognition.h`.

`keySRSpeechResult`
The ID for the recognition result parameter.
Available in Mac OS X v10.0 and later.
Declared in `SpeechRecognition.h`.

`keySRSpeechStatus`
The ID for the speech status parameter, which is of type `typeShortInteger`.
Available in Mac OS X v10.0 and later.
Declared in `SpeechRecognition.h`.

`typeSRRecognizer`
The type for the recognizer parameter.
Available in Mac OS X v10.0 and later.
Declared in `SpeechRecognition.h`.

`typeSRSpeechResult`
The type for the recognition result parameter.

Available in Mac OS X v10.0 and later.

Declared in `SpeechRecognition.h`.

Discussion

Default Rejection Level

Represents a default rejection level.

```
enum {
    kSRDefaultRejectionLevel = 50
};
```

Language Object Properties

Define property selectors for language objects.

```
enum {
    kSRSpelling = 'spel',
    kSRLMObjType = 'lmtp',
    kSRRefCon = 'refc',
    kSROptional = 'optl',
    kSREnabled = 'enbl',
    kSRRepeatable = 'rptb',
    kSRRejectable = 'rjbl',
    kSRRejectionLevel = 'rjct'
};
```

Constants

`kSRSpelling`
The spelling of a language object. The value of this property is a variable-length string of characters. For an object of type `SRWord`, the value is the spelled word. For an object of type `SRPhrase`, the value is the concatenation of the spellings of each word in the phrase, separated by a language-dependent separation character (for example, by a space character). For an object of type `SRPath`, the value is the concatenation of the spellings of each word and language model name in the path. For an object of type `SRLanguageModel`, the value is the name of the language model. For any object, the string value does not include either a length byte (as in Pascal strings) or a null terminating character (as in C strings).

Available in Mac OS X v10.0 and later.

Declared in `SpeechRecognition.h`.

`kSRLMObjType`
The type of a language object. The value of this property is a four-character constant of type `OSType`; see the section "Language Object Types" (page 1559) for the values that are defined for this property. You cannot set a property of this type.

Available in Mac OS X v10.0 and later.

Declared in `SpeechRecognition.h`.

`kSRRefCon`

The reference constant. The value of this property is a 4-byte value specified by your application. By default, the value of a reference constant property is zero (0).

Available in Mac OS X v10.0 and later.

Declared in `SpeechRecognition.h`.

`kSROptional`

The optional flag. The value of this property is a Boolean value that indicates whether speaking the words, phrases, paths, and language models represented by the object is optional (`TRUE`) or required (`FALSE`). A user is not required to utter optional words, phrases, or language models. By default, the value of an object's optional flag is `FALSE`.

Available in Mac OS X v10.0 and later.

Declared in `SpeechRecognition.h`.

`kSREnabled`

The enabled flag. The value of this property is a Boolean value that indicates whether the object is enabled (`TRUE`) or disabled (`FALSE`). Disabled objects are ignored during speech recognition. By default, the value of an object's enabled flag is `TRUE`.

Available in Mac OS X v10.0 and later.

Declared in `SpeechRecognition.h`.

`kSRRepeatable`

The repeatable flag. The value of this property is a Boolean value that indicates whether the object is repeatable (`TRUE`) or not (`FALSE`). A user can utter a repeatable object more than once. By default, the value of an object's repeatable flag is `FALSE`.

Available in Mac OS X v10.0 and later.

Declared in `SpeechRecognition.h`.

`kSRRejectable`

The rejectable flag. The value of this property is a Boolean value that indicates whether the object is rejectable (`TRUE`) or not (`FALSE`). An object is rejectable if a recognition system can return the rejected word instead of that object. (The rejected word is the value of the `kSRRejectedWord` property of the recognition system.) By default, the value of an object's rejectable flag is `FALSE`. However, if an entire utterance is rejected, you can still get the rejected word. See "Recognition Result Properties" (page 1561).

Available in Mac OS X v10.0 and later.

Declared in `SpeechRecognition.h`.

`kSRRejectionLevel`

The rejection level. The value of this property is a 2-byte unsigned integer of type `SRRejectionLevel` between 0 and 100, inclusive, that determines how likely a recognizer is to reject a language object whose `kSRRejectable` property is `TRUE`. If an object's rejection level is close to 0, the recognizer is less likely to reject utterances (and hence more likely to return a result with phrases from the current language model, whether or not the user actually said something in that language model) if an object's rejection level is close to 100, the recognizer is more likely to reject utterances. You can set an object's rejection flag to `TRUE` and its rejection level to some appropriate value to reduce the likelihood that a recognizer will mistakenly recognize a random user utterance as part of the active language model. By default, the value of an object's rejection level is `50`.

Available in Mac OS X v10.0 and later.

Declared in `SpeechRecognition.h`.

Discussion
Every language object (that is, any instance of a subclass of the `SRLanguageObject` class) has a set of properties that you can inspect and change by calling the `SRGetProperty` and `SRSetProperty` functions. You specify a property by passing a property selector to those functions.

Language Object Types

Identify the four subclasses of the `SRLanguageObject` class.

```
enum {
    kSRLanguageModelType = 'lmob',
    kSRPathType = 'path',
    kSRPhraseType = 'phra',
    kSRWordType = 'word'
};
```

Constants

`kSRLanguageModelType`
A language model (that is, an object of type `SRLanguageModel`).
Available in Mac OS X v10.0 and later.
Declared in `SpeechRecognition.h`.

`kSRPathType`
A path (that is, an object of type `SRPath`).
Available in Mac OS X v10.0 and later.
Declared in `SpeechRecognition.h`.

`kSRPhraseType`
A phrase (that is, an object of type `SRPhrase`).
Available in Mac OS X v10.0 and later.
Declared in `SpeechRecognition.h`.

`kSRWordType`
A word (that is, an object of type `SRWord`).
Available in Mac OS X v10.0 and later.
Declared in `SpeechRecognition.h`.

Discussion
You can use these constants, for instance, to help interpret the value of a language object's `kSRLMObjType` property.

Listen Key Modes

Identify listen key modes.

```
enum {
    kSRUseToggleListen = 0,
    kSRUsePushToTalk = 1
};
```

Constants

`kSRUseToggleListen`

The recognizer interprets presses on the listen key as a toggle to turn listening on or off.

Available in Mac OS X v10.0 and later.

Declared in `SpeechRecognition.h`.

`kSRUsePushToTalk`

The recognizer listens only when the listen key is held down.

Available in Mac OS X v10.0 and later.

Declared in `SpeechRecognition.h`.

Discussion

You can get (but not set) a recognizer's listen key mode by accessing its property of type `kSRListenKeyMode`. That property's value is a 2-byte unsigned integer that determines whether the listen key operates in push-to-talk or toggle-listening mode.

Notification Flags

Identify the recognizer events that may be sent to an application.

```
enum {
    kSRNotifyRecognitionBeginning = 1L << 0,
    kSRNotifyRecognitionDone = 1L << 1
};
```

Constants

`kSRNotifyRecognitionBeginning`

If this bit is set, your application will be notified when the user starts speaking and recognition is ready to begin. When your application gets this notification, it must call either `SRContinueRecognition` or `SRCancelRecognition` in order for recognition either to continue or be canceled. If you do not call one of these functions, the recognizer will simply wait until you do (and hence appear to have quit working). Note that calling `SRCancelRecognition` cancels a recognition only for the application that requested it, not for all applications.

Available in Mac OS X v10.0 and later.

Declared in `SpeechRecognition.h`.

`kSRNotifyRecognitionDone`

If this bit is set, your application will be notified when recognition is finished and the result (if any) of that recognition is available.

Available in Mac OS X v10.0 and later.

Declared in `SpeechRecognition.h`.

Discussion

You can indicate which recognizer events you want your application be to notified of by setting the recognizer's notification property, which is a property of type `kSRNotificationParam`. That property's value is a 4-byte unsigned integer. The Speech Recognition Manager defines these masks for bits in that value.

Recognition Result Properties

Identify property selectors for recognition results.

```
enum {
    kSRTEXTFormat = 'TEXT',
    kSRPhraseFormat = 'lmph',
    kSRPathFormat = 'lmpt',
    kSRLanguageModelFormat = 'lmfm'
};
```

Constants

`kSRTEXTFormat`

The text format. The value of this property is a variable-length string of characters that is the text of the recognized utterance. If the utterance was rejected, this text is the spelling of the rejected word. The string value does not include either a length byte (as in Pascal strings) or a null terminating character (as in C strings).

Available in Mac OS X v10.0 and later.

Declared in `SpeechRecognition.h`.

`kSRPhraseFormat`

The phrase format. The value of this property is a phrase that contains one word (of type `SRWord`) for each word in the recognized utterance. If the utterance was rejected, this path or phrase contains one object, the rejected word. The reference constant value of the phrase is always 0, but each word in the phrase retains its own reference constant property value.

Available in Mac OS X v10.0 and later.

Declared in `SpeechRecognition.h`.

`kSRPathFormat`

The path format. The value of this property is a path that contains a sequence of words (of type `SRWord`) and phrases (of type `SRPhrase`) representing the text of the recognized utterance. If the utterance was rejected, this path or phrase contains one object, the rejected word. The reference constant value of the path is always 0, but each word or phrase in the path retains its own reference constant property value.

Available in Mac OS X v10.0 and later.

Declared in `SpeechRecognition.h`.

`kSRLanguageModelFormat`

The language model format. The value of this property is a language model that contains a copy of each word, phrase, path, and language model used in the recognized utterance. If the utterance was rejected, the value of this property is the rejected word (that is, the `kSRRejectedWord` property of the recognition system). The name and reference constant of this language model are the same as the name and reference constant of the active language model, and each subitem in the language model retains its own reference constant property value.

Available in Mac OS X v10.0 and later.

Declared in `SpeechRecognition.h`.

Discussion

Every recognition result object has a set of properties that you can inspect by calling the `SRGetProperty` function. You specify a property by passing a property selector to those functions.

`SRGetProperty` returns an object reference as the value of a recognition result's `kSRPhraseFormat`, `kSRPathFormat`, or `kSRLanguageModelFormat` property. You must make sure to release that object reference (by calling `SRReleaseObject`) when you are finished using it.

Recognition System IDs

Defines a recognition system ID.

```
enum {
    kSRDefaultRecognitionSystemID = 0
};
```

Constants

`kSRDefaultRecognitionSystemID`

The default speech recognition system.

Available in Mac OS X v10.0 and later.

Declared in `SpeechRecognition.h`.

Discussion

When you call `SROpenRecognitionSystem` to open a recognition system, you indicate the system to open by passing a recognition system ID.

Recognition System Properties

Define property selectors for recognition systems.

```
enum {
    kSRFeedbackAndListeningModes = 'fbwn',
    kSRRejectedWord = 'rejq',
    kSRCleanupOnClientExit = 'clup'
};
```

Constants

`kSRFeedbackAndListeningModes`

The feedback and listening modes of the recognition system. The value of this property is an integer that determines some of the features of a recognizer subsequently created by your application. See "Feedback and Listening Modes" (page 1554) for a description of the values possible here.

The default value for version 1.5 is `kSRNoFeedbackNoListenModes`, but most applications should set this to `kSRHasFeedbackHasListenModes`.

Available in Mac OS X v10.0 and later.

Declared in `SpeechRecognition.h`.

`kSRRejectedWord`

The rejected word of the recognition system. The value of this property is a value of type `SRWord` that will be returned in a recognition result object when a recognizer encounters an unrecognizable utterance. For example, if an utterance is rejected, the `kSRLMObjType` property of the rejection result is the rejected word. By default, a recognition system's rejected word is spelled "???" and has a reference constant of 0.

Available in Mac OS X v10.0 and later.

Declared in `SpeechRecognition.h`.

`kSRCleanupOnClientExit`

The cleanup mode of the recognition system. Applications should never set this property. If, however, you do not have a process ID (as issued by the Process Manager), you should set this property to `FALSE` so that speech objects you allocate will not be associated with any other process. By default, the value of a recognition system's cleanup mode is `TRUE`.

Available in Mac OS X v10.0 and later.

Declared in `SpeechRecognition.h`.

Discussion

A recognition system (that is, an instance of the `SRRecognitionSystem` class) has a set of properties that you can inspect and change by calling the `SRGetProperty` and `SRSetProperty` functions. You specify a property by passing a property selector to those functions.

Recognizer Listen Key Properties

Define listen key property selectors for recognizers.

```
enum {
    kSRListenKeyMode = 'lkmd',
    kSRListenKeyCombo = 'lkey',
    kSRListenKeyName = 'lnam',
    kSRKeyWord = 'kwrd',
    kSRKeyExpected = 'kexp'
};
```

Constants

`kSRListenKeyMode`

The listen key mode. The value of this property is a 2-byte unsigned integer that indicates whether the listen key operates in push-to-talk or toggle-listening mode. See "Listen Key Modes" (page 1559) for a description of the available listen key modes. The value of a recognizer's listen key mode is whatever the user has selected in the Speech control panel. This property is read-only you cannot set a property of this type.

Available in Mac OS X v10.0 and later.

Declared in `SpeechRecognition.h`.

`kSRListenKeyCombo`

The listen key combination property. The value of this property is a 2-byte unsigned integer that specifies the key combination the user must press for the listen key. The high-order byte of this value has the same format as the high-order byte of the `modifiers` field of an event record. The low-order byte of this value has the same format as the key code contained in the `message` field of an event record. The value of a recognizer's listen key combination property is whatever the user has selected in the Speech control panel. This property is read-only you cannot set a property of this type.

Available in Mac OS X v10.0 and later.

Declared in `SpeechRecognition.h`.

`kSRListenKeyName`

The listen key name property. The value of this property is a string (of type `Str63`) that represents the listen key combination specified by the `kSRListenKeyCombo` property. The value of a recognizer's listen key name property is whatever the user has selected in the Speech control panel. This property is read-only you cannot set a property of this type.

Available in Mac OS X v10.0 and later.

Declared in `SpeechRecognition.h`.

`kSRKeyWord`
: The key word property. The value of this property is a string (of type `Str255`) that represents the key word that must precede utterances when the recognizer is in toggle-listen mode. The value of a recognizer's key word property is whatever the user has selected in the Speech control panel. This property is read-only you cannot set a property of this type.

  Available in Mac OS X v10.0 and later.

  Declared in `SpeechRecognition.h`.

`kSRKeyExpected`
: The key expected flag. The value of this property is a Boolean value that indicates whether the recognizer expects the user to hold down a key or to utter the key word in order to have the recognizer begin listening (`TRUE`) or not (`FALSE`). The value of a recognizer's key expected flag is a function of the user's Speech control panel selections. This property is `TRUE` whenever text is visible below the feedback character in the lower-left corner of the feedback window. This property is read-only.

  Available in Mac OS X v10.0 and later.

  Declared in `SpeechRecognition.h`.

Discussion
Every recognizer has a set of properties that you can inspect and change by calling the `SRGetProperty` and `SRSetProperty` functions. You specify a property by passing a property selector to those functions.

The listen key properties are provided for use by applications that want to provide their own visual feedback. If your application uses the default feedback mechanisms, you do not need to access those properties.

Recognizer Properties

Define property selectors for recognizers.

```
enum {
    kSRNotificationParam = 'noti',
    kSRCallBackParam = 'call',
    kSRSearchStatusParam = 'stat',
    kSRAutoFinishingParam = 'afin',
    kSRForegroundOnly = 'fgon',
    kSRBlockBackground = 'blbg',
    kSRBlockModally = 'blmd',
    kSRWantsResultTextDrawn = 'txfb',
    kSRWantsAutoFBGestures = 'dfbr',
    kSRSoundInVolume = 'volu',
    kSRReadAudioFSSpec = 'aurd',
    kSRCancelOnSoundOut = 'caso',
    kSRSpeedVsAccuracyParam = 'sped'
};
```

Constants
`kSRNotificationParam`
: The notification property. The value of this property is a 4-byte unsigned integer whose bits encode the kinds of events of which the recognizer will notify your application. See the section "Notification Flags" (page 1560) for the bit masks that are defined for this property. By default, the value of a recognizer's notification property is `kSRNotifyRecognitionDone`.

  Available in Mac OS X v10.0 and later.

  Declared in `SpeechRecognition.h`.

`kSRCallBackParam`

The callback property. The value of this property is of type `SRCallBackParam` that determines whether recognition notifications are sent to your application via Apple events or via an application-defined callback routine. To specify a callback routine, set the value of this property to the address of a callback routine parameter structure. By default, the value of a recognizer's callback property is `NULL`, indicating that Apple events are to be used to report recognizer events.

Available in Mac OS X v10.0 and later.

Declared in `SpeechRecognition.h`.

`kSRSearchStatusParam`

The search status. The value of this property is a 4-byte unsigned integer whose bits indicate the current state of the recognizer. See the section "Search Status Flags" (page 1567) for the bit masks that are defined for this property. This property is read-only; you cannot set a property of this type.

Available in Mac OS X v10.0 and later.

Declared in `SpeechRecognition.h`.

`kSRAutoFinishingParam`

Available in Mac OS X v10.0 and later.

Declared in `SpeechRecognition.h`.

`kSRForegroundOnly`

The foreground-only flag. The value of this property is a Boolean value that indicates whether the recognizer is enabled only when your application is the foreground application (`TRUE`) or not (`FALSE`). By default, the value of a recognizer's foreground-only flag is `TRUE`.

Available in Mac OS X v10.0 and later.

Declared in `SpeechRecognition.h`.

`kSRBlockBackground`

The background-blocking flag. The value of this property is a Boolean value that indicates whether all recognizers owned by other applications are automatically disabled whenever your application is the foreground application (`TRUE`) or are not automatically disabled (`FALSE`). By default, the value of a recognizer's background-blocking flag is `FALSE`.

Available in Mac OS X v10.0 and later.

Declared in `SpeechRecognition.h`.

`kSRBlockModally`

The modal-blocking flag. The value of this property is a Boolean value that indicates whether the language model associated with this recognizer is the only active language model (`TRUE`) or not (`FALSE`). When this flag is `TRUE`, your application's recognizer blocks those of other applications even when it is not the foreground application in addition, the feedback window is hidden if you are not using it. Setting this property to `TRUE` prevents speech recognition from working for other applications, so you want to use this property only if your application is taking over the computer (like some games) or briefly attempting to constrain the language model severely. By default, the value of a recognizer's modal-blocking flag is `FALSE`.

Available in Mac OS X v10.0 and later.

Declared in `SpeechRecognition.h`.

`kSRWantsResultTextDrawn`

The text feedback flag. The value of this property is a Boolean value that indicates whether the results of a search are to be automatically displayed as text in the feedback window (`TRUE`) or not (`FALSE`). If you set the value of this property to `FALSE`, you should call `SRDrawRecognizedText` with a string representing what the user said. By default, the value of a recognizer's text feedback flag is `TRUE`.

Available in Mac OS X v10.0 and later.

Declared in `SpeechRecognition.h`.

`kSRWantsAutoFBGestures`

The automatic feedback gestures flag. The value of this property is a Boolean value that determines whether the feedback gestures are automatically drawn (`TRUE`) or not (`FALSE`). If you want more control over feedback behavior, you should set this property to `FALSE`; then call `SRProcessBegin` when you want to begin responding to a spoken request and `SRProcessEnd` when you are finished. During that time, the feedback character displays appropriate animated gestures to indicate that it is busy performing the task. By default, the value of a recognizer's automatic feedback gestures flag is `TRUE`.

Available in Mac OS X v10.0 and later.

Declared in `SpeechRecognition.h`.

`kSRSoundInVolume`

The sound input volume. The value of this property is a 2-byte unsigned integer between 0 and 100, inclusive, that indicates the current sound input volume. This property is read-only; you cannot set a property of this type.

Available in Mac OS X v10.0 and later.

Declared in `SpeechRecognition.h`.

`kSRReadAudioFSSpec`

The audio file property. You can use this property to perform speech recognition from an audio file. The value of this property is a pointer to a file system specification (a structure of type `FSSpec`). The file system specification indicates an AIFF file that contains raw audio data (16-bit audio data sampled at 22.050 kHz). After you create a new recognizer using the speech source ID `kSRCanned22kHzSpeechSource`, you must set this recognizer property to perform recognition from an audio file. Setting the audio source to a file also allows the Speech Recognition Manager to process sound data at system background time rather than at interrupt time or deferred task time.

Available in Mac OS X v10.0 and later.

Declared in `SpeechRecognition.h`.

`kSRCancelOnSoundOut`

The cancel during sound output flag. The value of this property is a Boolean value that indicates whether speech recognition is canceled whenever any sound is output by the computer during an utterance (`TRUE`) or whether speech recognition continues (`FALSE`). By default, the value of a recognizer's cancel during sound output flag is `TRUE`.

Available in Mac OS X v10.0 and later.

Declared in `SpeechRecognition.h`.

`kSRSpeedVsAccuracyParam`

Available in Mac OS X v10.0 and later.

Declared in `SpeechRecognition.h`.

Discussion

Every recognizer has a set of properties that you can inspect and change by calling the `SRGetProperty` and `SRSetProperty` functions. You specify a property by passing a property selector to those functions.

Search Status Flags

Indicate the status of a recognizer search.

```
enum {
    kSRIdleRecognizer = 1L << 0,
    kSRSearchInProgress = 1L << 1,
    kSRSearchWaitForAllClients = 1L << 2,
    kSRMustCancelSearch = 1L << 3,
    kSRPendingSearch = 1L << 4
};
```

Constants

`kSRIdleRecognizer`
If this bit is set, the search engine is not active and the user is able to make a new utterance.
Available in Mac OS X v10.0 and later.
Declared in `SpeechRecognition.h`.

`kSRSearchInProgress`
If this bit is set, a search is currently in progress.
Available in Mac OS X v10.0 and later.
Declared in `SpeechRecognition.h`.

`kSRSearchWaitForAllClients`
If this bit is set, a search is not currently in progress, but will begin as soon as every recognizer using the speech source used by this recognizer has called `SRContinueRecognition` to indicate that the search should begin.
Available in Mac OS X v10.0 and later.
Declared in `SpeechRecognition.h`.

`kSRMustCancelSearch`
If this bit is set, a search is about to be canceled (for example, because the recognizer determined a sound to be non-speech).
Available in Mac OS X v10.0 and later.
Declared in `SpeechRecognition.h`.

`kSRPendingSearch`
If this bit is set, a search is about to begin.
Available in Mac OS X v10.0 and later.
Declared in `SpeechRecognition.h`.

Discussion

You can determine the current status of a recognizer search by getting the recognizer's search status, which is a property of type `kSRSearchStatusParam`. That property's value is a 4-byte unsigned integer. The Speech Recognition Manager defines these masks for bits in that value.

Speech Source Constants

Identify Speech Recognition Manager-supported speech sources.

```
enum {
    kSRDefaultSpeechSource = 0,
    kSRLiveDesktopSpeechSource = 'dklv',
    kSRCanned22kHzSpeechSource = 'ca22'
};
```

Constants

`kSRDefaultSpeechSource`
The default speech source.

Available in Mac OS X v10.0 and later.

Declared in `SpeechRecognition.h`.

`kSRLiveDesktopSpeechSource`
Live desktop sound input.

Available in Mac OS X v10.0 and later.

Declared in `SpeechRecognition.h`.

`kSRCanned22kHzSpeechSource`
AIFF file based 16 bit, 22.050 KHz sound input.

Available in Mac OS X v10.0 and later.

Declared in `SpeechRecognition.h`.

Discussion

The Speech Recognition Manager supports several speech sources, which you can specify using these constants. In version 1.5, the default speech source is `kSRLiveDesktopSpeechSource`.

Result Codes

The most common result codes returned by Speech Recognition Manager are listed below.

Result Code	Value	Description
`kSRNotAvailable`	-5100	Requested service not available or applicable Available in Mac OS X v10.0 and later.
`kSRInternalError`	-5101	Internal system or hardware error condition Available in Mac OS X v10.0 and later.
`kSRComponentNotFound`	-5102	Required component cannot be located Available in Mac OS X v10.0 and later.
`kSROutOfMemory`	-5103	Not enough memory available Available in Mac OS X v10.0 and later.
`kSRNotASpeechObject`	-5104	Object is not valid Available in Mac OS X v10.0 and later.
`kSRBadParameter`	-5105	Invalid parameter specified Available in Mac OS X v10.0 and later.

Result Code	Value	Description
kSRParamOutOfRange	-5106	Parameter is out of valid range Available in Mac OS X v10.0 and later.
kSRBadSelector	-5107	Unrecognized selector specified Available in Mac OS X v10.0 and later.
kSRBufferTooSmall	-5108	Buffer is too small Available in Mac OS X v10.0 and later.
kSRNotARecSystem	-5109	Specified object is not a recognition system Available in Mac OS X v10.0 and later.
kSRFeedbackNotAvail	-5110	No feedback window associated with recognizer Available in Mac OS X v10.0 and later.
kSRCantSetProperty	-5111	Cannot set the specified property Available in Mac OS X v10.0 and later.
kSRCantGetProperty	-5112	Cannot get the specified property Available in Mac OS X v10.0 and later.
kSRCantSetDuringRecognition	-5113	Cannot set property during recognition Available in Mac OS X v10.0 and later.
kSRAlreadyListening	-5114	System is already listening Available in Mac OS X v10.0 and later.
kSRNotListeningState	-5115	System is not listening Available in Mac OS X v10.0 and later.
kSRModelMismatch	-5116	No acoustical models available to match request Available in Mac OS X v10.0 and later.
kSRNoClientLanguageModel	-5117	Cannot access specified language model Available in Mac OS X v10.0 and later.
kSRNoPendingUtterances	-5118	No utterances to search Available in Mac OS X v10.0 and later.
kSRRecognitionCanceled	-5119	Search was canceled Available in Mac OS X v10.0 and later.
kSRRecognitionDone	-5120	Search has finished, but nothing was recognized Available in Mac OS X v10.0 and later.

Result Code	Value	Description
kSROtherRecAlreadyModal	-5121	Another recognizer is already operating modally Available in Mac OS X v10.0 and later.
kSRHasNoSubItems	-5122	Specified object has no subitems Available in Mac OS X v10.0 and later.
kSRSubItemNotFound	-5123	Specified subitem cannot be located Available in Mac OS X v10.0 and later.
kSRLanguageModelTooBig	-5124	Language model too big to be built Available in Mac OS X v10.0 and later.
kSRAlreadyReleased	-5125	Specified object has already been released Available in Mac OS X v10.0 and later.
kSRAlreadyFinished	-5126	Specified language model has already been finished Available in Mac OS X v10.0 and later.
kSRWordNotFound	-5127	Spelling could not be found Available in Mac OS X v10.0 and later.
kSRNotFinishedWithRejection	-5128	Language model not finished with rejection Available in Mac OS X v10.0 and later.
kSRExpansionTooDeep	-5129	Language model is left recursive or is embedded too many levels Available in Mac OS X v10.0 and later.
kSRTooManyElements	-5130	Too many elements added to phrase, path, or other language object Available in Mac OS X v10.0 and later.
kSRCantAdd	-5131	Can't add specified type of object to the base language object Available in Mac OS X v10.0 and later.
kSRSndInSourceDisconnected	-5132	Sound input source is disconnected Available in Mac OS X v10.0 and later.
kSRCantReadLanguageObject	-5133	Cannot create language object from file or pointer Available in Mac OS X v10.0 and later.
kSRNotImplementedYet	-5199	Feature is not yet implemented Available in Mac OS X v10.0 and later.

Gestalt Constants

You can check for version and feature availability information by using the Speech Recognition Manager selectors defined in the Gestalt Manager. For more information see *Inside Mac OS X: Gestalt Manager Reference*.

Text Services Manager Reference

Framework: Carbon/Carbon.h

Declared in TextServices.h

Overview

The Text Services Manager ("TSM") provides an environment for applications to use non-application-specific text services. The Text Services Manager handles communication between client applications that request text services and the software modules, known as text service components, that provide them. The Text Services Manager exists so that these two types of programs can work together without needing to know anything about the internal structures or identities of each other.

A **client application** is any text-processing program that uses the Text Services Manager to request a service from a text service component. To accomplish this, a client application needs to make specific Text Services Manager calls during execution.

A **text service component** is a utility program that uses the Text Services Manager to provide a text service to an application. Text service components are registered components with the Component Manager. Text services can include many different types of specific text-handling tasks, including spell-checking, hyphenation, and handling the input of complex text.

The most prevalent category of text services are those that handle the entry of complex text, that is, input methods. A typical example of an input method is a service that converts keyboard input into text that cannot be directly entered via a keyboard. Text input in Japanese, Chinese, Korean, or Unicode usually requires an input method.

TSM introduces input modes in Mac OS X version 10.3. An **input mode** allows an input method to temporarily accept text input in a script other than the one it normally supports. An input method uses a CFDictionary to define the input modes it supports and the tag `kTextServiceInputModePropertyTag` to specify that the input method supports input modes. An application finds out what input modes are supported by an input method by calling the function `CopyTextServiceInputModeList`.

Also new in Mac OS X version 10.3 is a suite of Carbon events that allow a text service relatively direct access to a document's text content and text attributes, such as font and glyph information. To take advantage of this new functionality in TSM, all text and offsets in your application must map to and from a flattened Unicode space. Your application must also implement callback functions to handle the appropriate Carbon events.

Mac OS X version 10.4 introduces input mode palette configuration routines.

The Text Services Manager defines three separate programming interfaces:

- The first are functions implemented by the Text Services Manager and called by the application clients of text service components.

- The second are functions implemented by the Text Services Manager and called by text service components.
- The third are low-level functions implemented by text service components and called by either applications or the Text Services Manager.

Functions by Task

Applications - Facilitating User Interactions With Components

`CopyTextServiceInputModeList` (page 1579)
Obtains a copy of the set of input modes supported by a keyboard-class input method.

`TSMCopyInputMethodEnabledInputModes` (page 1606)
Obtain the array of the enabled (and visible) input modes for a component.

`TSMSelectInputMode` (page 1610)
Sets the specified input method input mode as the current input source.

`CloseTextService` (page 1578) Deprecated in Mac OS X v10.5
Closes a text service component, other than an input method, and disassociates it from the active TSM document.

`GetDefaultInputMethod` (page 1585) Deprecated in Mac OS X v10.5
Obtains the default input method text service component for a given script and language.

`GetDefaultInputMethodOfClass` (page 1586) Deprecated in Mac OS X v10.5
Obtains the default input method text service component for a given text service class.

`GetServiceList` (page 1589) Deprecated in Mac OS X v10.5
Obtains a list of the text service components of a specified type that are currently available.

`GetTextServiceLanguage` (page 1590) Deprecated in Mac OS X v10.5
Obtains the current input script and language.

`OpenTextService` (page 1597) Deprecated in Mac OS X v10.5
Opens a text service component, other than an input method, and associates it with a TSM document.

`SetDefaultInputMethod` (page 1601) Deprecated in Mac OS X v10.5
Sets a default input method to a given script and language.

`SetDefaultInputMethodOfClass` (page 1602) Deprecated in Mac OS X v10.5
Sets the default input method text service component for a given text service class.

`SetTextServiceLanguage` (page 1603) Deprecated in Mac OS X v10.5
Changes the current input script and language.

Applications - Managing TSM Documents

`NewTSMDocument` (page 1595)
Creates a TSM document and returns a handle to the document's ID.

`DeleteTSMDocument` (page 1582)
Closes all opened text service components for the TSM document.

`ActivateTSMDocument` (page 1577)
Informs the Text Services Manager that a TSM document is active.

`DeactivateTSMDocument` (page 1581)
Informs the Text Services Manager that a TSM document is inactive.

`FixTSMDocument` (page 1584)
Informs the Text Services Manager that user input for a TSM document has been interrupted.

`UseInputWindow` (page 1613)
Associates a floating input window with one or more TSM documents.

`TSMGetActiveDocument` (page 1607)
Obtains the active TSM document in the current application context.

`TSMSetInlineInputRegion` (page 1611) Deprecated in Mac OS X v10.5
Defines a region within a TSM document in which inline input can occur.

Components - Sending Events

`SendTextInputEvent` (page 1600)
Sends Carbon text input events from a text service component to a client application.

`SendAEFromTSMComponent` (page 1598) Deprecated in Mac OS X v10.5
Sends Apple events from a text service component to a client application.

Low Level - Accessing Text Service Properties

`SetTextServiceProperty` (page 1603)
Notifies a text service component that one of its properties has been selected.

`GetTextServiceProperty` (page 1591)
Notifies a text service component that it must identify the current value of one of its properties.

Low Level - Confirming Text Service Input

`FixTextService` (page 1583)
Notifies a text service component that it must complete the processing of any input that is in progress.

Low Level - Managing Text Service States

`InitiateTextService` (page 1593)
Notifies a text service component that it must perform any necessary set-up tasks and begin operating.

`ActivateTextService` (page 1577)
Notifies a text service component that its associated document window is becoming active.

`DeactivateTextService` (page 1580)
Notifies a text service component that its associated document window is becoming inactive.

`TerminateTextService` (page 1604)
Notifies a text service component that it must terminate its operations in preparation for closing.

`HidePaletteWindows` (page 1592)
Notifies a text service component that it must hide its floating windows.

`DeselectTextService` (page 1583) Deprecated in Mac OS X v10.5
Notifies TSM that an input method has been closed.

`IsTextServiceSelected` (page 1595) Deprecated in Mac OS X v10.5
Determines if a text service component is selected.

`SelectTextService` (page 1598) Deprecated in Mac OS X v10.5
Selects a text service.

Low Level - Querying Text Services

`GetTextServiceMenu` (page 1590)
Notifies a text service component that it must produce a handle to its menu.

`GetScriptLanguageSupport` (page 1588) Deprecated in Mac OS X v10.5
Notifies a text service component that it must produce a list of its supported languages and scripts.

Low Level - Sending Events to Text Services

`TextServiceEventRef` (page 1605)
Provides an opportunity for a text service component to handle a Carbon event.

Working With Document Properties

`TSMSetDocumentProperty` (page 1611)
Sets a property for a TSM document.

`TSMGetDocumentProperty` (page 1607)
Obtains a TSM document property.

`TSMRemoveDocumentProperty` (page 1610)
Removes a property from a TSM document.

Input Mode Palette Configuration

`InputModePaletteItemHit` (page 1594)
Informs an input method that a function button on the input mode palette was pressed.

`GetInputModePaletteMenu` (page 1587) Deprecated in Mac OS X v10.5
Obtains from an input method the menu to display for a pull-down menu on the input mode palette.

`TSMInputModePaletteLoadButtons` (page 1609) Deprecated in Mac OS X v10.5
Notifies the input mode palette of changes to the controls for an input method and replaces the current controls with the new control array.

`TSMInputModePaletteUpdateButtons` (page 1609) Deprecated in Mac OS X v10.5
Notifies the input mode palette of changes to the controls for an input method and updates the controls.

Functions

ActivateTextService

Notifies a text service component that its associated document window is becoming active.

```
ComponentResult ActivateTextService (
   ComponentInstance ts
);
```

Parameters

ts

A Component Manager value of type `ComponentInstance` that identifies the component being called. When the Text Services Manager makes this call, it passes the `ComponentInstance` value returned by its call to the `OpenComponent` function. If an application makes this call, it may use the `ComponentInstance` value obtained from the `kEventParamTextInputSendComponentInstance` parameter of the Carbon event or the `keyAEServerInstance` parameter of an Apple event sent by the component being called. Alternately, an application may obtain a `ComponentInstance` value from a prior call to the function `OpenTextService` (page 1597).

Return Value

See the Component Manager documentation for a description of the `ComponentResult` data type.

Discussion

Text service components must implement a function for this call.

The appropriate response to `ActivateTextService` is for the text service component to restore its active state, including displaying all floating windows if they have been hidden. (Note that typically an input-method component should not hide its windows in response to being deactivated. If the subsequent document being activated is using the same component's service, it would be irritating to the user to hide and then immediately redisplay the same windows. An input-method component should hide its windows only in response to a `HidePaletteWindows` call.) If the component is an input method, it should specify the redisplay of any unconfirmed text currently in the active input area.

The Text Services Manager makes this call either on its own or in response to application-interface calls it receives from client applications. Client applications may directly make this call, but the Text Services Manager does not then play a role in the connection between the client application making the call and the text service component receiving it.

Availability

Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.

Available in Mac OS X v 10.0 and later.

Not available to 64-bit applications.

Declared In

`TextServices.h`

ActivateTSMDocument

Informs the Text Services Manager that a TSM document is active.

```
OSErr ActivateTSMDocument (
   TSMDocumentID idocID
);
```

Parameters

idocID

A TSM document identification number created by a prior call to the `NewTSMDocument` (page 1595) function.

Return Value

A result code. See "Text Services Manager Result Codes" (page 1647).

Discussion

When a window that has an associated TSM document becomes active, your client application must call the `ActivateTSMDocument` function to inform the Text Services Manager that the document is activated and is ready to use text service components.

`ActivateTSMDocument` calls the equivalent text service component function `ActivateTextService` (page 1577) for all open text service components associated with the TSM document.

If a text service component has a menu, the Text Services Manager inserts the menu into the menu bar.

Availability

Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.

Available in Mac OS X v 10.0 and later.

Not available to 64-bit applications.

Declared In

`TextServices.h`

CloseTextService

Closes a text service component, other than an input method, and disassociates it from the active TSM document. (Deprecated in Mac OS X v10.5.)

```
OSErr CloseTextService (
   TSMDocumentID idocID,
   ComponentInstance aComponentInstance
);
```

Parameters

idocID

The identification number of a TSM document created by a prior call to the `NewTSMDocument` (page 1595) function.

aComponentInstance

The component instance created by a prior call to `OpenTextService` (page 1597).

Return Value

A result code. See "Text Services Manager Result Codes" (page 1647).

Discussion

When a user wants to close an opened text service component, your client application should call the function `CloseTextService`.

If the text service component displays a menu, the Text Services Manager removes the menu from the menu bar. This function is for closing text service components other than input methods. Your application does not need to open or close input methods.

Availability
Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.
Available in Mac OS X v 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`TextServices.h`

CopyTextServiceInputModeList

Obtains a copy of the set of input modes supported by a keyboard-class input method.

```
ComponentResult CopyTextServiceInputModeList (
   ComponentInstance ts,
   CFDictionaryRef *outInputModes
);
```

Parameters

ts

The component whose set of input modes you want to obtain.

outInputModes

On output, the CFDictionary contains the list of supported input modes. See the Discussion for more information on the structure and requirements of the dictionary.

Return Value
See the Component Manager documentation for a description of the `ComponentResult` data type.

Discussion
This function is supported by input methods that adopt the input mode protocol. If this component call is not supported by an input method, calls to functions that access text service properties (using the tag `kTextServiceInputModePropertyTag`) return the result `tsmComponentPropertyUnsupportedErr`.

The CFDictionary of input modes (available in TSM 2.2 and later) must have the following form:

```
<dict>
<key> kTSInputModeListKey </key>
<dict>
<key> modeSignature : (internal ascii name)</key>
<!-- This can be any of the generic input modes defined in this  file,-->
<!-- such as kTextServiceInputModeRoman,or can be a private input-->
<!-- mode such as CFSTR("com.apple.MyInputmethod.Japanese.CoolMode")  -->
 <dict>
<key>menuIconFile</key>
<string> (path for menu icon image file)</string>
<key>alternateMenuIconFile</key>
<string> (path for alternate menu icon image file, when item  is hilited)</string>
<key>paletteIconFile</key>
<string> (path for palette icon image file) </string>
<key>defaultState</key>
<boolean> (default on/off state) </boolean>
```

```
<key>script</key>
<string> (scriptCode string for this mode, for example, "smRoman")  </string>
<key>primaryInScript</key>
<boolean> (true if this is primary mode in this script) </boolean>
<key>isVisible</key>
<boolean> (true if this input mode should appear in System  UI) </boolean>
<key>keyEquivalentModifiers</key>
<integer> (modifiers)</integer>
<key>keyEquivalent</key>
<string> (key equivalent character) </string>
<key>JISKeyboardShortcut</key>
<integer> (optional: 0=none,1=hiragana, 2=katakana, 3=eisu)  </integer>
</dict>
</dict>
</dict>
```

This dictionary must also be present in the Info.plist for the component bundle, in addition to being available through this component call. Availability in the Info.plist allows retrieval of input modes by the system without opening the component. The component call is used whenever the system is notified of a change in the contents of the input mode list, such as when the name or key-equivalents of individual input modes have changed.

If, when the input method is first activated in a login session, the settings of the individual input modes (names or key-equivalents) differ from the default settings as found in the component bundle Info.plist, the system needs to be notified of the change. The input method does this by sending out the Carbon event `kEventTextInputInputMenuChanged`, just as when the change originally took place.

For more information on the dictionary keys used to define input modes and the input mode dictionary, see "Input Mode Dictionary Key" (page 1633) and "Individual Input Mode Keys" (page 1635).

Availability
Not available in CarbonLib.
Available in Mac OS X v 10.3 and later.
Not available to 64-bit applications.

Declared In
`TextServices.h`

DeactivateTextService

Notifies a text service component that its associated document window is becoming inactive.

```
ComponentResult DeactivateTextService (
   ComponentInstance ts
);
```

Parameters

ts

A Component Manager value of type `ComponentInstance` that identifies the component being called. When the Text Services Manager makes this call, it passes the `ComponentInstance` value returned by its call to the `OpenComponent` function. If an application makes this call, it may use the `ComponentInstance` value obtained from the `kEventParamTextInputSendComponentInstance` parameter of the Carbon event or the `keyAEServerInstance` parameter of an Apple event sent by the component being called. Alternately, an application may obtain a `ComponentInstance` value from a prior call to the function `OpenTextService` (page 1597).

Return Value
See the Component Manager documentation for a description of the `ComponentResult` data type.

Discussion
Text service components must implement a function for this call.

When it receives a `DeactivateTextService` call, the text service component is responsible for saving whatever state information it needs to save, so that it can restore the proper information when it becomes active again. Note that an input method should not confirm any unconfirmed text in the active input area, but should save it until reactivated.

A component other than an input method should hide all its floating windows and menus. However, an input-method component should not hide its windows in response to this call. If the subsequent document being activated is using the same component's service, it would be irritating to the user to hide and then immediately redisplay the same windows. An input-method component should hide its windows only in response to a `HidePaletteWindows` call.

The Text Services Manager makes this call either on its own or in response to application-interface calls it receives from client applications. Client applications may directly make this call, but the Text Services Manager does not then play a role in the connection between the client application making the call and the text service component receiving it.

Availability
Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.
Available in Mac OS X v 10.0 and later.
Not available to 64-bit applications.

Declared In
`TextServices.h`

DeactivateTSMDocument

Informs the Text Services Manager that a TSM document is inactive.

```
OSErr DeactivateTSMDocument (
   TSMDocumentID idocID
);
```

Parameters

idocID

A TSM document identification number created by a prior call to the `NewTSMDocument` (page 1595) function.

Return Value

A result code. See "Text Services Manager Result Codes" (page 1647).

Discussion

The `DeactivateTSMDocument` function lets you inform the Text Services Manager that a TSM document in your client application is no longer active and must temporarily stop using text service components.

The Text Services Manager calls the equivalent text service component function `DeactivateTextService` (page 1580) for any text service component associated with the TSM document being deactivated.

An application that supports inline input should always strive to have a TSM document active at all times. If a situation arises in which all TSM documents are inactive and keyboard input occurs, the Text Services Manager automatically interacts with the user via its floating input window. (This is the same floating window that the Text Services Manager displays if an application calls the function `UseInputWindow` (page 1613) with a value of `TRUE` for the `useWindow` parameter.)

Availability

Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.

Available in Mac OS X v 10.0 and later.

Not available to 64-bit applications.

Declared In

`TextServices.h`

DeleteTSMDocument

Closes all opened text service components for the TSM document.

```
OSErr DeleteTSMDocument (
   TSMDocumentID idocID
);
```

Parameters

idocID

A TSM document identification number created by a prior call to the `NewTSMDocument` (page 1595) function.

Return Value

A result code. See "Text Services Manager Result Codes" (page 1647).

Discussion
When your application disposes of a TSM document, it must call the `DeleteTSMDocument` function to inform the Text Services Manager that the document is no longer using text service components. `DeleteTSMDocument` invokes the Component Manager `CloseComponent` function for each open text service component associated with this document. It also disposes of the internal data structure for the TSM document.

Availability
Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.
Available in Mac OS X v 10.0 and later.
Not available to 64-bit applications.

Declared In
`TextServices.h`

DeselectTextService

Notifies TSM that an input method has been closed. (Deprecated in Mac OS X v10.5.)

```
OSStatus DeselectTextService (
   Component aComp
);
```

Parameters
aComp

Return Value
A result code. See "Text Services Manager Result Codes" (page 1647).

Discussion
This API is currently only intended for use by Character Palette class input methods. It allows such an input method to notify TSM that it has been closed by the user as a result of interaction with the input method's own UI, such a palette's close button, instead of via the normal UI provided by the System, such as the Keyboard Menu.

Availability
Not available in CarbonLib.
Available in Mac OS X v 10.2 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`TextServices.h`

FixTextService

Notifies a text service component that it must complete the processing of any input that is in progress.

```
ComponentResult FixTextService (
   ComponentInstance ts
);
```

Parameters

ts

A Component Manager value of type `ComponentInstance` that identifies the component being called. When the Text Services Manager makes this call, it passes the `ComponentInstance` value returned by its call to the `OpenComponent` function. If an application makes this call, it may use the `ComponentInstance` value obtained from the `kEventParamTextInputSendComponentInstance` parameter of the Carbon event or the `keyAEServerInstance` parameter of an Apple event sent by the component being called. Alternately, an application may obtain a `ComponentInstance` value from a prior call to the function `OpenTextService` (page 1597).

Return Value
See the Component Manager documentation for a description of the `ComponentResult` data type.

Discussion
Text service components must implement a function for this call.

For input method components, this function is equivalent to the user explicitly confirming text, but in this case the request comes instead from the application or from the Text Services Manager. Typically, users confirm text explicitly (such as by pressing the Return key), and input methods continually process these user events and send the confirmed text to client applications. Circumstances may arise, however, in which an application needs the input method to confirm and send input without an explicit confirmation from the user.

If, for example, the user clicks the mouse in text outside the active input area, that constitutes an implicit user acceptance of the text in the active input area. In this case, applications should explicitly terminate any active input by calling the `FixTSMDocument` (page 1584) function, which notifies the Text Services Manager. The Text Services Manager then calls the `FixTextService` function, which notifies the text service component that it must stop accepting further input and pass the current contents (both converted and unconverted) of the active input area as confirmed text to the client application.

Client applications may directly make this call, but the Text Services Manager does not then play a role in the connection between the client application making the call and the text service component receiving it.

Availability
Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.
Available in Mac OS X v 10.0 and later.
Not available to 64-bit applications.

Declared In
`TextServices.h`

FixTSMDocument

Informs the Text Services Manager that user input for a TSM document has been interrupted.

```
OSErr FixTSMDocument (
   TSMDocumentID idocID
);
```

Parameters

idocID

The identification number of a TSM document created by a prior call to the `NewTSMDocument` (page 1595) function.

Return Value

A result code. See "Text Services Manager Result Codes" (page 1647).

Discussion

Typically, an inline input text service component removes confirmed input from the active input area each time the user presses the Return key, and passes the confirmed text to your application through a Carbon event or an Apple event.

In certain situations, however, your client application may need to inform the text service component that input in the active input area of a specified TSM document has been interrupted, and that the text service component must confirm the text and terminate user input.In this case you call the `FixTSMDocument` function to give the input method text service component the opportunity to confirm any input in progress.

For instance, if the user clicks in the close box of the window in which active input is taking place, call `FixTSMDocument` before you close the window. The text service component will pass you the current contents (both converted and unconverted) of the active input area as confirmed text.

For simple activating and deactivating of your application's window, it is not necessary to confirm the text in the active inline area. The input method saves the text and restores it when your window is reactivated.

Availability

Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.

Available in Mac OS X v 10.0 and later.

Not available to 64-bit applications.

Declared In

`TextServices.h`

GetDefaultInputMethod

Obtains the default input method text service component for a given script and language. (Deprecated in Mac OS X v10.5.)

Not recommended.

```
OSErr GetDefaultInputMethod (
   Component *ts,
   ScriptLanguageRecord *slRecordPtr
);
```

Parameters

ts

A pointer to the component identifier of the input method text service component that is associated with the script and language combination given in the `slRecord` parameter.

slRecordPtr

A pointer to a structure of type `ScriptLanguageRecord` (page 1614). This structure describes the script and language combination that is associated with the input method text service specified in the `ts` parameter.

Return Value
A result code. See "Text Services Manager Result Codes" (page 1647).

Discussion
You should use the function `GetDefaultInputMethodOfClass` (page 1586) instead of this one.

The operating system uses `GetDefaultInputMethod` to find out which input method to activate when the user selects a new keyboard script from the Keyboard menu or by Command-key combination, or when an application calls `KeyScript` to change keyboard scripts. Your application should not typically need to call this function.

Version Notes
For systems prior to Mac OS X, in versions of Japanese system software starting with KanjiTalk 7, if the default input method is pre-KanjiTalk 7 and non-TSM-aware, `GetDefaultInputMethod` returns the error `tsmInputMethodIsOldErr`. In that case the `ts` parameter contains the script code of the old input method in its high-order word, and the reference ID of the old input method in its low-order word.

Availability
Not recommended. Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.
Available in Mac OS X v 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`TextServices.h`

GetDefaultInputMethodOfClass

Obtains the default input method text service component for a given text service class. (Deprecated in Mac OS X v10.5.)

```
OSStatus GetDefaultInputMethodOfClass (
   Component *aComp,
   ScriptLanguageRecord *slRecPtr,
   TextServiceClass tsClass
);
```

Parameters

aComp

On return, a pointer to the component identifier of the input method text service component that is associated with the script and language combination given in the `slRecord` parameter.

slRecPtr

On return, a pointer to a structure of type `ScriptLanguageRecord` (page 1614). This structure describes the script and language combination that is associated with the input method text service specified in the `ts` parameter.

tsClass

The text service class whose component and script language record you want to obtain. Pass `kKeyboardInputMethodClass` to specify a keyboard input method. Pass `kInkInputMethod` to specify an Ink input method.

Return Value
A result code. See "Text Services Manager Result Codes" (page 1647).

Discussion

Availability
Not available in CarbonLib.
Available in Mac OS X v 10.2 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`TextServices.h`

GetInputModePaletteMenu

Obtains from an input method the menu to display for a pull-down menu on the input mode palette. (Deprecated in Mac OS X v10.5.)

```
ComponentResult GetInputModePaletteMenu (
   ComponentInstance inInstance,
   UInt32 inItemID,
   CFArrayRef *outMenuItemsArray
);
```

Parameters

inInstance

The component instance.

inItemID

The item ID of the pull-down menu button.

outMenuItemsArray

On return, points to an array of menu items. A pull-down menu consists of an array of CFDictionary objects that contain the keys described in `Input Mode Palette Menu Definition Keys` (page 1633).

Return Value
Returns a non-null value on successful handling of the call.

Availability
Available in Mac OS X v 10.4 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`TextServices.h`

GetScriptLanguageSupport

Notifies a text service component that it must produce a list of its supported languages and scripts. (Deprecated in Mac OS X v10.5.)

```
ComponentResult GetScriptLanguageSupport (
   ComponentInstance ts,
   ScriptLanguageSupportHandle *scriptHdl
);
```

Parameters

ts

A Component Manager value of type `ComponentInstance` that identifies the component being called. When the Text Services Manager makes this call, it passes the `ComponentInstance` value returned by its call to the `OpenComponent` function. If an application makes this call, it may use the `ComponentInstance` value obtained from the `kEventParamTextInputSendComponentInstance` parameter of the Carbon event or the `keyAEServerInstance` parameter of an Apple event sent by the component being called. Alternately, an application may obtain a `ComponentInstance` value from a prior call to the function `OpenTextService` (page 1597).

scriptHdl

A handle to a structure of type `ScriptLanguageSupport` (page 1615). The handle must be either `NULL` or a valid handle. If it is `NULL`, the text service component allocates a new handle. If it is already a valid handle, the text service component resizes it as necessary. `GetScriptLanguageSupport` should produce a list of scripts and languages in this parameter.

Return Value

The return value should contain 0 if the list is correct, or an error value if an error occurred. See the Component Manager documentation for a description of the `ComponentResult` data type.

Discussion

Text service components must implement a function for this call.

In response to this call, the component should list all its supported scripts and languages, starting with the primary script and language as specified in the `componentFlags` field of its component description structure. The Text Services Manager makes the `GetScriptLanguageSupport` call after a component is opened via the Component Manager function `OpenComponent`. One of the Text Services Manager's uses of this script information is to determine whether to exchange information with the component in the Unicode text encoding.

For example, if a component is associated with a Macintosh script, but includes the `smUnicodeScript` constant in its enumeration of supported constants, then the Text Services Manager determines that the component produces and expects text in the Unicode encoding. Additionally, in this example, the Text Services Manager also concludes that the Unicode characters which the component supports are limited to those encoded in the repertoire of the encoding corresponding to the Macintosh script in the `componentFlags` field of its component description structure. Note that if the script specified in the `componentFlags` field is itself `smUnicodeScript`, the Text Services Manager imposes no restriction on the set of supported characters, and it treats the component as being capable of handling any Unicode character.

Client applications may directly make this call, but the Text Services Manager does not then play a role in the connection between the client application making the call and the text service component receiving it.

Availability

Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.

Available in Mac OS X v 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
TextServices.h

GetServiceList

Obtains a list of the text service components of a specified type that are currently available. (Deprecated in Mac OS X v10.5.)

```
OSErr GetServiceList (
   SInt16 numOfInterface,
   const OSType *supportedInterfaceTypes,
   TextServiceListHandle *serviceInfo,
   SInt32 *seedValue
);
```

Parameters

numOfInterface
The number of text service interface types supported by your client application.

supportedInterfaceTypes
A pointer to a value of type InterfaceTypeList (page 1614) specifying the kinds of text services that your program supports. This list helps the Text Services Manager locate text services of the correct interface type.

serviceInfo
A pointer to a handle to a structure of type TextServiceList (page 1616). If the handle is NULL, the Text Services Manager allocates the handle; otherwise, it assumes the handle is a valid text service component list handle, as defined by the TextServiceListHandle data type.

seedValue
A pointer to a value that indicates whether the list of text service components returned by GetServiceList may have been modified.

Return Value
A result code. See "Text Services Manager Result Codes" (page 1647).

Discussion
Your client application is responsible for providing a way—usually a menu—for the user to choose from among all available text service components. To get a list of available text service components to display in a menu, call the GetServiceList function. Be sure to filter out input methods, because the Keyboard menu already displays them.

When your application calls GetServiceList, the Text Services Manager locates all the text service components of the specified types and creates a text service component list, defined by the TextServiceList data type, containing an entry for each of the text service components.

It is possible to register text service components or withdraw them from registration at any time. Once it has compiled a list of text services, the Text Services Manager invokes the GetComponentListModSeed function and returns the value in the modseed parameter. You can save that value and, the next time you need to draw or regenerate the list of services, call the Component Manager GetComponentListModSeed function. If the seed value differs from the one you received from your last call to GetServiceList, you need to call GetServiceList once more to update the information. Alternately, you can simply call GetServiceList each time you need to update the list, although that may be less efficient.

Availability
Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.
Available in Mac OS X v 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`TextServices.h`

GetTextServiceLanguage

Obtains the current input script and language. (Deprecated in Mac OS X v10.5.)

```
OSErr GetTextServiceLanguage (
   ScriptLanguageRecord *slRecordPtr
);
```

Parameters

slRecordPtr

A pointer to a structure of type `ScriptLanguageRecord` (page 1614). Upon completion of the call, this structure describes the language supported by the default (current) text service component for the current keyboard script.

Return Value
A result code. See "Text Services Manager Result Codes" (page 1647).

Discussion
Your application should not typically need to call this function.

Availability
Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.
Available in Mac OS X v 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`TextServices.h`

GetTextServiceMenu

Notifies a text service component that it must produce a handle to its menu.

```
ComponentResult GetTextServiceMenu (
   ComponentInstance ts,
   MenuRef *serviceMenu
);
```

Parameters

ts

A Component Manager value of type `ComponentInstance` that identifies the component being called. When the Text Services Manager makes this call, it passes the `ComponentInstance` value returned by its call to the `OpenComponent` function. If an application makes this call, it may use the `ComponentInstance` value obtained from the `kEventParamTextInputSendComponentInstance` parameter of the Carbon event or the `keyAEServerInstance` parameter of an Apple event sent by the component being called. Alternately, an application may obtain a `ComponentInstance` value from a prior call to the function `OpenTextService` (page 1597).

serviceMenu

A pointer to a menu handle (defined by the Menu Manager `MenuHandle` data type) for the text service component that is to be updated. The menu handle may be preallocated or it may be `NULL`. If the menu handle is `NULL`, the text service component should allocate a new menu and return it. On Mac OS 8 and 9, note that all instances of a component must share a single menu handle, allocated in the system heap. On Mac OS X, all instances of a component must share a single menu handle within an application's context.

Return Value

If the text service component does not have a menu, it should return a `ComponentResult` value of `TSMHasNoMenuErr`. See the Component Manager documentation for a description of the `ComponentResult` data type.

Discussion

Text service components must implement a function for this call.

The Text Services Manager calls `GetTextServiceMenu` when a component is opened or activated, so that it can put the component's menu on the menu bar.

Client applications may directly make this call, but the Text Services Manager does not then play a role in the connection between the client application making the call and the text service component receiving it.

Availability

Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.

Available in Mac OS X v 10.0 and later.

Not available to 64-bit applications.

Declared In

`TextServices.h`

GetTextServiceProperty

Notifies a text service component that it must identify the current value of one of its properties.

```
ComponentResult GetTextServiceProperty (
   ComponentInstance ts,
   TextServicePropertyTag inPropertyTag,
   TextServicePropertyValue *outPropertyValue
);
```

Parameters

ts

A Component Manager value of type `ComponentInstance` that identifies the component being called. When the Text Services Manager makes this call, it passes the `ComponentInstance` value returned by its call to the `OpenComponent` function. If an application makes this call, it may use the `ComponentInstance` value obtained from the `kEventParamTextInputSendComponentInstance` parameter of the Carbon event or the `keyAEServerInstance` parameter of an Apple event sent by the component being called. Alternately, an application may obtain a `ComponentInstance` value from a prior call to the function `OpenTextService` (page 1597).

propertySelector

A constant identifying a general property of a text service. For descriptions of the system-defined property selectors, see "Text Service Properties" (page 1640).

result

On return, a constant specifying the value for a text service property. For descriptions of the system-defined property values, see "Text Service Properties" (page 1640).

Return Value

See the Component Manager documentation for a description of the `ComponentResult` data type.

Discussion

Text service components have the option of implementing a function for this call.

Both the Text Services Manager and client applications can call this function to manage text service properties. If client applications directly make this call, the Text Services Manager does not then play a role in the connection between the client application making the call and the text service component receiving it.

Availability

Available in CarbonLib 1.0 and later.

Available in Mac OS X v 10.0 and later.

Not available to 64-bit applications.

Declared In

`TextServices.h`

HidePaletteWindows

Notifies a text service component that it must hide its floating windows.

```
ComponentResult HidePaletteWindows (
   ComponentInstance ts
);
```

Parameters

ts

A Component Manager value of type `ComponentInstance` that identifies the component being called. When the Text Services Manager makes this call, it passes the `ComponentInstance` value returned by its call to the `OpenComponent` function. If an application makes this call, it may use the `ComponentInstance` value obtained from the `kEventParamTextInputSendComponentInstance` parameter of the Carbon event or the `keyAEServerInstance` parameter of an Apple event sent by the component being called. Alternately, an application may obtain a `ComponentInstance` value from a prior call to the function `OpenTextService` (page 1597).

Return Value

If the text service component has no palettes, it should return a `ComponentResult` value of `noErr`. See the Component Manager documentation for a description of the `ComponentResult` data type.

Discussion

Text service components must implement a function for this call.

If a window associated with a TSM document associated with your text service is being deactivated, your text service component receives the `DeactivateTextService` (page 1580) call. You should perform any necessary cleanup or other tasks associated with deactivating your current component instance. If your text service component is not an input method, you should also hide all floating windows associated with the document being deactivated. If your text service component is an input method and if the newly activated document does not use your text services, you receive the `HidePaletteWindows` call. When it receives a `HidePaletteWindows` call, your input method should hide all its floating and nonfloating windows associated with the component instance being deactivated. Its menus, if any, will be removed from the menu bar by the Text Services Manager.

The Text Services Manager makes this call either on its own or in response to application-interface calls it receives from client applications. Client applications may directly make this call, but the Text Services Manager does not then play a role in the connection between the client application making the call and the text service component receiving it.

Availability

Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.

Available in Mac OS X v 10.0 and later.

Not available to 64-bit applications.

Declared In

`TextServices.h`

InitiateTextService

Notifies a text service component that it must perform any necessary set-up tasks and begin operating.

```
ComponentResult InitiateTextService (
   ComponentInstance ts
);
```

Parameters

ts

A Component Manager value of type `ComponentInstance` that identifies the component being called. When the Text Services Manager makes this call, it passes the `ComponentInstance` value returned by its call to the `OpenComponent` function. If an application makes this call, it may use the `ComponentInstance` value obtained from the `kEventParamTextInputSendComponentInstance` parameter of the Carbon event or the `keyAEServerInstance` parameter of an Apple event sent by the component being called. Alternately, an application may obtain a `ComponentInstance` value from a prior call to the function `OpenTextService` (page 1597).

Return Value

This function should return a `ComponentResult` value of zero if there is no error, and an error code if there is one. See the Component Manager documentation for a description of the `ComponentResult` data type.

Discussion

Text service components must implement a function for this call.

The Text Services Manager can call `InitiateTextService` to any component that it has already opened with the Component Manager `OpenComponent` or `OpenDefaultComponent` functions. Text service components should be prepared to handle `InitiateTextService` calls at any time.

Any text service component can receive multiple `InitiateTextService` calls. The Text Services Manager calls `InitiateTextService` each time the user adds a text service to a TSM document, even if the text service component has already been opened. This provides an opportunity for the component to restart or to display user interface elements that the user may have closed.

The Text Services Manager makes this call either on its own or in response to application-interface calls it receives from client applications. Client applications may directly make this call, but the Text Services Manager does not then play a role in the connection between the client application making the call and the text service component receiving it.

Availability

Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.

Available in Mac OS X v 10.0 and later.

Not available to 64-bit applications.

Declared In

`TextServices.h`

InputModePaletteItemHit

Informs an input method that a function button on the input mode palette was pressed. (Deprecated in Mac OS X v10.5.)

```
ComponentResult InputModePaletteItemHit (
   ComponentInstance inInstance,
   UInt32 inItemID,
   UInt32 inItemState
);
```

Parameters

inInstance

The component instance.

inItemID

The item ID of the function button pressed on the palette.

inItemState

The new state of the button.

Return Value

Returns a non-null value on successful handling of the call.

Availability

Available in Mac OS X v 10.4 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`TextServices.h`

IsTextServiceSelected

Determines if a text service component is selected. (Deprecated in Mac OS X v10.5.)

```
Boolean IsTextServiceSelected (
   Component aComp
);
```

Parameters

aComp

The component you want to determine is selected or not.

Return Value

Returns `true` if the component is selected.

Availability

Not available in CarbonLib.

Available in Mac OS X v 10.3 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`TextServices.h`

NewTSMDocument

Creates a TSM document and returns a handle to the document's ID.

```
OSErr NewTSMDocument (
   SInt16 numOfInterface,
   InterfaceTypeList supportedInterfaceTypes,
   TSMDocumentID *idocID,
   SRefCon refcon
);
```

Parameters

numOfInterface

The number of text service interface types that your application supports.

supportedInterfaceTypes

A value of type `InterfaceTypeList` (page 1614) specifying the kinds of text services that your program supports. This list helps the Text Services Manager locate the text services that have the correct interface type.

idocID

Upon successful completion of the call, a pointer to the document identification number of the TSM document created. If `NewTSMDocument` fails to create a new TSM document, it returns an error and sets `idocID` to `NULL`.

refcon

A reference constant to be associated with the TSM document. This may have any value you wish.

Return Value

A result code. See "Text Services Manager Result Codes" (page 1647).

Discussion

Each time your client application calls the `NewTSMDocument` function, the Text Services Manager creates an internal structure called a TSM document and returns its ID.

If the call is successful, `NewTSMDocument` opens the default input method text service component of the current keyboard script and assigns it to this document. If `NewTSMDocument` returns `tsmScriptHasNoIMErr`, it has still created a valid TSM document, but has not associated an input method with it.

Starting in Mac OS X v10.3, the `NewTSMDocument` function turns on the `kTSMDocumentUnicodeInputWindowPropertyTag` implicitly for TSMDocuments of interface type `kUnicodeDocumentInterfaceType`. The effect is that Unicode input sources (keyboard layouts) can remain available, not only in an editing mode where Unicode is supported by an application, but also outside of editing mode where no TSMDocument in particular is active.

This change also provides compatibility with many applications that relied on Unicode input being available even without activating any of their own TSMDocuments, as well as other applications that do create their own Unicode TSMDocument but call the function `UseInputWindow` passing `true` (which was really an undefined operation in the original Unicode/TSM specification).

Availability

Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.

Available in Mac OS X v 10.0 and later.

Not available to 64-bit applications.

Declared In

`TextServices.h`

OpenTextService

Opens a text service component, other than an input method, and associates it with a TSM document. (Deprecated in Mac OS X v10.5.)

```
OSErr OpenTextService (
   TSMDocumentID idocID,
   Component aComponent,
   ComponentInstance *aComponentInstance
);
```

Parameters

idocID

The identification number of a TSM document created by a prior call to the `NewTSMDocument` (page 1595) function.

aComponent

A component identifier for this text service component. You can obtain the component identifier to pass in `aComponent` by comparing the menu item name selected by the user with the component item name found in a `TextServiceInfo` (page 1615) structure. You can obtain a `TextServiceInfo` structure by calling the function `GetServiceList` (page 1589) and examining the `fServices` field of the `TextServiceList` structure that it produces.

aComponentInstance

Upon completion of the call, a pointer to a component instance. This value identifies your application's connection to a text service component. You must supply this value if you call the text service functions provided by the component directly.

Return Value

A result code. See "Text Services Manager Result Codes" (page 1647).

Discussion

This function instructs the Text Services Manager to open a text service component, other than an input method, that a user has chosen and to associate it with a TSM document. The Text Services Manager opens the requested component by calling the Component Manager `OpenComponent` function.

If the specified text service component is already open, the Text Services Manager does not open it again and the `tsmComponentAlreadyOpenErr` error message is returned as a result code. Whether or not the text service is open, the Text Services Manager calls the functions `InitiateTextService` (page 1593) and `ActivateTextService` (page 1577) for the given text service and returns a valid component instance. Upon completion of the `OpenTextService` call, the selected text service component is initialized and active.

This function is for opening text service components other than input methods. Your application does not need to open or close input methods.

Availability

Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.

Available in Mac OS X v 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`TextServices.h`

SelectTextService

Selects a text service. (Deprecated in Mac OS X v10.5.)

```
OSStatus SelectTextService (
   Component aComp
);
```

Parameters

aComp

The text service you want to select.

Return Value

A result code. See "Text Services Manager Result Codes" (page 1647).

Discussion

This function is intended for use by input methods in text service classes which are additive in nature, that is where the input method can operate in parallel to other input methods in the same class and other additive text service classes. An example of such a class is the Character Palette class. This function is not for use by traditional input methods, such as those that belong to the keyboard input method class.

Availability

Not available in CarbonLib.

Available in Mac OS X v 10.2 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`TextServices.h`

SendAEFromTSMComponent

Sends Apple events from a text service component to a client application. (Deprecated in Mac OS X v10.5.)

```
OSErr SendAEFromTSMComponent (
   const AppleEvent *theAppleEvent,
   AppleEvent *reply,
   AESendMode sendMode,
   AESendPriority sendPriority,
   SInt32 timeOutInTicks,
   AEIdleUPP idleProc,
   AEFilterUPP filterProc
);
```

Parameters

theAppleEvent

A pointer to the Apple event to be sent.

reply

A pointer to the reply Apple event returned by `SendAEFromTSMComponent`.

sendMode

The value that lets you specify one of the following modes specified by corresponding constants: the reply mode for the Apple event, the interaction level, the application switch mode, the reconnection mode, and the return receipt mode. To obtain the value for this parameter, add the appropriate constants. Comprehensive details about these constants are provided in the description of the Apple Event Manager `AESend` function.

sendPriority

The value that specifies whether to put the Apple event at the back of the event queue (set with the `kAENormalPriority` flag) or at the front of the queue (`kAEHighPriority` flag).

timeOutInTicks

The length of time (in ticks) that the client application is willing to wait for the reply or return receipt from the server application before it times out. If the value of this parameter is `kNoTimeOut`, the Apple event never times out.

idleProc

A pointer to a function for any tasks (such as displaying a globe, a wristwatch, or a spinning beach ball cursor) that the application performs while waiting for a reply or a return receipt.

filterProc

A pointer to a function that accepts certain incoming Apple events that are received while the handler waits for a reply or a return receipt and filters out the rest.

Return Value
A result code. See "Text Services Manager Result Codes" (page 1647).

Discussion
Component use of Apple events and the function `SendAEFromTSMComponent` is discouraged on Mac OS X. Text service components should use Carbon text input events and the function `SendTextInputEvent` (page 1600) on Mac OS X, instead. See "Carbon Porting Notes" below for more details.

The `SendAEFromTSMComponent` function is essentially a wrapper function for the Apple Event Manager function `AESend`. See the description of `AESend` for additional necessary information, including constants for the `sendMode` parameter and result codes.

`SendAEFromTSMComponent` identifies your text service component from the `keyAEServerInstance` parameter in the Apple event specified in the `theAppleEvent` parameter. If a reference constant in a TSM document that corresponds to this parameter is found in the internal data structures of the Text Services Manager, `SendAEFromTSMComponent` adds the reference constant as the `keyAETSMDocumentRefcon` parameter to the given Apple event before sending it to the application.

If the client application is not TSM-aware, `SendAEFromTSMComponent` routes the Apple events to the floating input window to allow bottom-line input.

If your text service component changes the environment in any way—such as by modifying the A5 world or changing the current zone—while constructing an Apple event, it must restore the previous settings before sending the Apple event.

Your text service component should always use the `kCurrentProcess` constant as the target address when it creates an Apple event to send to the Text Services Manager.

Availability
Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.
Available in Mac OS X v 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Carbon Porting Notes

Note that this function is superseded by the function `SendTextInputEvent` (page 1600) on Mac OS X only. With Mac OS X, text service components must be Carbon clients. This is in contrast to Mac OS 8 and 9, where text service components must not be Carbon clients. (This restriction is due to the fact that it is potentially destabilizing for a Carbon-based component to load Carbon in the context of a non-Carbon application.) Therefore, text service components use Carbon text input events and the `SendTextInputEvent` function only on Mac OS X. The function `SendAEFromTSMComponent` must be used by components running on Mac OS 8 and 9.

On any system, the Text Services Manager automatically converts component-originated text input events to the proper form for client applications. On Mac OS X, the Text Services Manager automatically converts component-originated Carbon events to Apple events, if a client application does not provide handlers for Carbon events. Conversely, on Mac OS 8 and 9, the Text Services Manager automatically converts component-originated Apple events to Carbon events and provides these Carbon events to applications, so they have the option of handling them.

Declared In
`TextServices.h`

SendTextInputEvent

Sends Carbon text input events from a text service component to a client application.

```
OSStatus SendTextInputEvent (
   EventRef inEvent
);
```

Parameters

inEvent
A reference to the Carbon event to be sent.

Return Value
A result code. See "Text Services Manager Result Codes" (page 1647). The `SendTextInputEvent` function returns `noErr` if the event is successfully handled. If the event is not handled, the function may return a Carbon Event Manager error, as well as Apple event or Text Encoding Conversion Manager errors.

Discussion
The `SendTextInputEvent` function allows a Carbon text service component on Mac OS X to send a Carbon text input event to the Text Services Manager for dispatching to a client application. This function can be used for events of Carbon event class `kEventClassTextInput` as well as for events of class `kEventClassTSMDocumentAccess`.

If the client application does not handle a particular Carbon text input event, the Text Services Manager converts the event to the corresponding Apple event and sends it again. An exception to this is when the application is not Unicode-aware (that is, the active TSM document was not created with the `kUnicodeDocument` interface type). In this case, a `kEventUnicodeForKeyEvent` Carbon event would not be converted to the corresponding Apple event (`kUnicodeNotFromInputMethod`). In every case, if the application handles neither the Unicode Carbon text input event nor the corresponding Apple event, the Text Services Manager converts the component's text input event into a stream of "classic" key events for delivery to `WaitNextEvent` clients.

If the application has no active TSM documents or has called the function `UseInputWindow` (page 1613) to request input via the Text Services Manager's floating input window—that is, if the application does not handle the event at all—the Text Services Manager routes the component's text input event to the floating input window to allow bottom-line input.

Availability
Not available in CarbonLib.
Available in Mac OS X v 10.0 and later.
Not available to 64-bit applications.

Carbon Porting Notes

Note that this function replaces the function `SendAEFromTSMComponent` (page 1598) on Mac OS X only. With Mac OS X, text service components must be Carbon clients. This is in contrast to Mac OS 8 and 9, where text service components must not be Carbon clients. (This restriction is due to the fact that it is potentially destabilizing for a Carbon-based component to load Carbon in the context of a non-Carbon application.) Therefore, text service components use Carbon text input events and the `SendTextInputEvent` function only on Mac OS X. The function `SendAEFromTSMComponent` must be used by components running on Mac OS 8 and 9.

On any system, the Text Services Manager automatically converts component-originated text input events to the proper form for client applications. On Mac OS X, the Text Services Manager automatically converts component-originated Carbon events to Apple events, if a client application does not provide handlers for Carbon events. Conversely, on Mac OS 8 and 9, the Text Services Manager automatically converts component-originated Apple events to Carbon events and provides these Carbon events to applications, so they have the option of handling them.

Declared In
`TextServices.h`

SetDefaultInputMethod

Sets a default input method to a given script and language. (Deprecated in Mac OS X v10.5.)

Not recommended.

```
OSErr SetDefaultInputMethod (
   Component ts,
   ScriptLanguageRecord *slRecordPtr
);
```

Parameters

ts

The component identifier of the input method to be associated with the script and language combination given in the `slRecord` parameter.

slRecordPtr

A pointer to a structure of type `ScriptLanguageRecord` (page 1614). This structure describes the script and language combination to be associated with the input method specified in the `ts` parameter.

Return Value
A result code. See "Text Services Manager Result Codes" (page 1647). If the script code and language code specified in the script-language structure are incompatible, `SetDefaultInputMethod` returns the error `paramErr`.

Discussion
You should use the function `SetDefaultInputMethodOfClass` (page 1602) instead of this one.

The operating system uses `SetDefaultInputMethod` to associate an input method text service component with a given script and language. The operating system calls this function when the user expresses input method preferences through the Keyboard menu, Keyboard control panel, or other device. The associations made with this function are permanent; that is, they persist after restart. Your application should not typically need to call this function.

Availability
Not recommended. Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.
Available in Mac OS X v 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`TextServices.h`

SetDefaultInputMethodOfClass

Sets the default input method text service component for a given text service class. (Deprecated in Mac OS X v10.5.)

```
OSStatus SetDefaultInputMethodOfClass (
   Component aComp,
   ScriptLanguageRecord *slRecPtr,
   TextServiceClass tsClass
);
```

Parameters

aComp
The component identifier of the input method to be associated with the script and language combination given in the `slRecord` parameter.

slRecPtr
A pointer to a structure of type `ScriptLanguageRecord` (page 1614). This structure describes the script and language combination to be associated with the input method specified in the `ts` parameter.

tsClass
The text service class whose component and script language record you want to obtain. Pass `kKeyboardInputMethodClass` to specify a keyboard input method. Pass `kInkInputMethod` to specify an Ink input method.

Return Value
A result code. See "Text Services Manager Result Codes" (page 1647).

Discussion

Availability
Not available in CarbonLib.
Available in Mac OS X v 10.2 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`TextServices.h`

SetTextServiceLanguage

Changes the current input script and language. (Deprecated in Mac OS X v10.5.)

```
OSErr SetTextServiceLanguage (
   ScriptLanguageRecord *slRecordPtr
);
```

Parameters

slRecordPtr

A pointer to a structure of type `ScriptLanguageRecord` (page 1614) specifying the new script and language combination.

Return Value
A result code. See "Text Services Manager Result Codes" (page 1647).

Discussion
The operating system calls this function when the user switches the keyboard script, so that the Text Services Manager can synchronize the input method with the current keyboard script.

Your application should not typically need to call this function.

Availability
Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.
Available in Mac OS X v 10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`TextServices.h`

SetTextServiceProperty

Notifies a text service component that one of its properties has been selected.

```
ComponentResult SetTextServiceProperty (
   ComponentInstance ts,
   TextServicePropertyTag inPropertyTag,
   TextServicePropertyValue inPropertyValue
);
```

Parameters

ts

A Component Manager value of type `ComponentInstance` that identifies the component being called. When the Text Services Manager makes this call, it passes the `ComponentInstance` value returned by its call to the `OpenComponent` function. If an application makes this call, it may use the `ComponentInstance` value obtained from the `kEventParamTextInputSendComponentInstance` parameter of the Carbon event or the `keyAEServerInstance` parameter of an Apple event sent by the component being called. Alternately, an application may obtain a `ComponentInstance` value from a prior call to the function `OpenTextService` (page 1597).

propertySelector

A constant identifying a general property of a text service. For descriptions of the system-defined property selectors, see "Text Service Properties" (page 1640).

value

A constant specifying a particular value for the text service property. For descriptions of the system-defined property values, see "Text Service Properties" (page 1640).

Return Value
See the Component Manager documentation for a description of the `ComponentResult` data type.

Discussion
Text service components have the option of implementing a function for this call.

An application can call `SetTextServiceProperty` to request that a text service component use a specific feature or functionality of the component's program. For example, if an application knows that a Japanese input method which supports various typing methods is the currently active input method, the application can solicit the user's preference of typing methods. Then the application can call `SetTextServiceProperty` to request that the input method use the preferred typing method, for example, Roman or Kana. Currently, the only properties that are defined by the system are typing methods for Japanese input methods.

Both the Text Services Manager and client applications can call this function to manage text service properties. If client applications directly make this call, the Text Services Manager does not then play a role in the connection between the client application making the call and the text service component receiving it.

Availability
Available in CarbonLib 1.0 and later.
Available in Mac OS X v 10.0 and later.
Not available to 64-bit applications.

Declared In
`TextServices.h`

TerminateTextService

Notifies a text service component that it must terminate its operations in preparation for closing.

```
ComponentResult TerminateTextService (
   ComponentInstance ts
);
```

Parameters

ts

A Component Manager value of type `ComponentInstance` that identifies the component being called. When the Text Services Manager makes this call, it passes the `ComponentInstance` value returned by its call to the `OpenComponent` function. If an application makes this call, it may use the `ComponentInstance` value obtained from the `kEventParamTextInputSendComponentInstance` parameter of the Carbon event or the `keyAEServerInstance` parameter of an Apple event sent by the component being called. Alternately, an application may obtain a `ComponentInstance` value from a prior call to the function `OpenTextService` (page 1597).

Return Value
If the text service component needs to remain open, it should return an `OSErr` value in the component result return value. This could happen, for example, if the user chooses Cancel in response to a text service component dialog box. See the Component Manager documentation for a description of the `ComponentResult` data type.

Discussion
Text service components must implement a function for this call.

The Text Services Manager calls `TerminateTextService` before closing the component instance. A text service component must use this opportunity to confirm any inline input in progress. If this call is made to the last open instance of a text service component, the component should hide any open palette windows. If it is an input method, the component should not dispose of its menu handle if it has a menu.

The Text Services Manager makes this call either on its own or in response to application-interface calls it receives from client applications. Client applications may directly make this call, but the Text Services Manager does not then play a role in the connection between the client application making the call and the text service component receiving it.

Availability
Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.
Available in Mac OS X v 10.0 and later.
Not available to 64-bit applications.

Declared In
`TextServices.h`

TextServiceEventRef

Provides an opportunity for a text service component to handle a Carbon event.

```
ComponentResult TextServiceEventRef (
   ComponentInstance ts,
   EventRef event
);
```

Parameters

ts
A Component Manager value of type `ComponentInstance` that identifies the component being called. When the Text Services Manager makes this call, it passes the `ComponentInstance` value returned by its call to the `OpenComponent` function. If an application makes this call, it may use the `ComponentInstance` value obtained from the `kEventParamTextInputSendComponentInstance` parameter of the Carbon event or the `keyAEServerInstance` parameter of an Apple event sent by the component being called. Alternately, an application may obtain a `ComponentInstance` value from a prior call to the function `OpenTextService` (page 1597).

event
A reference to the Carbon event being passed to the component.

Return Value
If the text service component handles the event, it should return a nonzero value for `componentResult`. If it does not handle the event, it should return 0. Note that the Text Services Manager clones an event before passing it to a component, so any changes made to the contents of an event by the text service have no effect on the original event. See the Component Manager documentation for a description of the `ComponentResult` data type.

Discussion
Carbon text service components (that is, Mac OS X text services) must implement a function for this call.

The Text Services Manager automatically passes raw keyboard Carbon events (events of class `kEventClassKeyboard`) and some Carbon mouse events to text service components associated with an active TSM document. The Text Services Manager passes mouse-click events (`kEventMouseDown`, `kEventMouseUp`, `kEventMouseDragged`) to active text services directly. However, the Text Services Manager does not send `kEventMouseMoved` events to text service components. Instead, when a mouse-moved event occurs inside an inline input region (as registered via an application call to the `TSMSetInlineInputRegion` function), the Text Services Manager promotes the `kEventMouseMoved` event to the window-specific `kEventWindowCursorChange` event, which it then sends to the text service. For more details, see the function `TSMSetInlineInputRegion` (page 1611).

Both the Text Services Manager and client applications can call this function to send Carbon events to components. If client applications directly make this call, the Text Services Manager does not then play a role in the connection between the client application making the call and the text service component receiving it.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X v 10.0 and later.
Not available to 64-bit applications.

Declared In
`TextServices.h`

TSMCopyInputMethodEnabledInputModes

Obtain the array of the enabled (and visible) input modes for a component. (Deprecated in Mac OS X v10.5.)

```
Boolean TSMCopyInputMethodEnabledInputModes (
   Component inComponent,
   CFArrayRef *outInputModeArray
);
```

Parameters

inComponent
The component whose input modes you want to obtain.

outInputModeArray
On return, points to an array of the enabled and visible input modes for the specified component. This function is meaningful only for input methods that adopt the input mode protocol. If the component passed is not input mode-savvy, the returned array is `NULL`. It is the responsibility of the caller to release the returned array.

Discussion
You use this function to allow an input method to query the system for the subset of its own input modes that are enabled. This allows you to omit from the component UI any input modes that are disabled by the user or the system. The enabled input modes returned in the array are always visible ones. That is, the array contains those input modes for which `kTSInputModeIsVisibleKey` is true; non-visible input modes are not tracked by the system.

Availability
Available in Mac OS X v 10.3 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`TextServices.h`

TSMGetActiveDocument

Obtains the active TSM document in the current application context.

```
TSMDocumentID TSMGetActiveDocument (
   void
);
```

Parameters

Return Value
If the Text Services Manager has enabled bottom line input because no TSM document is active, `NULL` is returned. See the description of the `TSMDocumentID` data type.

Discussion
This function can be useful to identify whether the currently active TSM document belongs to the application, or whether it may belong to a control or a plug-in which has user focus within the application's window.

Availability
Available in CarbonLib 1.3 and later.
Available in Mac OS X v 10.0 and later.

Declared In
`TextServices.h`

TSMGetDocumentProperty

Obtains a TSM document property.

```
OSStatus TSMGetDocumentProperty (
   TSMDocumentID docID,
   TSMDocumentPropertyTag propertyTag,
   UInt32 bufferSize,
   UInt32 *actualSize,
   void *propertyBuffer
);
```

Parameters

docID

The `TSMDocumentID` that identifies the document whose property you want to obtain.

propertyTag

A tag that specifies the property you want to obtain.

bufferSize

The size of the data pointed to by the `propertyBuffer` parameter. See the Discussion for what to supply.

actualSize

On return, the actual size of the data.

propertyBuffer

On return, a pointer to the property data.

Return Value

A result code. See "Text Services Manager Result Codes" (page 1647).

Discussion

You can call the function `TSMGetDocumentProperty` to retrieve arbitrary data with a specific TSM document. You associated arbitrary data with a TSM document by calling the function `TSMSetDocumentProperty`.

Input methods can call the function `TSMGetDocumentProperty` to determine whether the application that owns the document supports the glyph ID specification (`kTSMDocumentPropertySupportGlyphInfo`).

You can call the function `TSMGetDocumentProperty` to check for the following predefined properties:

- `kUseFloatingWindowTag`. The presence of this property indicates the document is using bottom-line floating window for input. See "Collection Tags" (page 1644) for more information.
- `kUnicodeDocument`. The presence of this property indicates the document is Unicode-savvy. See "Unicode Identifiers" (page 1643) for more information.
- `kTSMTEInterfaceType`. The presence of this property indicates a TSM Text Edit interface. See "TSM Document Interfaces" (page 1642) for more information.
- `kTextService`. The presence of this property indicates the document is not Unicode-savvy. See "TSM Document Interfaces" (page 1642) for more information.

These properties do not have any data associated with them. If the function `TSMGetDocumentProperty` returns `noErr` when you call the function with one of these properties, it indicated the property is present.

Typically you need to call this function twice, as follows:

1. Pass the document ID for the document, the tag that specifies the property you want to obtain, 0 for the `bufferSize` parameter, `NULL` for the `actualSize` parameter, and `NULL` for the `propertyBuffer` parameter.

2. Allocate enough space for a buffer of the returned size, then call the function again, passing a pointer in the `propertyBuffer` parameter. On return, the pointer references the property data.

Availability
Not available in CarbonLib.
Available in Mac OS X v 10.2 and later.

Declared In
`TextServices.h`

TSMInputModePaletteLoadButtons

Notifies the input mode palette of changes to the controls for an input method and replaces the current controls with the new control array. (Deprecated in Mac OS X v10.5.)

```
void TSMInputModePaletteLoadButtons (
   CFArrayRef paletteButtonsArray
);
```

Parameters

`paletteButtonsArray`

A CFArray that contains descriptions of the controls. Use a CFDictionary to describe each control. See "Input Mode Palette Control Keys" (page 1634) for a description of the keys you can supply.

Availability
Available in Mac OS X v 10.4 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`TextServices.h`

TSMInputModePaletteUpdateButtons

Notifies the input mode palette of changes to the controls for an input method and updates the controls. (Deprecated in Mac OS X v10.5.)

```
void TSMInputModePaletteUpdateButtons (
   CFArrayRef paletteButtonsArray
);
```

Parameters

`paletteButtonsArray`

A CFArray that contains descriptions of the controls. Use a CFDictionary to describe each control. See "Input Mode Palette Control Keys" (page 1634) for a description of the keys you can supply.

Discussion
This function updates controls based on the control tag ID. It doe not replace or remove existing controls.

Availability
Available in Mac OS X v 10.4 and later.
Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`TextServices.h`

TSMRemoveDocumentProperty

Removes a property from a TSM document.

```
OSStatus TSMRemoveDocumentProperty (
   TSMDocumentID docID,
   TSMDocumentPropertyTag propertyTag
);
```

Parameters

docID

The `TSMDocumentID` that identifies the document whose property you want to obtain.

propertyTag

A tag that specifies the property you want to remove.

Return Value
A result code. See "Text Services Manager Result Codes" (page 1647).

Availability
Not available in CarbonLib.
Available in Mac OS X v 10.2 and later.

Declared In
`TextServices.h`

TSMSelectInputMode

Sets the specified input method input mode as the current input source. (Deprecated in Mac OS X v10.5.)

```
OSStatus TSMSelectInputMode (
   Component inComponent,
   CFStringRef inInputMode
);
```

Parameters

inComponent

The component whose input mode you want to set.

inInputMode

The input mode you want to set as the current input source.

Discussion
You use this function to allow an input method to select one of its own input modes as the current input source and update the Text Input menu icon in the menu bar. This function is only meaningful for input methods that adopt the input mode protocol.

Availability
Available in Mac OS X v 10.3 and later.
Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`TextServices.h`

TSMSetDocumentProperty

Sets a property for a TSM document.

```
OSStatus TSMSetDocumentProperty (
   TSMDocumentID docID,
   TSMDocumentPropertyTag propertyTag,
   UInt32 propertySize,
   void *propertyData
);
```

Parameters

docID
The `TSMDocumentID` that identifies the document whose property you want to set.

propertyTag
A tag that specifies the property you want to set.

propertySize
The size of the property data.

propertyData
A pointer to the property data.

Return Value
A result code. See "Text Services Manager Result Codes" (page 1647).

Discussion
You can call the function `TSMSetDocumentProperty` to associate arbitrary data with a specific TSM document. You can call the function `TSMGetDocumentProperty` to retrieve arbitrary data.

If your application supports input of unencoded glyphs you must notify the Text Service Manager and input methods by setting the glyph ID specification (`kTSMDocumentPropertySupportGlyphInfo`) as a property of each TSM document.

Availability
Not available in CarbonLib.
Available in Mac OS X v 10.2 and later.

Declared In
`TextServices.h`

TSMSetInlineInputRegion

Defines a region within a TSM document in which inline input can occur. (Deprecated in Mac OS X v10.5.)

```
OSStatus TSMSetInlineInputRegion (
   TSMDocumentID inTSMDocument,
   WindowRef inWindow,
   RgnHandle inRegion
);
```

Parameters

inTSMDocument

The identification number of a TSM document created by a prior call to the `NewTSMDocument` (page 1595) function.

inWindow

A reference to the window that contains the inline input session. You can pass `NULL` for this parameter to indicate the window that currently has user focus.

inRegion

The current inline input region. This region should be in coordinates local to the port associated with the window specified in the `inWindow` parameter. The region must be recomputed each time the text content of the inline input session changes (such as after an Update Active Input Area event) and when the region moves for other reasons (such as window resizing or scrolling). If you pass `NULL` for this parameter, the Text Service Manager defaults to intercepting mouse events for the window's entire content region.

Return Value

A result code. See "Text Services Manager Result Codes" (page 1647).

Discussion

The `TSMSetInlineInputRegion` function informs the Text Services Manager of the region occupied by an inline input session. If certain mouse events (such as clicks and mouse-moved events) occur within this region, the Text Services Manager forwards these events to the current input method, so the component can respond to the user's actions.

When a mouse-moved event occurs inside an inline input region (as registered via the `TSMSetInlineInputRegion` function), the Text Services Manager promotes the `kEventMouseMoved` event, after it has been received by the applicable control or window, to the window-specific `kEventWindowCursorChange` event. The Text Services Manager first delivers the `kEventWindowCursorChange` event to the active input method, then, if it is not handled, to any other active text services. If the event has still not been handled, the Text Services Manager finally passes the event to the application's `kEventWindowCursorChange` event handler, if any. This event-dispatching process gives applications that need to see the low-level mouse-moved events a chance to see these events first, while providing a mechanism for text services and applications to act on these events without conflict. After completing this process, the Carbon Event Manager converts any `kEventWindowCursorChange` event that remains unhandled to a "classic" mouse-moved event for `WaitNextEvent` clients.

If an application does not call this function, when an input method is active the Text Services Manager by default intercepts mouse events in the entire content region of the window that currently has user focus.

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X v 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`TextServices.h`

UseInputWindow

Associates a floating input window with one or more TSM documents.

```
OSErr UseInputWindow (
   TSMDocumentID idocID,
   Boolean useWindow
);
```

Parameters

idocID

The TSM document ID of the particular TSM document to be associated with the floating input window. If `NULL`, this call affects all your application's TSM documents.

useWindow

Indicates whether to use the floating input window. Pass `TRUE` if you want to use a floating window; pass `FALSE` if you do not want to use a floating window.

Return Value

A result code. See "Text Services Manager Result Codes" (page 1647).

Discussion

The Text Services Manager provides a floating input window for your application's use if you call `UseInputWindow` with a value of `TRUE` in the `useWindow` parameter. To specify inline input instead, call `UseInputWindow` with a value of `FALSE` in the `useWindow` parameter.

The default value for `useWindow` is `FALSE`; if you do not call `UseInputWindow`, the Text Services Manager assumes that your application wants to use inline input. If your application wants to save the user's choice, it can put the last-used value for `useWindow` in a preferences file before quitting.

If you pass a valid TSM document ID for the `idocID` parameter, the `useWindow` parameter affects only that TSM document. If you pass `NULL` for the `idocID` parameter, the `useWindow` parameter affects all your application's TSM documents, including documents you create after making this call.

Availability

Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.

Available in Mac OS X v 10.0 and later.

Not available to 64-bit applications.

Declared In

`TextServices.h`

Data Types

TSM Document Interface Type

Defines an interface type for a TSM document.

```
typedef OSType TSMDocumentInterfaceType;
```

Discussion
As of Mac OS X version 10.3, TSM interface types are also stored as TSM document properties. Interface types are a subset of TSM document properties; not all properties are interface types. Once a TSM document is created, you can easily find out its interface types at document creation. See "TSM Document Interfaces" (page 1642) for a list of the possible interface types.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`TextServices.h`

InterfaceTypeList

An array of four-character codes identifying Text Services Manager interface types.

```
typedef OSType InterfaceTypeList[1];
```

Discussion
The `InterfaceTypeList` type is used in the function `NewTSMDocument` (page 1595) to identify the type of interfaces that an application supports and in the function `GetServiceList` (page 1589) to identify the types of interfaces that are currently available. See "TSM Document Interfaces" (page 1642) for a list of the possible interfaces.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TextServices.h`

ScriptLanguageRecord

Identifies a specific script-language combination.

```
struct ScriptLanguageRecord {
    ScriptCode fScript;
    LangCode fLanguage;
};
typedef struct ScriptLanguageRecord ScriptLanguageRecord;
```

Fields

`fScript`
A `ScriptCode` value identifying a particular set of written characters (for example, Roman versus Cyrillic) and their encoding.

`fLanguage`
A `LangCode` value identifying a particular language (for example, English), as represented using a particular `ScriptCode` value.

Discussion
Structures of type ScriptLanguageRecord are used in the functions `SetDefaultInputMethod` (page 1601), `GetDefaultInputMethod` (page 1585), `SetTextServiceLanguage` (page 1603), and `GetTextServiceLanguage` (page 1590).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TextServices.h`

ScriptLanguageSupport

Lists a component's supported scripts and languages.

```
struct ScriptLanguageSupport {
    short fScriptLanguageCount;
    ScriptLanguageRecord fScriptLanguageArray[1];
};
typedef struct ScriptLanguageSupport ScriptLanguageSupport;
typedef ScriptLanguageSupport * ScriptLanguageSupportPtr;
typedef ScriptLanguageSupportPtr * ScriptLanguageSupportHandle;
```

Fields

`fScriptLanguageCount`
An integer specifying the number of `ScriptLanguageRecord` structures provided in the `fScriptLanguageArray` field.

`fScriptLanguageArray`
A variable-length array of structures of type `ScriptLanguageRecord` (page 1614). Each of these structures identifies a specific script-language combination.

Discussion
A structure of type `ScriptLanguageSupport` is used in the function `GetScriptLanguageSupport` (page 1588) to list all of a component's supported scripts and languages. If you are a component developer filling out a `ScriptLanguageSupport` structure, you should start with the component's primary script and language as specified in the `componentFlags` field of its `ComponentDescription` structure.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TextServices.h`

TextServiceInfo

Identifies a single text service component by name and `Component` value.

```
struct TextServiceInfo {
    Component fComponent;
    Str255 fItemName;
};
typedef struct TextServiceInfo TextServiceInfo;
typedef TextServiceInfo *TextServiceInfoPtr;
```

Fields

`fComponent`
A Component Manager value of type `Component`. A `Component` value is a pointer to an opaque structure called a `ComponentRecord` that describes a component. You must supply a `Component` value in the function `OpenTextService` (page 1597).

`fItemName`
A Pascal string with the name of a text service component. (The script system to use for displaying the string is specified in the `componentFlags` field of a `ComponentDescription` structure.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TextServices.h`

TextServiceList

Lists one or more text service components by name and `Component` value.

```
struct TextServiceList {
    short fTextServiceCount;
    TextServiceInfo fServices[1];
};
typedef struct TextServiceList TextServiceList;
typedef TextServiceList * TextServiceListPtr;
typedef TextServiceListPtr * TextServiceListHandle;
```

Fields
`fTextServiceCount`
An integer specifying the number of `TextServiceInfo` structures in the text service component list provided in the `fServices` field.

`fServices`
A variable-length array of structures of type `TextServiceInfo` (page 1615). Each `TextServiceInfo` structure identifies a specific component by name and `Component` value.

Discussion
A structure of type `TextServiceInfo` is used in the function `GetServiceList` (page 1589) to list of all the text service components of a specified type that are currently available on a system.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TextServices.h`

TextServicePropertyValue

Defines a data type for text service property values.

```
typedef SInt32 TextServicePropertyValue;
```

Discussion
The property values associated with this data type are "Text Services Property Values" (page 1641). Note that these values are declared as `CFStringRef` data types, so they require a cast to the `SInt32` data type before you can supply them as a `TextServicePropertyValue`.

Availability
Available in Mac OS X v10.3 and later.

Declared In
TextServices.h

TSMContext

A reference to an opaque object that specifies a TSM context.

```
typedef struct OpaqueTSMContext * TSMContext;
```

Availability
Available in Mac OS X v10.0 through Mac OS X v10.2.

Declared In
TextServices.h

TSMDocumentID

A reference to an opaque object that identifies a specific TSM document.

```
typedef struct OpaqueTSMDocumentID * TSMDocumentID;
```

Discussion
Each time a client application calls the function NewTSMDocument (page 1595), the Text Services Manager creates an opaque internal structure called a TSM Document and returns a pointer to the document's identification number.

Availability
Available in Mac OS X v10.0 and later.

Declared In
TextServices.h

TSMGlyphInfo

Describes one glyph embedded in a run of text.

```
struct TSMGlyphInfo {
    CFRange range;
    ATSFontRef fontRef;
    UInt16 collection;
    UInt16 glyphID;
};
typedef struct TSMGlyphInfo TSMGlyphInfo;
```

Fields

range
: A CFRange data structure that specifies, in UTF-16 offsets, a range within the text to which this TSMGlyphInfo data structure applies. I

fontRef
: An ATS font reference that specifies the font with which the glyph should be displayed. Note that the character collection ROS (Adobe Registry, Ordering, Supplement) is a property of the font.

`collection`

A glyph collection type that specifies how the `glyphID` parameter should be interpreted. When the value is `kGlyphCollectionID`, `glyphID` specifies the glyph's ID. When `collection` is a non-zero value, it specifies a character collection and `glyphID` specifies a CID. Note that `collection` must match the character collection of the font specified by the `fontRef` parameter.

When collections do not match, the `TSMGlyphInfo` data structure is invalid and should be ignored. You need to supply an ATSUI constant of type `GlyphCollection` to specify the character set you want to use. See *Inside Mac OS X: ATSUI Reference* for a list of the glyph collection constants you can specify.

`glyphID`

A glyph ID that specifies the glyph to use you to use in place of the current glyph. If you pass 0 instead of specifying a glyph, the `TSMGlyphInfo` data structure is used to attach a font to a range of text. In this case, the `fontRef` parameter specifies a font that should be used to display the range of text specified by the `range` parameter. This is useful when using characters in the Unicode private use area. Windings and other Windows based pi fonts are examples of such characters. When `glyphID` is zero, `collection` should also be zero and applications should ignore its value.

Discussion
The `TSMGlyphInfo` data structure is used as an item in the `TSMGlyphInfoArray` (page 1618) data structure. You use these structures to provide TSM with glyph and font information when you want to override the current glyph or font.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`TextServices.h`

TSMGlyphInfoArray

Contains an array of glyph information structures.

```
struct TSMGlyphInfoArray {
    ItemCount numGlyphInfo;
    TSMGlyphInfo glyphInfo[1];
};
typedef struct TSMGlyphInfoArray TSMGlyphInfoArray;
```

Fields
`numGlyphInfo`

The number of items in the `glyphInfo` array.

`glyphInfo`

An array of glyph information structures.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`TextServices.h`

Constants

Attribute Bits for TSM Document Access Carbon Events

Represents TSM document attributes.

```
enum {
    kTSMDocAccessFontSizeAttributeBit = 0,
    kTSMDocAccessEffectiveRangeAttributeBit = 1
};
```

Constants

`kTSMDocAccessFontSizeAttributeBit`

When this bit is set, indicates to obtain font size information; used in the Carbon event `kEventTSMDocumentAccessGetFont`.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`kTSMDocAccessEffectiveRangeAttributeBit`

When this bit is set, indicates to obtain effective range information used in the Carbon events `kEventTSMDocumentAccessGetFont` and `kEventTSMDocumentAccessGetGlyphInfo`.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

Discussion

These bit assignments are used for the TSM document access attribute parameters. You can use these bits to specify desired (optional) attributes in the `kEventParamTSMDocAccessRequestedCharacterAttributes` parameter available for the events `kEventTSMDocumentAccessGetFont` and `kEventTSMDocumentAccessGetGlyphInfo`.

Availability

Not available in CarbonLib.

Available in Mac OS X v 10.3 and later.

Attribute Masks for TSM Document Access Carbon Events

Used to set or test for document-access attributes.

```
typedef UInt32 TSMDocAccessAttributes;
enum {
    kTSMDocAccessFontSizeAttribute = 1L << kTSMDocAccessFontSizeAttributeBit,
    kTSMDocAccessEffectiveRangeAttribute = 1L <<
kTSMDocAccessEffectiveRangeAttributeBit
};
```

Constants

`kTSMDocAccessFontSizeAttribute`

Use to set or test for the `kTSMDocAccessFontSizeAttributeBit` bit.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`kTSMDocAccessEffectiveRangeAttribute`
Use to set or test for the `kTSMDocAccessEffectiveRangeAttributeBit` bit.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

Availability
Not available in CarbonLib.
Available in Mac OS X v 10.3 and later.

Carbon Event Class for TSM Document Access

Defines a constant for the Carbon event class used to allow TSM access to application documents content.

```
enum {
    kEventClassTSMDocumentAccess = 'tdac'
};
```

Constants
`kEventClassTSMDocumentAccess`
Used to request and deliver document content information. The events associated with this class provide text access, text attribute access, and transaction information. See "Carbon Events for TSM Document Access" (page 1620) for a list of the events defined for this class.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

Discussion
The Text Services Manager (TSM) dispatches TSM document access events as Carbon events. You must install a Carbon event handler to access these events because they are not available through AppleEvent handlers.

Text Services dispatches these Carbon events through the function `SendTextInputEvent` (page 1600).

Availability
Not available in CarbonLib.
Available in Mac OS X v 10.3 and later.

Carbon Events for TSM Document Access

Define constants for the Carbon events associated with the TSM document access event class.

```
enum {
    kEventTSMDocumentAccessGetLength = 1,
    kEventTSMDocumentAccessGetSelectedRange = 2,
    kEventTSMDocumentAccessGetCharactersPtr = 3,
    kEventTSMDocumentAccessGetCharactersPtrForLargestBuffer = 4,
    kEventTSMDocumentAccessGetCharacters = 5,
    kEventTSMDocumentAccessGetFont = 6,
    kEventTSMDocumentAccessGetGlyphInfo = 7,
    kEventTSMDocumentAccessLockDocument = 8,
    kEventTSMDocumentAccessUnlockDocument = 9
};
```

Constants

`kEventTSMDocumentAccessGetLength`

Returns the number of 16-bit Unicode characters in the document.

You can obtain the following event parameters from this event:

- `kEventParamTSMDocAccessSendComponentInstance`. This parameter is provided by the text service originating the event.
- `kEventParamTSMDocAccessSendRefCon`. The TSM function `SendTextInputEvent` (page 1600), called by an input method, inserts this parameter before dispatching the event to the user focus.
- `kEventParamTSMDocAccessCharacterCount`. The size of the document in `UniChar` characters.

You can obtain the same information from this event as you can by calling the function `CFStringGetLength`, passing the document content formatted as a CFString.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`kEventTSMDocumentAccessGetSelectedRange`

Returns the selection range in the document.

You can obtain the following event parameters from this event:

- `kEventParamTSMDocAccessSendComponentInstance`. This parameter is provided by the text service originating the event. This can be `NULL` for input methods of the palette class, such as the typography panel.
- `kEventParamTSMDocAccessSendRefCon`. The TSM function `SendTextInputEvent` (page 1600), called by an input method, inserts this parameter before dispatching the event to the user focus.
- `kEventParamTSMDocAccessReplyCharacterRange`. The selection range as a CFRange in `UniChar` characters. If the selection is empty, the range identifies the insertion point and the range specifies a length of 0.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`kEventTSMDocumentAccessGetCharactersPtr`

Returns a Unicode pointer to the entire document content. Handle this event when your application has access to the entire document. If your application has access to a cache, use the event `kEventTSMDocumentAccessGetCharactersPtrForLargestBuffer`.

Some text engines may not support this event for reasons that are implementation-dependent. For example, a text engine backing store may consist of legacy encoding runs. It may also consist of unflattened Unicode, stored as a B-tree of text blocks. For such reasons, a text engine may reject a request for a pointer to a flattened Unicode buffer. Note that text access through this pointer is to be strictly read-only, so any changes to the document should be made through TSM text input events, such as `kEventTextInputUpdateActiveInputArea` or `kEventTextInputUnicodeText`. This pointer is valid only during a transaction surrounded by document lock/unlock events, or until an event causes the document to change, such as dispatching `kEventTextInputUpdateActiveInputArea` or `kEventTextInputUnicodeText` events, whichever occurs first.

You can obtain the following event parameters from this event:

- `kEventParamTSMDocAccessSendComponentInstance`. This parameter is provided by the input method originating the event.
- `kEventParamTSMDocAccessSendRefCon`. The TSM function `SendTextInputEvent` (page 1600), called by an input method, inserts this parameter before dispatching the event to the user focus.
- `kEventParamTSMDocAccessReplyCharactersPtr`. The `UniChar` pointer to the document.

You can obtain the same information from this event as you can by calling the function `CFStringGetCharactersPtr`, passing the document content formatted as a CFString.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`kEventTSMDocumentAccessGetCharactersPtrForLargestBuffer`

Returns a Unicode pointer to a portion of the document. Handle this event when your application has access to a cache. If your application has access to the entire document, use the event `kEventTSMDocumentAccessGetCharactersPtr`.

Some text engines keep text in unflattened Unicode—for example, stored as a B-tree of text blocks. Sometimes, especially for chunks of text near the insertion point, the text engine caches a chunk of text to which it can readily provide a pointer. But because the text is not flattened, the text engine might reject a request for such a pointer. See the Discussion for more information.

Note that text access through this pointer is strictly read-only, so any changes to the document should be made through TSM text input events, such as `kEventTextInputUpdateActiveInputArea` or `kEventTextInputUnicodeText`. This pointer is valid only during a transaction surrounded by document lock/unlock, or until an event causes the document to change, such as dispatching `kEventTextInputUpdateActiveInputArea` or `kEventTextInputUnicodeText` events.

You can obtain the following event parameters from this event:

- `kEventParamTSMDocAccessSendComponentInstance`. This parameter is provided by the input method originating the event.
- `kEventParamTSMDocAccessSendRefCon`. The TSM function `SendTextInputEvent`, called by an input method, inserts this parameter before dispatching the event to the user focus.
- `kEventParamTSMDocAccessSendCharacterIndex`. The location in the document for which the caller wants a pointer to a buffer of text that includes that location. This buffer could be available from a cache due to recent interaction near that location, such as the insertion point.
- `kEventParamTSMDocAccessReplyCharactersPtr`. The `UniChar` pointer to a portion of the document text.
- `kEventParamTSMDocAccessReplyCharacterRange`. A `CFRange` value for the text returned by the text pointer. The initial offset in the range is document-relative.

This event is similar to calling the function `CFStringGetCharactersPtr` on a portion of the document content formatted as a CFString, except that the substring is determined by the text engine.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`kEventTSMDocumentAccessGetCharacters`

This fills a caller provided buffer with Unicode characters in the specified range. This event is equivalent to calling the function `CFStringGetCharacters` on the document content treated as a CFString.

You can obtain the following event parameters from this event:

- `kEventParamTSMDocAccessSendComponentInstance`. This parameter is provided by the input method originating the event.
- `kEventParamTSMDocAccessSendRefCon`. The TSM function `SendTextInputEvent`, called by an input method, inserts this parameter before dispatching the event to the user focus.
- `kEventParamTSMDocAccessSendCharacterRange`. The range of text that should be copied into the buffer provided by the caller.
- `kEventParamTSMDocAccessSendCharactersPtr`. A buffer provided by the caller to contain the specified range of `UniChar` characters. This buffer is identical in usage to the one used in the function `CFStringGetCharacters`.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`kEventTSMDocumentAccessGetFont`

Returns font, font size, and the range over which these attributes are constant. Where the font/font size attributes span multiple characters, an effective range (over which requested attributes are constant) is returned by the text engine.

You can obtain the following event parameters from this event:

- `kEventParamTSMDocAccessSendComponentInstance`. This parameter is provided by the input method originating the event.
- `kEventParamTSMDocAccessSendRefCon`. The TSM function `SendTextInputEvent`, called by an input method, inserts this parameter before dispatching the event to the user focus.
- `kEventParamTSMDocAccessSendCharacterIndex`. The location in the document for which the caller would like font information.
- `kEventParamTSMDocAccessRequestedCharacterAttributes`. A `TSMDocAccessAttributes` bit field filled out with the desired attributes. Applicable values for this event are: `kTSMDocAccessFontSizeAttribute` which requests font size information through the `kEventParamTSMDocAccessReplyFontSize` parameter, and `kTSMDocAccessEffectiveRangeAttribute` which requests the text range over which font or font/size is constant.
- `kEventParamTSMDocAccessReplyATSFont`. The `ATSFontRef` for the location specified by the caller.
- `kEventParamTSMDocAccessReplyFontSize`. The font size for the requested location. This is an optional reply parameter. Return this information if `kTSMDocAccessFontSizeAttribute` is specified in the bit field passed as the `kEventParamTSMDocAccessRequestedCharacterAttributes` parameter.
- `kEventParamTSMDocAccessSendCharacterRange`. The maximum range of text the caller cares about. This is used to restrict the area of interest to the caller so the text engine doesn't process more characters than necessary in order to return an effective range.
- `kEventParamTSMDocAccessEffectiveRange`. The range of text over which both font and size are constant, within the bounds of the `kEventParamTSMDocAccessSendCharacterRange` parameter. This is an optional reply parameter. Return this information if `kTSMDocAccessEffectiveRangeAttribute` is specified in the bit field passed as the `kEventParamTSMDocAccessRequestedCharacterAttributes` parameter.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`kEventTSMDocumentAccessGetGlyphInfo`

Returns glyph info and the range covered by that glyph. Where a glyph spans multiple characters, the effective range, represented by the glyph, is returned by the application.

You can obtain the following event parameters from this event:

- `kEventParamTSMDocAccessSendComponentInstance`. This parameter is provided by the input method originating the event.
- `kEventParamTSMDocAccessSendRefCon`. The TSM function `SendTextInputEvent`, called by an input method, inserts this parameter before dispatching the event to the user focus.
- `kEventParamTSMDocAccessSendCharacterIndex`. The location in the document for which the caller would like glyph information.
- `kEventParamTSMDocAccessRequestedCharacterAttributes`. A `TSMDocAccessAttributes` bit field filled out with the information desired. The applicable value for this event is `kTSMDocAccessEffectiveRangeAttribute`, which requests the text range represented by a glyph.
- `kEventParamTSMDocAccessReplyATSUGlyphSelector`. The glyph used to display the range of text returned in the `kEventParamTSMDocAccessEffectiveRange` parameter. If the glyph used is the one that ATSUI would normally derive, this parameter can be omitted.
- `kEventParamTSMDocAccessEffectiveRange`. The range of text displayed as a glyph ID or CID. This is an optional reply parameter. Return this information if `kTSMDocAccessEffectiveRangeAttribute` is specified in the bit field passed as the `kEventParamTSMDocAccessRequestedCharacterAttributes` parameter.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`kEventTSMDocumentAccessLockDocument`

Notifies the application that it should not change its document's text content (on its own) while a text service is involved in a transaction. The application should not allow changes, for example, by its secondary threads. This type of event defines how a text service can obtain access to a document in a way that ensures data integrity during its transaction. The event can be used to prevent the application from letting its secondary threads modify the document while a text service is busy servicing an event, such as a key event, or some user interaction with text-service-provided user interface such as a menu selection. Also, while the document is locked, a text service is free to request pointer access to the document's text content (if this is supported by the application's text engine.) These lock-related events should be implemented using a retention counting scheme. Most applications will not support this kind of threading, so implementation of these events in the text engine are optional. In most text engines, the implementation of these events should be trivial, that is, just maintain a simple semaphore. TSM itself will implicitly lock/unlock around normal entry points into a text service, such as when it delivers key events to an input method, but there may be times when document changes can be driven by an input method without TSM involvement, such as the Carbon events involved when the user interacts with some user interface. In this case, the input method must manage locking, if the application supports it. However, the logic in an input method should not depend on whether TSM is in the call chain or not, and TSM should not depend on whether an input method performs correctly. This is why the lock mechanism needs to be some kind of retention counting scheme instead of a simple on and off mechanism. Document lock support is optional on the part of the text engine (if it is not threaded). TSM implicitly locks/unlocks the document around delivery of events to input methods, if the application supports it.You can obtain the following event parameters from this event:

- `kEventParamTSMDocAccessSendComponentInstance`. This parameter is provided by the input method originating the event.
- `kEventParamTSMDocAccessSendRefCon`. The TSM function `SendTextInputEvent` (page 1600), called by an input method, inserts this parameter before dispatching the event to the user focus.
- `kEventParamTSMDocAccessLockCount`. The resulting retention count of locks on the document.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`kEventTSMDocumentAccessUnlockDocument`

Unlock the document so the application text engine is free to initiate changes again. (See `kEventTSMDocumentAccessLockDocument`).

You can obtain the following event parameters from this event:

- `kEventParamTSMDocAccessSendComponentInstance`. This parameter is provided by the input method originating the event.
- `kEventParamTSMDocAccessSendRefCon`. The TSM function `SendTextInputEvent`, called by an input method, inserts this parameter before dispatching the event to the user focus.
- `kEventParamTSMDocAccessLockCount`. The resulting retention count of locks on the document.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

Discussion

Text-access events are very similar in design to the CFString API. You can think of an entire document as a flattened Unicode string, and the events in this interface can access any portion of it. Just as the text is Unicode, the text offsets are also Unicode.

The event `kEventTSMDocumentAccessGetSelectedRange` allows a text service to obtain text near the insertion point (or selection), but access is by no means restricted to this vicinity. Use the event `kEventTSMDocumentAccessGetLength` to obtain the size of the document.

Supporting these events effectively provide hooks into the text engine, but it is understood that access to a document in this way is strictly read-only. Where direct access to document content cannot be provided through a pointer, the requested text can be copied instead. Situations where a pointer may not be available from the text engine include the following:

- The pointer requires conversion of text in Mac encodings to Unicode.
- The pointer requires sparse Unicode text blocks to be flattened into a single buffer.

The idea is to minimize copying and converting text encodings where possible. The text service typically begins by asking for a document pointer through the event `kEventTSMDocumentAccessGetCharactersPtr`. If this fails, it typically falls back to the event `kEventTSMDocumentAccessGetCharactersPtrForLargestBuffer`, specifying a location of interest. If this fails, it falls back to `kEventTSMDocumentAccessGetCharacters`, specifying a range of interest. Of course, when requesting small amounts of data with such a few characters on either side of the insertion point, there is no obligation to optimize in this way. It's valid to simply use `kEventTSMDocumentAccessGetCharacters`.

The text engine is entirely free to deny a request for a text pointer for these or any other implementation-specific reason.

Availability
Not available in CarbonLib.
Available in Mac OS X v 10.3 and later.

Carbon Event Parameters for General TSM Events

Define general parameters for TSM events.

```
enum {
    kEventParamTSMSendRefCon      = 'tsrc',
    kEventParamTSMSendComponentInstance = 'tsci'
};
```

Constants

`kEventParamTSMSendRefCon`
This parameter is equivalent to the text input parameter `kEventParamTextInputSendRefCon`; the parameter data type is `typeLongInteger`.
Available in Mac OS X v10.3 and later.
Declared in `CarbonEvents.h`.

`kEventParamTSMSendComponentInstance`
This parameter is equivalent to the text input parameter `kEventParamTextInputSendComponentInstance`; the parameter data type is `typeComponentInstance`.
Available in Mac OS X v10.3 and later.
Declared in `CarbonEvents.h`.

Availability
Not available in CarbonLib.
Available in Mac OS X v 10.3 and later.

Carbon Event Parameters for TSM Document Access

Define document access parameters for TSM events.

```
enum {
    kEventParamTSMDocAccessSendRefCon = kEventParamTSMSendRefCon,
    kEventParamTSMDocAccessSendComponentInstance =
kEventParamTSMSendComponentInstance,
    kEventParamTSMDocAccessCharacterCount = 'tdct',
    kEventParamTSMDocAccessReplyCharacterRange = 'tdrr',
    kEventParamTSMDocAccessReplyCharactersPtr = 'tdrp',
    kEventParamTSMDocAccessSendCharacterIndex = 'tdsi',
    kEventParamTSMDocAccessSendCharacterRange = 'tdsr',
    kEventParamTSMDocAccessSendCharactersPtr = 'tdsp',
    kEventParamTSMDocAccessRequestedCharacterAttributes = 'tdca',
    kEventParamTSMDocAccessReplyATSFont = 'tdaf',
    kEventParamTSMDocAccessReplyFontSize = 'tdrs',
    kEventParamTSMDocAccessEffectiveRange = 'tder',
    kEventParamTSMDocAccessReplyATSUGlyphSelector = 'tdrg',
    kEventParamTSMDocAccessLockCount = 'tdlc',
    typeATSFontRef                = 'atsf',
    typeGlyphSelector             = 'glfs'
};
```

Constants

`kEventParamTSMDocAccessSendRefCon`
The TSM function `SendTextInputEvent` (page 1600), called by an input method, inserts this parameter before dispatching the event to the user focus. The parameter data type is `typeLongInteger`.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`kEventParamTSMDocAccessSendComponentInstance`
The parameter data type is `typeComponentInstance`.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`kEventParamTSMDocAccessCharacterCount`
The parameter data type is `typeCFIndex`.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`kEventParamTSMDocAccessReplyCharacterRange`
The parameter data type is `typeCFRange`.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`kEventParamTSMDocAccessReplyCharactersPtr`
The parameter data type is `typePtr`.
Available in Mac OS X v10.3 and later.
Declared in `CarbonEvents.h`.

`kEventParamTSMDocAccessSendCharacterIndex`
The parameter data type is `typeCFIndex`.
Available in Mac OS X v10.3 and later.
Declared in `CarbonEvents.h`.

`kEventParamTSMDocAccessSendCharacterRange`
The parameter data type is `typeCFRange`.
Available in Mac OS X v10.3 and later.
Declared in `CarbonEvents.h`.

`kEventParamTSMDocAccessSendCharactersPtr`
The parameter data type is `typePtr`.
Available in Mac OS X v10.3 and later.
Declared in `CarbonEvents.h`.

`kEventParamTSMDocAccessRequestedCharacterAttributes`
The parameter data type is `typeUInt32`.
Available in Mac OS X v10.3 and later.
Declared in `CarbonEvents.h`.

`kEventParamTSMDocAccessReplyATSFont`
The parameter data type is `typeATSFontRef`.
Available in Mac OS X v10.3 and later.
Declared in `CarbonEvents.h`.

`kEventParamTSMDocAccessReplyFontSize`
The parameter data type is `typeFloat`.
Available in Mac OS X v10.3 and later.
Declared in `CarbonEvents.h`.

`kEventParamTSMDocAccessEffectiveRange`
The parameter data type is `typeRange`.
Available in Mac OS X v10.3 and later.
Declared in `CarbonEvents.h`.

`kEventParamTSMDocAccessReplyATSUGlyphSelector`
The parameter data type is `typeGlyphSelector`.
Available in Mac OS X v10.3 and later.
Declared in `CarbonEvents.h`.

`kEventParamTSMDocAccessLockCount`
The parameter data type is `typeCFIndex`.
Available in Mac OS X v10.3 and later.
Declared in `CarbonEvents.h`.

`typeATSFontRef`
The parameter data type is `ATSFontRef`.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

`typeGlyphSelector`
The parameter data type is `ATSUGlyphSelector`.

Available in Mac OS X v10.3 and later.

Declared in `CarbonEvents.h`.

Discussion
See "Carbon Events for TSM Document Access" (page 1620) for more information on these parameters and the information they contain for a specific event.

Availability
Not available in CarbonLib.
Available in Mac OS X v 10.3 and later.

Component Flags

Specify flags used for input method components.

```
enum {
    bTakeActiveEvent = 15,
    bHandleAERecording = 16,
    bScriptMask = 0x00007F00,
    bLanguageMask = 0x000000FF,
    bScriptLanguageMask = bScriptMask + bLanguageMask
};
```

Constants
`bTakeActiveEvent`
This bit is set if the component takes an active event,

Available in Mac OS X v10.0 and later.

Declared in `TextServices.h`.

`bHandleAERecording`
This bit is set if the component takes care of recording Apple Events.

Available beginning with version 2.0.

Declared in `TextServices.h`.

`bScriptMask`
Specifies bits 8 - 14.

Available in Mac OS X v10.0 and later.

Declared in `TextServices.h`.

`bLanguageMask`
Specifies bits 0 - 7.

Available in Mac OS X v10.0 and later.

Declared in `TextServices.h`.

`bScriptLanguageMask`
Specifies bits 0 - 14.

Available in Mac OS X v10.0 and later.

Declared in `TextServices.h`.

Document Property Tags

Specify property tags for a TSM document.

```
typedef OSType      TSMDocumentPropertyTag;
enum {

kTSMDocumentSupportGlyphInfoPropertyTag = 'dpgi',
kTSMDocumentUseFloatingWindowPropertyTag = 'uswm',
kTSMDocumentUnicodeInputWindowPropertyTag = 'dpub',
kTSMDocumentSupportDocumentAccessPropertyTag = 'dapy',
kTSMDocumentRefconPropertyTag = 'refc',
kTSMDocumentInputModePropertyTag = 'imim',
kTSMDocumentPropertySupportGlyphInfo =
        kTSMDocumentSupportGlyphInfoPropertyTag,
kTSMDocumentPropertyUnicodeInputWindow =
        kTSMDocumentUnicodeInputWindowPropertyTag,
kTSMDocumentTextServicePropertyTag = kTextServiceDocumentInterfaceType,
kTSMDocumentUnicodePropertyTag = kUnicodeDocumentInterfaceType,
kTSMDocumentTSMTEPropertyTag  = kTSMTEDocumentInterfaceType
};
```

Constants

`kTSMDocumentSupportGlyphInfoPropertyTag`
The existence of this property in a TSM document indicates that the event handlers associated with he TSM document are aware of the TSM `GlyhInfo` data structure. This structure allows the input source producing text to apply Glyph IDs, CIDs, or fonts to subranges of text produced. This is useful or characters in Unicode private use area, such as Windings. For more information, see Technical Note TN2079 Glyph Access Protocol. By convention, this value can be a `UInt32` with a value of 0, but this is arbitrary. Available in TSM 1.5, in Mac OS X 10.2 and later.

Available in Mac OS X v10.3 and later.

Declared in `TextServices.h`.

`kTSMDocumentUseFloatingWindowPropertyTag`
The presence of this property tag indicates that the TSM document should use the TSM floating input window to handle input from input methods. This form of input does not support Unicode input by default, unless the property `kTSMDocumentUnicodeInputWindowPropertyTag` is set.

Available in Mac OS X v10.3 and later.

Declared in `TextServices.h`.

`kTSMDocumentUnicodeInputWindowPropertyTag`
The presence of this property tag indicates that although the TSM document has been told to use the TSM floating input window to handle input from input methods, the floating window is to support Unicode input. This is useful when non input-related activity is to produce Unicode, such as keyboard navigation.

Available in Mac OS X v10.3 and later.

Declared in `TextServices.h`.

`kTSMDocumentSupportDocumentAccessPropertyTag`

The presence of this property tag indicates that the event handlers associated with this TSM document support the TSM document access event suite (see "Carbon Events for TSM Document Access" (page 1620).) This property also indicates that the handler for the TSMevent `kEventTextInputUpdateActiveInputArea` supports the `replaceRange` parameter and that the handler is a Carbon event handler, not an AppleEvent handler.

Available in Mac OS X v10.3 and later.

Declared in `TextServices.h`.

`kTSMDocumentRefconPropertyTag`

The property value initially contains the `refcon` value passed to the function `NewTSMDocument`. This property is useful for changing the `refcon` value after the TSM document has been created. The `refcon` value is a `long`, the same as that passed to `NewTSMDocument`. Property is value-dependent; see the Discussion for more information.

Available in Mac OS X v10.3 and later.

Declared in `TextServices.h`.

`kTSMDocumentInputModePropertyTag`

The property value indicates which input mode should be used by the current keyboard-class input method. It is useful for temporarily restricting text input to a subset of characters normally produced by an input method in a given script, such as Katakana for Japanese input. See "Text Service Properties" (page 1640) for more details. Also note that this property tag and value are passed unchanged to the function "SetTextServiceProperty" (page 1603), so it also serves as a text service property tag. See `kTextServiceInputModePropertyTag` for discussion on the values associated with this property.

The property value is a `CFStringRef` data type. With the function `TSMGetTextServiceProperty`, the behavior is that of a Copy function. The implementation of `SetTextServiceProperty` (in the component) retains or copies the CFString object. In either case the caller is responsible for releasing the reference. Property is value-dependent; see the Discussion for more information.

Available in Mac OS X v10.3 and later.

Declared in `TextServices.h`.

`kTSMDocumentPropertySupportGlyphInfo`

You should no longer use this property.

Available in Mac OS X v10.2 and later.

Declared in `TextServices.h`.

`kTSMDocumentPropertyUnicodeInputWindow`

You should no longer use this property.

Available in Mac OS X v10.3 and later.

Declared in `TextServices.h`.

`kTSMDocumentTextServicePropertyTag`

Specifies a non-Unicode savvy document. This property is equivalent to a pre-existing document interface type.

Available in Mac OS X v10.3 and later.

Declared in `TextServices.h`.

`kTSMDocumentUnicodePropertyTag`

This property is equivalent to the Unicode document interface type.

Available in Mac OS X v10.3 and later.

Declared in `TextServices.h`.

`kTSMDocumentTSMTEPropertyTag`
This property is equivalent to the TSMTE document interface type.

Available in Mac OS X v10.3 and later.

Declared in `TextServices.h`.

Discussion
You can use the functions `TSMSetDocumentProperty` (page 1611) and `TSMGetDocumentProperty` (page 1607) to set and get arbitrary property data needed by your application.

Unless otherwise noted, all properties are read-only, value-independent, and available in TSM version 2.2, which is the version available starting in Mac OS X version 10.3.

Value-independent properties are used where the existence of the property, and not its value, is sufficient. These properties can read by other clients, and are most often used by input methods. For example, input methods can query the current TSM document to see if supports unrestricted Unicode input, or if it supports the `GlyphInfo` protocol.

Value-dependent properties are used when the value associated with a property is meaningful.

Input Method Identifier

Specifies a keyboard input method text service.

```
enum {
    kInputMethodService = kKeyboardInputMethodClass
};
```

Constants
`kInputMethodService`
A four-character code identifying an input method text service. Specifies that the older constant name `kInputMethodService` is equivalent to the newer constant name `kKeyboardInputMethodClass`.

Available in Mac OS X v10.0 and later.

Declared in `TextServices.h`.

Input Mode Dictionary Key

Defines a string for the input mode dictionary key that you can use in the Component bundle info.plist.

```
#define kComponentBundleInputModeDictKey  CFSTR("ComponentInputModeDict")
```

Discussion
If you are developing an input method, you use this key in the component bundle info.plist to identify a dictionary of input mode information. The dictionary should contain keys to identify input modes—see "Individual Input Mode Keys" (page 1635)—and should have the form described in the Discussion section of the function `CopyTextServiceInputModeList` (page 1579). The function `CopyTextServiceInputModeList` returns an input mode dictionary.

Input Mode Palette Menu Definition Keys

Defines keys used to describe the items in a pull-down menu.

```
#define kTSInputModePaletteItemTitleKey   CFSTR("tsInputModePaletteItemTitleKey")
#define kTSInputModePaletteItemKeyEquivalentKey
CFSTR("tsInputModePaletteItemKeyEquivalentKey")
#define kTSInputModePaletteItemKeyEquivalentModifiersKey
CFSTR("tsInputModePaletteItemKeyEquivalentModifiersKey")
```

Constants

`kTSInputModePaletteItemTitleKey`
A CFString that specifies a menu item title. Use - for a separator.

`kTSInputModePaletteItemKeyEquivalentKey`
A CFString that specifies a menu item keyboard shortcut .

`kTSInputModePaletteItemKeyEquivalentModifiersKey`
A CFNumber that specifies a menu item keyboard shortcut modifier (from Events.h).

Discussion

These keys are returned by the function `GetInputModePaletteMenu` (page 1587), in the `outMenuItemsArray` parameter. For information on the structure of the CFDictionary, see the TextServices.h header file.

Input Mode Palette Control Keys

Defines keys used to describe controls for an input palette.

```
#define kTSInputModePaletteItemTypeKey  CFSTR("tsInputModePaletteItemTypeKey")
#define kTSInputModePaletteItemIconKey   CFSTR("tsInputModePaletteItemIconKey")
#define kTSInputModePaletteItemAltIconKey   CFSTR("tsInputModePaletteItemAltIconKey")
#define kTSInputModePaletteItemStateKey   CFSTR("tsInputModePaletteItemStateKey")
#define kTSInputModePaletteItemEnabledKey   CFSTR("tsInputModePaletteItemEnabledKey")
#define kTSInputModePaletteItemIDKey    CFSTR("tsInputModePaletteItemIDKey")
```

Constants

`kTSInputModePaletteItemTypeKey`
A CFNumber that specifies the type of control (0: push button, 1: toggle button, 2: pull-down menu),

`kTSInputModePaletteItemIconKey`
A CFString that specifies an icon file name. The file should be located in the input method bundle resource directory, so this is just the file name, not full path.

`kTSInputModePaletteItemAltIconKey`
A CFString that specifies an alternate icon file name. The file should be located in the input method bundle resource directory, so this is just the file name, not full path.

`kTSInputModePaletteItemStateKey`
A CFNumber that specifies the state of the control (0: clear or unpressed, 1: checked or pressed, 2: mixed).

`kTSInputModePaletteItemEnabledKey`
A CFBoolean that specifies the enabled state of the control.

`kTSInputModePaletteItemIDKey`
A CFNumber that specifies a `UInt32` tag ID for the control.

Discussion

You use these keys in a CFDictionary that contains control descriptions passed to the functions `TSMInputModePaletteLoadButtons` (page 1609) and `TSMInputModePaletteUpdateButtons` (page 1609). For information on the structure of the CFDictionary, see the TextServices.h header file.

Individual Input Mode Keys

Defines keys used to identify input modes in an input mode dictionary.

```
#define kTSInputModeListKey CFSTR("tsInputModeListKey")
#define kTSInputModeMenuIconFileKey CFSTR("tsInputModeMenuIconFileKey")
#define kTSInputModeAlternateMenuIconFileKey
        CFSTR("tsInputModeAlternateMenuIconFileKey")
#define kTSInputModePaletteIconFileKey
        CFSTR("tsInputModePaletteIconFileKey")
#define kTSInputModeDefaultStateKey CFSTR("tsInputModeDefaultStateKey")
#define kTSInputModeScriptKey CFSTR("tsInputModeScriptKey")
#define kTSInputModePrimaryInScriptKey
        CFSTR("tsInputModePrimaryInScriptKey")
#define kTSInputModeIsVisibleKey      CFSTR("tsInputModeIsVisibleKey")
#define kTSInputModeKeyEquivalentModifiersKey
        CFSTR("tsInputModeKeyEquivalentModifiersKey")
#define kTSInputModeKeyEquivalentKey
            CFSTR("tsInputModeKeyEquivalentKey")
#define kTSInputModeJISKeyboardShortcutKey
        CFSTR("tsInputModeJISKeyboardShortcutKey")
```

Discussion
If you are developing an input method, you use these keys in a dictionary of input mode information. The Component bundle info.plist should an input mode dictionary key that identifies the dictionary—see "Input Mode Dictionary Key" (page 1633). The dictionary should have the form described in the Discussion section of the function `CopyTextServiceInputModeList` (page 1579). The function `CopyTextServiceInputModeList` returns an input mode dictionary.

Interfaces

Specify types of text services interfaces.

```
enum {
    kTextService = 'tsvc'
};
```

Constants
`kTextService`
A four-character code identifying a text service of any kind (including input methods). This value is also used to identify non-Unicode TSM documents.

Available in Mac OS X v10.0 and later.

Declared in `TextServices.h`.

Discussion
This constant is used in arrays of type `InterfaceTypeList` (page 1614). In addition to this constants, the constant `kTSMTEInterfaceType` (`'tmTE'`), from TSMTE.h, is a supported interface type that allows TextEdit to provide automatic inline input support in TextEdit documents.

Language and Script Constants

Specify the language or script is not known or neutral.

```
enum {
    kUnknownLanguage = 0xFFFF,
    kUnknownScript = 0xFFFF,
    kNeutralScript = 0xFFFF
};
```

Constants

`kUnknownLanguage`
Specifies an unknown language.
Available in Mac OS X v10.0 and later.
Declared in `TextServices.h`.

`kUnknownScript`
Specifies an unknown script.
Available in Mac OS X v10.0 and later.
Declared in `TextServices.h`.

`kNeutralScript`
Specifies a neutral script.
Available in Mac OS X v10.0 and later.
Declared in `TextServices.h`.

Low-level Routine Selectors

Specify low level routines which are dispatched directly to the Component Manager.

```
enum {
    kCMGetScriptLangSupport = 0x0001,
    kCMInitiateTextService = 0x0002,
    kCMTerminateTextService = 0x0003,
    kCMActivateTextService = 0x0004,
    kCMDeactivateTextService = 0x0005,
    kCMTextServiceEvent = 0x0006,
    kCMGetTextServiceMenu = 0x0007,
    kCMTextServiceMenuSelect = 0x0008,
    kCMFixTextService = 0x0009,
    kCMSetTextServiceCursor = 0x000A,
    kCMHidePaletteWindows = 0x000B,
    kCMGetTextServiceProperty = 0x000C,
    kCMSetTextServiceProperty = 0x000D
};
```

Constants

`kCMGetScriptLangSupport`
Specifies the function `GetScriptLanguageSupport` (page 1588); Component Manager call selector 1.
Available in Mac OS X v10.0 and later.
Declared in `TextServices.h`.

`kCMInitiateTextService`
Specifies the function `InitiateTextService` (page 1593); Component Manager call selector 2.
Available in Mac OS X v10.0 and later.
Declared in `TextServices.h`.

`kCMTerminateTextService`
Specifies the function `TerminateTextService` (page 1604); Component Manager call selector 3.

Available in Mac OS X v10.0 and later.

Declared in `TextServices.h`.

`kCMActivateTextService`
Specifies the function `ActivateTextService` (page 1577); Component Manager call selector 4.

Available in Mac OS X v10.0 and later.

Declared in `TextServices.h`.

`kCMDeactivateTextService`
Specifies the function `DeactivateTextService` (page 1580); Component Manager call selector 5.

Available in Mac OS X v10.0 and later.

Declared in `TextServices.h`.

`kCMTextServiceEvent`
Specifies the function `TextServiceEventRef` (page 1605); Component Manager call selector 6.

Available in Mac OS X v10.0 and later.

Declared in `TextServices.h`.

`kCMGetTextServiceMenu`
Specifies the function `GetTextServiceMenu` (page 1590); Component Manager call selector 7.

Available in Mac OS X v10.0 and later.

Declared in `TextServices.h`.

`kCMTextServiceMenuSelect`
Component Manager call selector 8.

Available in Mac OS X v10.0 and later.

Declared in `TextServices.h`.

`kCMFixTextService`
Specifies the function `FixTextService` (page 1583); Component Manager call selector 9.

Available in Mac OS X v10.0 and later.

Declared in `TextServices.h`.

`kCMSetTextServiceCursor`
Component Manager call selector 10.

Available in Mac OS X v10.0 and later.

Declared in `TextServices.h`.

`kCMHidePaletteWindows`
Specifies the function `HidePaletteWindows` (page 1592); Component Manager call selector 11.

Available in Mac OS X v10.0 and later.

Declared in `TextServices.h`.

`kCMGetTextServiceProperty`
Specifies the function `GetTextServiceProperty` (page 1591); Component Manager call selector 12.

Available in Mac OS X v10.0 and later.

Declared in `TextServices.h`.

`kCMSetTextServiceProperty`
Specifies the function `SetTextServiceProperty` (page 1603); Component Manager call selector 13.

Available in Mac OS X v10.0 and later.

Declared in `TextServices.h`.

New Low-level Routine Selector

Specifies new low-level routine that are dispatched directly to the Component Manager.

```
enum {
    kCMUCTextServiceEvent = 0x000E
};
```

Constants

`kCMUCTextServiceEvent`
Component Manager call selector 14.

Available in Mac OS X v10.0 and later.

Declared in `TextServices.h`.

Text Service Classes

Specify text service classes supported by TSM.

```
enum {
    kKeyboardInputMethodClass = 'inpm',
    kInkInputMethodClass = 'ink ',
    kCharacterPaletteInputMethodClass = 'cplt',
    kSpeechInputMethodClass = 'voic',
    kOCRInputMethodClass = 'ocr '
};
typedef OSType TextServiceClass;
```

Constants

`kKeyboardInputMethodClass`
Specifies a text service class for keyboard input methods. Behavior is exclusive. Input methods in this class are normally associated with a Mac ScriptCode or Unicode, although they can be associated with several scripts by adopting the input mode protocol.

Available in Mac OS X v10.2 and later.

Declared in `TextServices.h`.

`kInkInputMethodClass`
Specifies a text service class for Ink input methods. Behavior is additive. Text services in the Ink class do not belong to any given script in the sense that those of the Keyboard class do. Once selected, this kind of text service remains active regardless of the current keyboard script. Although text services in this class are keyboard script agnostic, similar to input methods of the keyboard class they can still profess to produce only those Unicodes that are encoded in the Mac encoding specified in their component description record or their implementation of the `GetScriptLanguageSupport` component call.

Available in Mac OS X v10.2 and later.

Declared in `TextServices.h`.

`kCharacterPaletteInputMethodClass`
: Specifies a text service class for Character Palette input methods. Behavior is additive. Text services in the character palette class do not belong to any given script in the same sense that do those of the keyboard class. Once selected, this kind of text service remains active regardless of the current keyboard script. Although text services in this class are keyboard script agnostic, similar to input methods of the keyboard class, they can still produce only those Unicodes that are encoded in the Mac encoding specified in their component description record or their implementation of the `GetScriptLanguageSupport` component call. Unlike input methods in the keyboard class, multiple such text services can be activate in parallel. Mac OS X provides a System user interface to allow the user to both enable and select multiple such input methods.

  Available in Mac OS X v10.2 and later.

  Declared in `TextServices.h`.

`kSpeechInputMethodClass`
: Specifies a text service class for Speech input methods. Behavior is additive. Similar to Character palette class.

  Available in Mac OS X v10.3 and later.

  Declared in `TextServices.h`.

`kOCRInputMethodClass`
: Specifies a text service class for Optical Character Recognition input methods. Behavior is additive. Similar to Character palette class.

  Available in Mac OS X v10.3 and later.

  Declared in `TextServices.h`.

Discussion
Text service classes fall in two categories or behaviors. Text services that belong to some classes are exclusive of one another within a given Mac script code, such input methods of the keyboard class. Input Methods of other classes are additive in nature, regardless of the current keyboard script.

Within a given class and script, exclusive input methods can only be activated one at a time. Input methods in additive classes are keyboard script agnostic and can be active in parallel with other text services in the same class, such as multiple character palettes.

These are the same as the component subtype for the component description.

Text Service Version

Specifies the interface type for version 2.

```
enum {
    kTextServiceVersion2 = 'tsv2'
};
```

Constants
`kTextServiceVersion2`
: The interface type for V2 interfaces

  Available in Mac OS X v10.0 through Mac OS X v10.2.

  Declared in `TextServices.h`.

Text Service Properties

Specify a feature or functionality of a component.

```
typedef OSType TextServicePropertyTag;
enum {
    kTextServiceInputModePropertyTag = kTSMDocumentInputModePropertyTag,
    kIMJaTypingMethodRoman = 'roma',
    kIMJaTypingMethodKana = 'kana',
    kIMJaTypingMethodProperty = kTextServiceJaTypingMethodPropertyTag,
    kTextServiceJaTypingMethodPropertyTag = 'jtyp'
};
```

Constants

`kTextServiceInputModePropertyTag`
Specifies the input mode property for input methods. This property is a CFString object that uniquely identifies which input mode should be made current by a keyboard class input method, if possible. This property tag is identical to the tag `kTSMDocumentInputModePropertyTag` passed to the function `TSMDocumentProperty`. This allows the tag and value to be passed through without interpretation.

Available in Mac OS X v10.3 and later.

Declared in `TextServices.h`.

`kIMJaTypingMethodRoman`
Not recommended. Specify Japanese input in Roman script.

Available in Mac OS X v10.0 and later.

Declared in `TextServices.h`.

`kIMJaTypingMethodKana`
Not recommended. Specify Japanese input in Kana script.

Available in Mac OS X v10.0 and later.

Declared in `TextServices.h`.

`kIMJaTypingMethodProperty`
Deprecated. Specify the typing method as Japanese input. This property is deprecated. Use the tag `kTextServiceInputModePropertyTag` instead.

Available in Mac OS X v10.0 and later.

Declared in `TextServices.h`.

`kTextServiceJaTypingMethodPropertyTag`
Deprecated.Use the tag `kTextServiceInputModePropertyTag` instead.

Available in Mac OS X v10.3 and later.

Declared in `TextServices.h`.

Discussion

Text Service Property constants are used in `SetTextServiceProperty` (page 1603) and `GetTextServiceProperty` (page 1591). The only property that is recommend for you to use is the property `kTextServiceInputModePropertyTag`.

Input modes are either generic (pre-defined by TSM), or specific to an input method. An example of a generic input mode is Katakana input (Japanese) where input in a text field needs to be restricted to that character subset. Another is Roman input mode. This is useful to temporarily provide Roman input from an input method that normally allows text input in another script. The advantage to using Roman input mode over

forcing the keyboard script to Roman is that the same user interface for the input method continues to be available to the user, even though the input script changed. An example of a special input mode (input method specific) is Hanin input mode in Traditional Chinese input methods.

To temporarily change the current input mode from whatever it is to a generic one, use the function `GetTextServiceProperty` to obtain the current input mode, then call the function `SetTextServiceProperty` to switch to the generic mode. When done using the generic input mode, you can restore the original input mode.

You can find out what input modes are supported by an input method by calling the function `CopyTextServiceInputModeList`. If the input method does not support a specified input mode, the functions `GetTextServiceProperty` and `SetTextServiceProperty` return the result `tsmComponentPropertyUnsupportedErr`. The function `GetTextServiceProperty` returns the result `tsmComponentPropertyNotFoundErr`.

Text Services Property Values

Define values for the text services input mode property tags.

```
#define kTextServiceInputModeRoman     CFSTR("com.apple.inputmethod.Roman")
#define kTextServiceInputModePassword
        CFSTR("com.apple.inputmethod.Password")
#define kTextServiceInputModeJapaneseHiragana
    CFSTR("com.apple.inputmethod.Japanese.Hiragana")
#define kTextServiceInputModeJapaneseKatakana
    CFSTR("com.apple.inputmethod.Japanese.Katakana")
#define kTextServiceInputModeJapaneseFullWidthRoman
    CFSTR("com.apple.inputmethod.Japanese.FullWidthRoman")
#define kTextServiceInputModeJapaneseHalfWidthKana
    CFSTR("com.apple.inputmethod.Japanese.HalfWidthKana")
#define kTextServiceInputModeJapanesePlaceName
    CFSTR("com.apple.inputmethod.Japanese.PlaceName")
#define kTextServiceInputModeJapaneseFirstName
    CFSTR("com.apple.inputmethod.Japanese.FirstName")
#define kTextServiceInputModeJapaneseLastName
    CFSTR("com.apple.inputmethod.Japanese.LastName")
#define kTextServiceInputModeBopomofo
    CFSTR("com.apple.inputmethod.TradChinese.Bopomofo")
#define kTextServiceInputModeTradChinesePlaceName
    CFSTR("com.apple.inputmethod.TradChinese.PlaceName")
#define kTextServiceInputModeHangul
    CFSTR("com.apple.inputmethod.Korean.Hangul")
#define kTextServiceInputModeJapanese
    CFSTR("com.apple.inputmethod.Japanese")
#define kTextServiceInputModeTradChinese
    CFSTR("com.apple.inputmethod.TradChinese")
#define kTextServiceInputModeSimpChinese
    CFSTR("com.apple.inputmethod.SimpChinese")
#define kTextServiceInputModeKorean
    CFSTR("com.apple.inputmethod.Korean")
```

Constants

`kTextServiceInputModeRoman`
Specifies to restrict output to Roman characters only.

`kTextServiceInputModePassword`
`kTextServiceInputModeJapaneseHiragana`
Specifies to restrict output to Hiragana characters only (no conversion to Kanji, that is, yomi).

`kTextServiceInputModeJapaneseKatakana`
Specifies to restrict output to Katakana characters only (no conversion to Kanji).

`kTextServiceInputModeJapaneseFullWidthRoman`
`kTextServiceInputModeJapaneseHalfWidthKana`
`kTextServiceInputModeJapanesePlaceName`
`kTextServiceInputModeJapaneseFirstName`
`kTextServiceInputModeJapaneseLastName`
`kTextServiceInputModeBopomofo`
Specifies to restrict output to Bopomofo characters only (no conversion to Han).

`kTextServiceInputModeTradChinesePlaceName`
Specifies to restrict output to traditional chines place name.

`kTextServiceInputModeHangul`
Specifies to restrict output to Hangul syllables only (no conversion to Hanja).

`kTextServiceInputModeJapanese`
Specifies unrestricted Japanese output.

`kTextServiceInputModeTradChinese`
Specifies traditional Chinese generic (unrestricted) input mode.

`kTextServiceInputModeSimpChinese`
Specifies simplified Chinese generic (unrestricted) input mode.

`kTextServiceInputModeKorean`
Specifies Korean generic (unrestricted) output (Hanja possible).

Discussion
These values require a cast from the `CFStringRef` data type to an `SInt32` data type before they can be used in text services functions.

Text Services Object Attributes

Specify characteristics of text services.

```
#define kKeyboardInputMethodTypeName        "\pkeyboardinputmethod"
#define kHandwritingInputMethodTypeName     "\phandwritinginputmethod"
#define kSpeechInputMethodTypeName          "\pspeechinputmethod"
#define kTokenizeServiceTypeName            "\ptokenizetextservice"
#define kInteractiveTextServiceTypeName     "\pinteractivetextservice"
#define kInputMethodModeName                "\pinputmethodmode"
#define kInputMethodModeVariantName         "\pinputmethodvariantmode"
#define kTextServiceModeName                "\ptextservicemode"
#define kTextServiceNeedsInlineAppMode      "\ptextservicesneedsinlineapp"
#define kTextServiceNeedsGetProtocolMode "\ptextservicesneedsGetProtocol"
#define kTextServiceAnyAppMode              "\ptextservicesanyapp"
```

TSM Document Interfaces

Specify types of TSM document interfaces.

```
enum {
    kTextServiceDocumentInterfaceType = kTextService,
    kTSMTEDocumentInterfaceType   = 'tmTE',
    kUnicodeDocumentInterfaceType = 'udoc',
    };
```

Constants

`kTextServiceDocumentInterfaceType`

A four-character code identifying a TSM document type for traditional (non-Unicode) TSM documents. This is the traditional TSM document type. It does not support Unicode. TSM converts all Unicode produced by input methods to the Mac encoding represented by the current keyboard script (or the Mac encoding specified by the input method producing text.) Full Unicode input sources may not be selectable when this TSM document is active.

Available in Mac OS X v10.3 and later.

Declared in `TextServices.h`.

`kTSMTEDocumentInterfaceType`

Deprecated. Specifies a TSM document type for TSMTE document (see `kTSMTEInterfaceType` in TSMTE.h). This requests automatic management of inline input sessions by TextEdit (the text engine.) See *Technote TE27 - Inline Input for TextEdit with TSMTE*. This document interface type should no longer be used because TextEdit has been replaced by MLTE.

Available in Mac OS X v10.3 and later.

Declared in `TextServices.h`.

`kUnicodeDocumentInterfaceType`

Specifies a TSM document type for Unicode-savvy applications. TSM pass through all Unicode text unchanged. When this TSM document is active, the full range of input sources is available to the user, such as Unicode keyboard layouts.

Available in Mac OS X v10.3 and later.

Declared in `TextServices.h`.

Discussion

These constants are used in arrays of type `InterfaceTypeList` (page 1614). TSM Interface types, as of Mac OS X 10.3, are also stored as TSM document properties, so once a TSM document is created, you can easily find out its interface types at document creation.

Unicode Identifiers

Specify constants that identify Unicode components and documents.

```
enum {
    kUnicodeDocument = 'udoc',
    kUnicodeTextService = 'utsv'
};
```

Constants

`kUnicodeDocument`

A four-character code that identifies a Unicode TSM document, for use by Unicode-savvy applications.

Available in Mac OS X v10.0 and later.

Declared in `TextServices.h`.

`kUnicodeTextService`
Specifies a component type for a Unicode text service.

Available in Mac OS X v10.0 and later.

Declared in `TextServices.h`.

Collection Tags

Specify collection tags.

```
enum {
    kInteractiveServicesTag = 'tmin',
    kLocaleIDTag = 'loce',
    kTextInputObjectTag = 'tiot',
    kLocaleObjectRefTag = 'lobj',
    kLocaleRefTag = 'lref',
    kKeyboardInputMethodContextTag = 'kinp',
    kKeyboardLocaleObjectRefTag = 'kilo',
    kHandwritingInputMethodContextTag = 'hinp',
    kHandwritingLocaleObjectRefTag = 'hilo',
    kSpeechInputMethodContextTag = 'sinp',
    kSpeechLocaleObjectRefTag = 'silo',
    kPasswordModeTag = 'pwdm',
    kRefconTag = 'refc',
    kUseFloatingWindowTag = 'uswm',
    kReadOnlyDocumentTag = 'isro',
    kSupportsMultiInlineHolesTag = 'minl',
    kProtocolVersionTag = 'nprt',
    kTSMContextCollectionTag = 'tsmx'
};
```

Constants

`kUseFloatingWindowTag`
Specifies the use of a bottom-line floating window for an input method.

Available in Mac OS X v10.0 through Mac OS X v10.2.

Declared in `TextServices.h`.

Discussion

All the constants in this enumeration, except `kUseFloatingWindowTag`, are reserved for future use.

Input Mode Variants

Specify variant tags for input modes.

```
enum {
    kIM2ByteInputMode = '2byt',
    kIM1ByteInputMode = '1byt',
    kIMDirectInputMode = 'dinp'
};
```

Constants

`kIM2ByteInputMode`
Specifies a double-byte input mode.
Available in Mac OS X v10.0 through Mac OS X v10.2.
Declared in `TextServices.h`.

`kIM1ByteInputMode`
Specifies a single-byte input mode.
Available in Mac OS X v10.0 through Mac OS X v10.2.
Declared in `TextServices.h`.

`kIMDirectInputMode`
Specifies a direct input mode.
Available in Mac OS X v10.0 through Mac OS X v10.2.
Declared in `TextServices.h`.

Discussion
These constants are reserved for future use.

Input Mode - Standard Tags

Specify standard tags for input method modes.

```
enum {
    kIMRomanInputMode = 'romn',
    kIMPasswordInputMode = 'pasw',
    kIMXingInputMode = 'xing',
    kIMHuaInputMode = 'huam',
    kIMPinyinInputMode = 'piny',
    kIMQuweiInputMode = 'quwe',
    kIMCangjieInputMode = 'cgji',
    kIMJianyiInputMode = 'jnyi',
    kIMZhuyinInputMode = 'zhuy',
    kIMB5CodeInputMode = 'b5cd',
    kIMKatakanaInputMode = 'kata',
    kIMHiraganaInputMode = 'hira'
};
```

Discussion
These constants are reserved for future use.

Locale Object Attributes

Specify attributes of a locale object.

```
enum {
    kNeedsInputWindow = 1,
    kHandlesUpdateRegion = 2,
    kHandlesGetRegion = 3,
    kHandlesPos2Offset = 4,
    kHandlesOffset2Pos = 5,
    kInPasswordMode = 6,
    kHandleMultipleHoles = 7,
    kDocumentIsReadOnly = 8
};
```

Discussion
These constants are reserved for future use.

Version Constants

Specify versions of the Text Services Manager.

```
enum {
    kTSMVersion = 0x0150,
    kTSM15Version = kTSMVersion,
    kTSM20Version = 0x0200,
    kTSM22Version = 0x0220,
    kTSM23Version = 0x0230
};
```

Constants

`kTSMVersion`
Specifies the version of the Text Services Manager is 1.5
Available in Mac OS X v10.0 and later.
Declared in `TextServices.h`.

`kTSM15Version`
Specifies the version of the Text Services Manager is 1.5
Available in Mac OS X v10.3 and later.
Declared in `TextServices.h`.

`kTSM20Version`
Specifies the version of the Text Services Manager is 2.0 (Mac OS X v10.0).
Available in Mac OS X v10.3 and later.
Declared in `TextServices.h`.

`kTSM22Version`
Specifies the version of the Text Services Manager is 2.2 (Mac OS X 1v0.3).
Available in Mac OS X v10.3 and later.
Declared in `TextServices.h`.

`kTSM23Version`
Specifies the version of the Text Services Manager is 2.3 (Mac OS X v10.4).
Available in Mac OS X v10.4 and later.
Declared in `TextServices.h`.

Result Codes

The most common result codes returned by Text Services Manager are listed below.

Result Code	Value	Description
`tsmComponentNoErr`	0	Component result: no error Available in Mac OS X v10.0 and later.
`tsmUnsupScriptLanguageErr`	-2500	Specified script and language are not supported Available in Mac OS X v10.0 and later.
`tsmInputMethodNotFoundErr`	-2501	Specified input method cannot be found Available in Mac OS X v10.0 and later.
`tsmNotAnAppErr`	-2502	The caller was not an application Available in Mac OS X v10.0 and later.
`tsmAlreadyRegisteredErr`	-2503	The caller is already TSM-initialized Available in Mac OS X v10.0 and later.
`tsmNeverRegisteredErr`	-2504	The caller is not TSM-aware Available in Mac OS X v10.0 and later.
`tsmInvalidDocIDErr`	-2505	Invalid TSM document ID Available in Mac OS X v10.0 and later.
`tsmTSMDocBusyErr`	-2506	Document is still active Available in Mac OS X v10.0 and later.
`tsmDocNotActiveErr`	-2507	Document is not active Available in Mac OS X v10.0 and later.
`tsmNoOpenTSErr`	-2508	There is no open text service component Available in Mac OS X v10.0 and later.
`tsmCantOpenComponentErr`	-2509	Can't open the component Available in Mac OS X v10.0 and later.
`tsmTextServiceNotFoundErr`	-2510	No text service component found Available in Mac OS X v10.0 and later.
`tsmDocumentOpenErr`	-2511	There are open documents Available in Mac OS X v10.0 and later.
`tsmUseInputWindowErr`	-2512	An input window is being used Available in Mac OS X v10.0 and later.

Result Code	Value	Description
tsmTSHasNoMenuErr	-2513	The text service component has no menu Available in Mac OS X v10.0 and later.
tsmTSNotOpenErr	-2514	Text service component is not open Available in Mac OS X v10.0 and later.
tsmComponentAlreadyOpenErr	-2515	Text service component already open for document Available in Mac OS X v10.0 and later.
tsmInputMethodIsOldErr	-2516	The default input method is old-style Available in Mac OS X v10.0 and later.
tsmScriptHasNoIMErr	-2517	Script has no (or old) input method Available in Mac OS X v10.0 and later.
tsmUnsupportedTypeErr	-2518	Unsupported interface type Available in Mac OS X v10.0 and later.
tsmUnknownErr	-2519	Any other error not listed in this table Available in Mac OS X v10.0 and later.
tsmDefaultIsNotInputMethodErr	-2524	Current input source is a keyboard layout resource Available in Mac OS X v10.0 and later.
tsmDocPropertyNotFoundErr	-2528	Requested TSM document property not found Available in Mac OS X v10.2 and later.
tsmDocPropertyBufferTooSmallErr	-2529	Buffer passed for property value is too small Available in Mac OS X v10.2 and later.
tsmCantChangeForcedClassStateErr	-2530	Enabled state of a TextService class has been forced and cannot be changed Available in Mac OS X v10.2 and later.

Text Utilities Reference

Framework:	CoreServices/CoreServices.h, Carbon/Carbon.h
Declared in	TextUtils.h StringCompare.h NumberFormatting.h TypeSelect.h

Overview

The Text Utilities provide you with an integrated collection of routines for performing a variety of operations on textual information, ranging from modifying the contents of a string, to sorting strings from different languages, to converting times, dates, and numbers from internal representations to formatted strings and back. These routines work in conjunction with QuickDraw text drawing routines to help you display and modify text in applications that are distributed to an international audience.

The Text Utilities functions are used for numerous text-handling tasks, including

- defining strings–including functions for allocating strings in the heap and for loading strings from resources
- comparing and sorting strings–including functions for testing whether two strings are equal and functions for finding the sorting relationship between two strings
- modifying the contents of strings–including routines for converting the case of characters, stripping diacritical marks, replacing substrings, and truncating strings
- finding breaks and boundaries in text–including routines for finding word and line breaks, and for finding different script runs in a line of text
- converting and formatting date and time strings–including routines that convert numeric and string representations of dates and times into record format, and routines that convert numeric and record representations of dates and times into strings
- converting and formatting numeric strings–including routines that convert string representations of numbers into numeric representations

Carbon supports the majority of Text Utilities. However, Apple recommends that you use the comparison and word breaking utilities supplied by Unicode Utilities instead.

A number of obsolete Text Utilities functions-such as those prefixed with iu or IU-are not supported.

Functions by Task

Comparing Strings for Equality

`EqualString` (page 1658) Deprecated in Mac OS X v10.4
: Compares two Pascal strings for equality, using the comparison rules of the Macintosh file system. (Deprecated. Use `CFStringCompare` instead.)

`IdenticalString` (page 1666) Deprecated in Mac OS X v10.4
: Compares two Pascal strings for equality, making use of the string comparison information from a resource that you specify as a parameter. (Deprecated. Use `CFStringCompare` instead.)

`IdenticalText` (page 1666) Deprecated in Mac OS X v10.4
: Compares two text strings for equality, making use of the string comparison information from a resource that you specify as a parameter. (Deprecated. Use `CFStringCompare` instead.)

Converting Between Integers and Strings

`NumToString` (page 1672) Deprecated in Mac OS X v10.4
: Converts a long integer value into a Pascal string. (Deprecated. Use `CFStringCreateWithFormat` instead.)

`StringToNum` (page 1682) Deprecated in Mac OS X v10.4
: Converts the Pascal string representation of a base-10 number into a long integer value. (Deprecated. Use `CFStringGetIntValue` instead.)

Converting Between Strings and Floating-Point Numbers

`ExtendedToString` (page 1659) Deprecated in Mac OS X v10.4
: Converts an internal floating-point representation of a number into a string that can be presented to the user, using a `NumFormatStringRec` structure to specify how the output number string is formatted (Deprecated. Use `CFNumberFormatterCreateNumberFromString` instead.)

`StringToExtended` (page 1679) Deprecated in Mac OS X v10.4
: Converts a string representation of a number into a floating-point number, using a `NumFormatStringRec` structure to specify how the input number string is formatted. (Deprecated. Use `CFNumberFormatterCreateStringWithNumber` instead.)

Converting Between C and Pascal Strings

`c2pstr` (page 1654)
: Converts a C string to a Pascal string. (Deprecated. You should store strings as Core Foundation CFStrings instead. See *CFString Reference*.)

`C2PStr` (page 1654)
: Converts a C string to a Pascal string. (Deprecated. You should store strings as Core Foundation CFStrings instead. See *CFString Reference*.)

P2CStr (page 1673)
Converts a Pascal string to a C string. (Deprecated. You should store strings as Core Foundation CFStrings instead. See *CFString Reference*.)

c2pstrcpy (page 1654) Deprecated in Mac OS X v10.4
Converts a C string to a Pascal string. (Deprecated. You should store strings as Core Foundation CFStrings instead. See *CFString Reference*.)

CopyCStringToPascal (page 1657) Deprecated in Mac OS X v10.4
Converts a C string to a Pascal string. (Deprecated. You should store strings as Core Foundation CFStrings instead. See *CFString Reference*.)

CopyPascalStringToC (page 1657) Deprecated in Mac OS X v10.4
Converts a Pascal String to a C string. (Deprecated. You should store strings as Core Foundation CFStrings instead. See *CFString Reference*.)

p2cstr (page 1673) Deprecated in Mac OS X v10.4 Deprecated in Mac OS X v10.4 Deprecated in Mac OS X v10.5
Converts a Pascal string to a C string. (Deprecated. You should store strings as Core Foundation CFStrings instead. See *CFString Reference*.)

p2cstrcpy (page 1674) Deprecated in Mac OS X v10.4
Converts a Pascal string to a C string. (Deprecated. You should store strings as Core Foundation CFStrings instead. See *CFString Reference*.)

Defining and Specifying Strings

GetIndString (page 1664) Deprecated in Mac OS X v10.4
Loads a string from a string list ('STR#') resource into memory, given the resource ID of the string list and the index of the individual string. (Deprecated. Use CFBundleCopyLocalizedString instead.)

GetString (page 1665) Deprecated in Mac OS X v10.4
Loads a string from a string ('STR') resource into memory. (Deprecated. Use CFBundleCopyLocalizedString instead.)

NewString (page 1671) Deprecated in Mac OS X v10.4
Allocates memory in the heap for a string, copies its contents, and produces a handle for the heap version of the string. (Deprecated. Use CFStringCreateCopy instead.)

SetString (page 1677) Deprecated in Mac OS X v10.4
Changes the contents of a string referenced by a string handle, replacing the previous contents by copying the specified string. (Deprecated. Use CFStringCreateWithPascalString and CFStringReplaceAll.)

Determining Sorting Order for Strings in Different Languages

LanguageOrder (page 1668) Deprecated in Mac OS X v10.4
Determines the order in which strings in two different languages should be sorted. (Deprecated. Use CFStringCompare or UCCompareText instead.)

ScriptOrder (page 1677) Deprecated in Mac OS X v10.4
Determines the order in which strings in two different scripts should be sorted. (Deprecated. Use CFStringCompare or UCCompareText instead.)

`StringOrder` (page 1678) Deprecated in Mac OS X v10.4
Compares two Pascal strings, taking into account the script system and language for each of the strings. (Deprecated. Use `CFStringCompare` or `UCCompareText` instead.)

`TextOrder` (page 1684) Deprecated in Mac OS X v10.4
Compares two text strings, taking into account the script and language for each of the strings. (Deprecated. Use `CFStringCompare` or `UCCompareText` instead.)

Determining Sorting Order for Strings in the Same Language

`CompareString` (page 1655) Deprecated in Mac OS X v10.4
Compares two Pascal strings, making use of the string comparison information from a resource that you specify as a parameter. (Deprecated. Use `CFStringCompare` or `UCCompareText` instead.)

`CompareText` (page 1656) Deprecated in Mac OS X v10.4
Compares two text strings, making use of the string comparison information from a resource that you specify as a parameter. (Deprecated. Use `CFStringCompare` or `UCCompareText` instead.)

`RelString` (page 1674) Deprecated in Mac OS X v10.4
Compares two Pascal strings using the string comparison rules of the Macintosh file system and returns a value that indicates the sorting order of the first string relative to the second string. (Deprecated. Use `CFStringCompare` or `UCCompareText` instead.)

`relstring` (page 1675) Deprecated in Mac OS X v10.4
Compares two strings. (Deprecated. Use `CFStringCompare` or `UCCompareText` instead.)

Modifying Characters and Diacritical Marks

`LowercaseText` (page 1669) Deprecated in Mac OS X v10.4
Converts any uppercase characters in a text string into their lowercase equivalents. (Deprecated. Use `CFStringLowercase` instead.)

`StripDiacritics` (page 1683) Deprecated in Mac OS X v10.4
Strips any diacritical marks from a text string. (Deprecated. Use `CFStringTransform` instead.)

`UppercaseStripDiacritics` (page 1688) Deprecated in Mac OS X v10.4
Converts any lowercase characters in a text string into their uppercase equivalents and strips any diacritical marks from the text. (Deprecated. Use `CFStringTransform` instead.)

`UppercaseText` (page 1689) Deprecated in Mac OS X v10.4
Converts any lowercase characters in a text string into their uppercase equivalents. (Deprecated. Use `CFStringUppercase` instead.)

`UpperString` (page 1690) Deprecated in Mac OS X v10.4
Converts any lowercase letters in a Pascal string to their uppercase equivalents, using the Macintosh file system rules. (Deprecated. Use `CFStringUppercase` instead.)

`upperstring` (page 1691) Deprecated in Mac OS X v10.4
Converts any lowercase letters in a Pascal string to their uppercase equivalents. (Deprecated. Use `CFStringUppercase` instead.)

Searching for and Replacing Strings

`Munger` (page 1669)
Searches text for a specified string pattern and replaces it with another string.

`ReplaceText` (page 1676) Deprecated in Mac OS X v10.4
Searches text on a character-by-character basis, replacing all instances of a string in that text with another string. (Deprecated. Use `CFStringReplace` instead.)

Using Number Format Specification Strings for International Number Formatting

`FormatRecToString` (page 1663) Deprecated in Mac OS X v10.4
Converts an internal representation of number formatting information into a number format specification string, which can be displayed and modified. (Deprecated. Use `CFNumberFormatterGetFormat` instead.)

`StringToFormatRec` (page 1680) Deprecated in Mac OS X v10.4
Creates a number format specification string structure from a number format specification string that you supply in a Pascal string. (Deprecated. Use `CFNumberFormatterSetFormat` instead.)

Working With Word, Script, and Line Boundaries

`FindScriptRun` (page 1660) Deprecated in Mac OS X v10.4
Finds the next block of subscript text within a script run. (Deprecated. There is no replacement function because this capability is no longer needed in Mac OS X.)

`FindWordBreaks` (page 1661) Deprecated in Mac OS X v10.4
Determines the beginning and ending boundaries of a word in a text string. (Deprecated. Use `UCFindTextBreak` instead.)

Working With Universal Procedure Pointers

`DisposeIndexToStringUPP` (page 1658) Deprecated in Mac OS X v10.4
Disposes of a universal procedure pointer to an index-to-string callback.

`InvokeIndexToStringUPP` (page 1667) Deprecated in Mac OS X v10.4
Call an index-to-string callback.

`NewIndexToStringUPP` (page 1671) Deprecated in Mac OS X v10.4
Creates a new universal procedure pointer (UPP) to an index-to-string callback.

Working With Type Select Records

`TypeSelectClear` (page 1685) Deprecated in Mac OS X v10.4
Clears the key list and resets the type select record. (Deprecated. Use `UCTypeSelectFlushSelectorData` instead.)

`TypeSelectCompare` (page 1686) Deprecated in Mac OS X v10.4
Compares a text buffer to the keystroke buffer. (Deprecated. Use `UCTypeSelectCompare` instead.)

`TypeSelectFindItem` (page 1686) Deprecated in Mac OS X v10.4
: Finds the closest match between a specified list of characters and the keystrokes stored in the type select record. (Deprecated. Use `UCTypeSelectFindItem` instead.)

`TypeSelectNewKey` (page 1687) Deprecated in Mac OS X v10.4
: Creates a new type select record. (Deprecated. Use `UCTypeSelectCreateSelector` instead.)

Functions

c2pstr

Converts a C string to a Pascal string. (Deprecated in Mac OS X v10.4. You should store strings as Core Foundation CFStrings instead. See *CFString Reference*.)

```
StringPtr c2pstr (
   char *aStr
);
```

Availability
Available in Mac OS X v10.4 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`TextUtils.h`

C2PStr

Converts a C string to a Pascal string. (Deprecated in Mac OS X v10.4. You should store strings as Core Foundation CFStrings instead. See *CFString Reference*.)

```
StringPtr C2PStr (
    Ptr cString
);
```

Availability
Available in Mac OS X v10.4 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`TextUtils.h`

c2pstrcpy

Converts a C string to a Pascal string. (Deprecated in Mac OS X v10.4. You should store strings as Core Foundation CFStrings instead. See *CFString Reference*.)

```
void c2pstrcpy (
   Str255 dst,
   const char *src
);
```

Parameters

dst

On output, the Pascal string.

src

The C string you want to convert.

Discussion

This function allows in-place conversion. That is, the `src` and `dst` parameters can point to the same memory location.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

SoftVDigX

Declared In

`TextUtils.h`

CompareString

Compares two Pascal strings, making use of the string comparison information from a resource that you specify as a parameter. (Deprecated in Mac OS X v10.4. Use `CFStringCompare` or `UCCompareText` instead.)

```
short CompareString (
    ConstStr255Param aStr,
    ConstStr255Param bStr,
    Handle itl2Handle
);
```

Parameters

aStr

One of the Pascal strings to be compared.

bStr

The other Pascal string to be compared.

itl2Handle

The handle to the string-manipulation resource that contains string comparison information. If the value of this parameter is `NULL`, `CompareString` makes use of the resource for the current script. The string-manipulation resource includes functions and tables for modifying string comparison and tables for case conversion and stripping of diacritical marks.

Return Value

Returns –1 if the first string is less than the second string, 0 if the first string is equal to the second string, and 1 if the first string is greater than the second string.

Discussion
This function takes both primary and secondary sorting orders into consideration and returns a value that indicates the sorting order of the first string relative to the second string.

Special Considerations

`CompareString` may move memory; your application should not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`StringCompare.h`

CompareText

Compares two text strings, making use of the string comparison information from a resource that you specify as a parameter. (Deprecated in Mac OS X v10.4. Use `CFStringCompare` or `UCCompareText` instead.)

```
short CompareText (
   const void *aPtr,
   const void *bPtr,
   short aLen,
   short bLen,
   Handle itl2Handle
);
```

Parameters

aPtr
A pointer to the first character of the first text string.

bPtr
A pointer to the first character of the second text string.

aLen
The length, in bytes, of the first text string.

bLen
The length, in bytes, of the second text string.

itl2Handle
A handle to a string-manipulation (`'itl2'`) resource that contains string comparison information. If the value of this parameter is `NULL`, `CompareText` makes use of the resource for the current script. The string-manipulation resource includes functions and tables for modifying string comparison and tables for case conversion and stripping of diacritical marks.

Return Value
Returns –1 if the first string is less than the second string, 0 if the first string is equal to the second string, and 1 if the first string is greater than the second string.

Discussion
This function takes both primary and secondary sorting orders into consideration and returns a value that indicates the sorting order of the first string relative to the second string.

Special Considerations

`CompareText` may move memory; your application should not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`StringCompare.h`

CopyCStringToPascal

Converts a C string to a Pascal string. (Deprecated in Mac OS X v10.4. You should store strings as Core Foundation CFStrings instead. See *CFString Reference*.)

```
void CopyCStringToPascal (
   const char *src,
   Str255 dst
);
```

Parameters

src

The C string you want to convert.

dst

On output, the Pascal string.

Discussion

This function allows in-place conversion. That is, the `src` and `dst` parameters can point to the same memory location.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Related Sample Code

BSDLLCTest

Declared In

`TextUtils.h`

CopyPascalStringToC

Converts a Pascal String to a C string. (Deprecated in Mac OS X v10.4. You should store strings as Core Foundation CFStrings instead. See *CFString Reference*.)

```
void CopyPascalStringToC (
   ConstStr255Param src,
   char *dst
);
```

Parameters

src

The Pascal string you want to convert.

dst

On output, the C string.

Discussion

This function allows in-place conversion. That is, the `src` and `dst` parameters can point to the same memory location.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`TextUtils.h`

DisposeIndexToStringUPP

Disposes of a universal procedure pointer to an index-to-string callback. (Deprecated in Mac OS X v10.4.)

```
void DisposeIndexToStringUPP (
   IndexToStringUPP userUPP
);
```

Parameters

userUPP

The universal procedure pointer.

Discussion

See the callback `IndexToStringProcPtr` (page 1691) for more information.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`TypeSelect.h`

EqualString

Compares two Pascal strings for equality, using the comparison rules of the Macintosh file system. (Deprecated in Mac OS X v10.4. Use `CFStringCompare` instead.)

```
Boolean EqualString (
   ConstStr255Param str1,
   ConstStr255Param str2,
   Boolean caseSensitive,
   Boolean diacSensitive
);
```

Parameters

str1

One of the Pascal strings to be compared.

str2

The other Pascal string to be compared.

caseSensitive

A flag that indicates how to handle case-sensitive information during the comparison. If the value of `caseSens` is `TRUE`, uppercase characters are distinguished from the corresponding lowercase characters. If it is `FALSE`, case information is ignored.

diacSensitive

A flag that indicates how to handle information about diacritical marks during the string comparison. If the value of `diacSens` is `TRUE`, characters with diacritical marks are distinguished from the corresponding characters without diacritical marks during the comparison. If it is `FALSE`, diacritical marks are ignored.

Return Value

`TRUE` if the two strings are equal and `FALSE` if they are not equal. If its value is `TRUE`, `EqualString` distinguishes uppercase characters from the corresponding lowercase characters. If its value is `FALSE`, `EqualString` ignores diacritical marks during the comparison.

Discussion

The comparison is a simple, character-by-character value comparison. This function does not make use of any script or language information (i.e., is not localizable); it assumes the use of a Roman script system.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`StringCompare.h`

ExtendedToString

Converts an internal floating-point representation of a number into a string that can be presented to the user, using a `NumFormatStringRec` structure to specify how the output number string is formatted (Deprecated in Mac OS X v10.4. Use `CFNumberFormatterCreateNumberFromString` instead.)

```
FormatStatus ExtendedToString (
   const extended80 *x,
   const NumFormatString *myCanonical,
   const NumberParts *partsTable,
   Str255 outString
);
```

Parameters

x

A pointer to a floating-point value in 80-bit SANE representation.

myCanonical

A pointer to the internal representation of the formatting information for numbers, as produced by the `StringToFormatRec` function.

partsTable

A pointer to a structure, obtained from the tokens (`'itl4'`) resource, that shows the correspondence between generic number part separators (tokens) and their localized version (for example, a thousand separator is a comma in the United States and a decimal point in France).

outString

On output, contains the number formatted according to the information in `myFormatRec`.

Return Value

A value that denotes the confidence level for the conversion that it performed. The low byte of the `FormatStatus` value is of type `FormatResultType`. Be sure to cast the result of `ExtendedToString` to a type `FormatResultType` before working with it. See the description of the `FormatStatus` data type.

Discussion

`ExtendedToString` creates a string representation of a floating-point number, using the formatting information in the `myFormatRec` parameter (which was created by a previous call to `StringToFormatRec`) to determine how the number should be formatted for output. It uses the number parts table to determine the component parts of the number string.

To obtain a handle to the number parts table from a tokens resource, use the `GetIntlResourceTable` function.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`NumberFormatting.h`

FindScriptRun

Finds the next block of subscript text within a script run. (Deprecated in Mac OS X v10.4. There is no replacement function because this capability is no longer needed in Mac OS X.)

```
ScriptRunStatus FindScriptRun (
   Ptr textPtr,
   long textLen,
   long *lenUsed
);
```

Parameters

textPtr

A pointer to the text string to be analyzed.

textLen

The number of bytes in the text string.

lenUsed

On output, a pointer to the length, in bytes, of the script run that begins with the first character in the string; this length is always greater than or equal to 1, unless the string passed in is of length 0.

Return Value

Identifies the run as either native text, Roman, or one of the defined subscripts of the script system and returns a structure of type `ScriptRunStatus` (page 1698). See the description of the `ScriptRunStatus` data type.

Discussion

The `FindScriptRun` function is used to identify blocks of subscript text in a string, taking into account script and language considerations, making use of tables in the string-manipulation (`'itl2'`) resource in its computations. Some script systems include subscripts, which are character sets that are subsidiary to the main character set. One useful subscript is the set of all character codes that have the same meaning in Roman as they do in a non-Roman script. For other scripts such as Japanese, there are additional useful subscripts. For example, a Japanese script system might include some Hiragana characters that are useful for input methods.

`FindScriptRun` computes the length of the current run of subscript text in the text string specified by `textPtr` and `textLen`. It assigns the length, in bytes, to the `lenUsed` parameter and returns a status code. You can advance the text pointer by the value of `lenUsed` to make subsequent calls to this function. You can use this function to identify runs of subscript characters so that you can treat them separately.

Word processors and other applications can call `FindScriptRun` to separate style runs of native text from non-native text. You can use this capability to extract those characters and apply a different font to them.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`TextUtils.h`

FindWordBreaks

Determines the beginning and ending boundaries of a word in a text string. (Deprecated in Mac OS X v10.4. Use `UCFindTextBreak` instead.)

```
void FindWordBreaks (
   Ptr textPtr,
   short textLength,
   short offset,
   Boolean leadingEdge,
   BreakTablePtr breaks,
   OffsetTable offsets,
   ScriptCode script
);
```

Parameters

textPtr

A pointer to the text string to be examined.

textLength

The number of bytes in the text string.

offset

A byte offset into the text. This parameter plus the `leadingEdge` parameter determine the position of the character at which to start the search.

leadingEdge

A flag that specifies which character should be used to start the search. If `leadingEdge` is `TRUE`, the search starts with the character specified in the `offset` parameter; if it is `FALSE`, the search starts with the character preceding the offset.

breaks

A pointer to a word-break table of type `NBreakTable` or `BreakTable`. If the value of this pointer is 0, the default word-break table of the script system specified by the `script` parameter is used. If the value of this pointer is –1, the default line-break table of the specified script system is used.

offsets

On output, the values in this table indicate the boundaries of the word that has been found.

script

The script code for the script system whose tables are used to determine where word boundaries occur.

Discussion

`FindWordBreaks` searches for a word in a text string, taking into account script and language considerations, making use of tables in the string-manipulation (`'itl2'`) resource in its computations. The `textPtr` and `textLength` parameters specify the text string that you want searched. The `offset` parameter and `leadingEdge` parameter together indicate where the search begins.

`FindWordBreaks` searches backward through the text string for one of the word boundaries and forward through the text string for its other boundary. It uses the definitions in the table specified by `nbreaks` to determine what constitutes the boundaries of a word. Each script system's word-break table is part of its string-manipulation (`'itl2'`) resource.

`FindWordBreaks` returns its results in an `OffsetTable` structure. `FindWordBreaks` uses only the first element of this three-element table. Each element is a pair of integers: `offFirst` and `offSecond`.

`FindWordBreaks` places the offset from the beginning of the text string to just before the leading edge of the character of the word that it finds in the `offFirst` field.

`FindWordBreaks` places the offset from the beginning of the text string to just after the trailing edge of the last character of the word that it finds in the `offSecond` field. For example, if the text "This is it" is passed with `offset` set to 0 and `leadingEdge` set to `TRUE`, then `FindWordBreaks` returns the offset pair (0,4).

If `leadingEdge` is `TRUE` and the value of `offset` is 0, then `FindWordBreaks` returns the offset pair (0,0). If `leadingEdge` is `FALSE` and the value of `offset` equals the value of `textLength`, then `FindWordBreaks` returns the offset pair with values (`textLength`, `textLength`).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextUtils.h`

FormatRecToString

Converts an internal representation of number formatting information into a number format specification string, which can be displayed and modified. (Deprecated in Mac OS X v10.4. Use `CFNumberFormatterGetFormat` instead.)

```
FormatStatus FormatRecToString (
   const NumFormatString *myCanonical,
   const NumberParts *partsTable,
   Str255 outString,
   TripleInt positions
);
```

Parameters

myCanonical
A pointer to the internal representation of number formatting information, as created by a previous call to the `StringToFormatRec` function.

partsTable
A pointer to a structure, obtained from the tokens (`'itl4'`) resource, that shows the correspondence between generic number part separators (tokens) and their localized version (for example, a thousand separator is a comma in the United States and a decimal point in France).

outString
On output, contains the number format specification string.

positions
An array that specifies the starting position and length of each of the three possible format strings (positive, negative, or zero) in the number format specification string. Semicolons are used as separators in the string.

Return Value
A value that denotes the confidence level for the conversion that it performed. The low byte of the `FormatStatus` value is of type `FormatResultType`. Be sure to cast the result of `FormatRecToString` to a type `FormatResultType` before working with it. See the description of the `FormatStatus` data type.

Discussion
`FormatRecToString` is the inverse operation of `StringToFormatRec` (page 1680). The internal representation of the formatting information in `myFormatRec` must have been created by a prior call to the `StringToFormatRec` function. The information in the number parts table specifies how to build the string representation.

The output number format specification string in `outString` specifies how numbers appear. This string contains three parts, which are separated by semicolons. The first part is the positive number format, the second is the negative number format, and the third part is the zero number format.

The `positions` parameter is an array of three integers (a `TripleInt` value), which specifies the starting position in `outString` of each of three formatting specifications:

- `positions[fPositive]`. The index in `outString` of the first byte of the formatting specification for positive number values.
- `positions[fNegative]`. The index in `outString` of the first byte of the formatting specification for negative number values.
- `positions[fZero]`. The index in `outString` of the first byte of the formatting specification for zero number values.

To obtain a handle to the number parts table from a tokens resource, use the `GetIntlResourceTable` function.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`NumberFormatting.h`

GetIndString

Loads a string from a string list (`'STR#'`) resource into memory, given the resource ID of the string list and the index of the individual string. (Deprecated in Mac OS X v10.4. Use `CFBundleCopyLocalizedString` instead.)

```
void GetIndString (
   Str255 theString,
   short strListID,
   short index
);
```

Parameters

`theString`
On output, the Pascal string result; specifically, a copy of the string from a string list that has the resource ID provided in the `strListID` parameter. If the resource that you specify cannot be read or the index that you specify is out of range for the string list, `GetIndString` sets `theString` to an empty string.

`strListID`
The resource ID of the `'STR#'` resource that contains the string list.

`index`
The index of the string in the list. This is a value from 1 to the number of strings in the list that is referenced by the `strListID` parameter.

Discussion
If necessary, `GetIndString` reads the string list from the resource file by calling the Resource Manager function `GetResource`. `GetIndString` accesses the string specified by the `index` parameter and copies it into `theString`.

Special Considerations

`GetIndString` may move memory; your application should not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextUtils.h`

GetString

Loads a string from a string (`'STR'`) resource into memory. (Deprecated in Mac OS X v10.4. Use `CFBundleCopyLocalizedString` instead.)

```
StringHandle GetString (
   short stringID
);
```

Parameters

`stringID`
The resource ID of the string (`'STR '`) resource containing the string.

Return Value
A handle to a string with the specified resource ID. If necessary, `GetString` reads the handle form the resource file. If `GetString` cannot read the resource, it returns `NULL`.

Discussion
`GetString` calls the `GetResource` function of the Resource Manager to access the string. This means that if the specified resource is already in memory, `GetString` simply returns its handle.

Like the `NewString` (page 1671) function, `GetString` returns a handle whose size is based upon the actual length of the string.

If your application uses a large number of strings, it is more efficient to store them in a string list (`'STR#'`) resource than as individual resources in the resource file. You then use the `GetIndString` (page 1664) function to access each string in the list.

Special Considerations

`GetString` does not create a copy of the string.

`GetString` may move memory; your application should not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextUtils.h`

IdenticalString

Compares two Pascal strings for equality, making use of the string comparison information from a resource that you specify as a parameter. (Deprecated in Mac OS X v10.4. Use `CFStringCompare` instead.)

```
short IdenticalString (
   ConstStr255Param aStr,
   ConstStr255Param bStr,
   Handle itl2Handle
);
```

Parameters

aStr

One of the Pascal strings to be compared.

bStr

The other Pascal string to be compared.

itl2Handle

A handle to a string-manipulation (`'itl2'`) resource that contains string comparison information.

The `itl2Handle` parameter is used to specify a string-manipulation resource. If the value of this parameter is `NULL`, `IdenticalString` makes use of the resource for the current script. The string-manipulation resource includes tables for modifying string comparison and tables for case conversion and stripping of diacritical marks.

Return Value
Returns 0 if the two strings are equal; 1 if they are not equal. It compares the two strings without regard for secondary sorting order, the meaning of which depends on the language of the strings. For example, for the English language, using only primary differences means that `IdenticalString` ignores diacritical marks and does not distinguish between lowercase and uppercase. For example, if the two strings are `'Rose'` and `'rosé'`, `IdenticalString` considers them equal and returns 0.

Discussion
`IdenticalString` uses only primary differences in its comparison.

Special Considerations

`IdenticalString` may move memory; your application should not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`StringCompare.h`

IdenticalText

Compares two text strings for equality, making use of the string comparison information from a resource that you specify as a parameter. (Deprecated in Mac OS X v10.4. Use `CFStringCompare` instead.)

```
short IdenticalText (
   const void *aPtr,
   const void *bPtr,
   short aLen,
   short bLen,
   Handle itl2Handle
);
```

Parameters

aPtr

A pointer to the first character of the first text string.

bPtr

A pointer to the first character of the second text string.

aLen

The length, in bytes, of the first text string.

bLen

The length, in bytes, of the second text string.

itl2Handle

A handle to a string-manipulation (`'itl2'`) resource that contains string comparison information.

The `itl2Handle` parameter is used to specify a string-manipulation resource. If the value of this parameter is `NULL`, `IdenticalText` makes use of the resource for the current script. The string-manipulation resource includes functions and tables for modifying string comparison and tables for case conversion and stripping of diacritical marks.

Return Value

0 if the two text strings are equal; 1 if they are not equal. It compares the strings without regard for secondary sorting order, which means that it ignores diacritical marks and does not distinguish between lowercase and uppercase. For example, if the two text strings are `'Rose'` and `'rosé'`, `IdenticalText` considers them equal and returns 0.

Discussion

`IdenticalText` uses only primary sorting order in its comparison.

Special Considerations

`IdenticalText` may move memory; your application should not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`StringCompare.h`

InvokeIndexToStringUPP

Call an index-to-string callback. (Deprecated in Mac OS X v10.4.)

```
Boolean InvokeIndexToStringUPP (
   short item,
   ScriptCode *itemsScript,
   StringPtr *itemsStringPtr,
   void *yourDataPtr,
   IndexToStringUPP userUPP
);
```

Discussion
You should not need to use the function `InvokeIndexToStringUPP`, as the system calls your index-to-string callback function for you. See the callback `IndexToStringProcPtr` (page 1691) for more information.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TypeSelect.h`

LanguageOrder

Determines the order in which strings in two different languages should be sorted. (Deprecated in Mac OS X v10.4. Use `CFStringCompare` or `UCCompareText` instead.)

```
short LanguageOrder (
   LangCode language1,
   LangCode language2
);
```

Parameters

language1
The language code of the first language.

language2
The language code of the second language.

Return Value
A value that indicates the sorting order: –1 if strings in the first language should be sorted before sorting text in the second language, 1 if strings in the first language should be sorted after sorting strings in the second language, or 0 if the sorting order does not matter (that is, if the languages are the same).

Discussion
`LanguageOrder` takes a pair of language codes and determines in which order strings from the first language should be sorted relative to strings from the second language.

"Implicit Language Codes" (page 1702) are listed in the Constants section. The implicit language codes `scriptCurLang` and `scriptDefLang` are not valid for `LanguageOrder` because the script system being used is not specified as a parameter to this function.

Special Considerations

`LanguageOrder` may move memory; your application should not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`StringCompare.h`

LowercaseText

Converts any uppercase characters in a text string into their lowercase equivalents. (Deprecated in Mac OS X v10.4. Use `CFStringLowercase` instead.)

```
void LowercaseText (
   Ptr textPtr,
   short len,
   ScriptCode script
);
```

Parameters

`textPtr`

A pointer to the text string to be converted.

`len`

The number of bytes in the text string. The text string can be up to 32 KB in length.

`script`

The script code for the script system whose resources are used to determine the results of converting characters.

The conversion uses tables in the string-manipulation (`'itl2'`) resource of the script specified by the value of the `script` parameter. You can specify `smSystemScript` to use the system script and `smCurrentScript` to use the script of the current font in the current graphics port.

Discussion
`LowercaseText` traverses the characters starting at the address specified by `textPtr` and continues for the number of characters specified in `len`. It converts any uppercase characters in the text into lowercase.

If `LowercaseText` cannot access the specified resource, it generates an error code and does not modify the string. You need to call the `ResError` function to determine which, if any, error occurred.

Special Considerations

`LowercaseText` may move memory; your application should not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextUtils.h`

Munger

Searches text for a specified string pattern and replaces it with another string.

```
long Munger (
   Handle h,
   long offset,
   const void *ptr1,
   long len1,
   const void *ptr2,
   long len2
);
```

Parameters

h

A handle to the text string that is being manipulated.

offset

The byte offset in the destination string at which `Munger` begins its operation.

ptr1

A pointer to the first character in the string for which `Munger` is searching.

len1

The number of bytes in the string for which `Munger` is searching.

ptr2

A pointer to the first character in the substitution string.

len2

The number of bytes in the substitution string.

Return Value

A negative value if `Munger` cannot find the designated string.

Discussion

`Munger` manipulates bytes in a string to which you specify a handle in the `h` parameter. The manipulation begins at a byte offset, specified in `offset`, in the string. `Munger` searches for the string specified by `ptr1` and `len1`; when it finds an instance of that string, it replaces it with the substitution string, which is specified by `ptr2` and `len2`.

`Munger` operates on a byte-by-byte basis, which can produce inappropriate results for 2-byte script systems. The `ReplaceText` (page 1676) function works properly for all languages. You are encouraged to use `ReplaceText` instead of `Munger` whenever possible.

`Munger` takes special action if either of the specified pointer values is `NULL` or if either of the length values is `0`.

- If `ptr1` is `NULL`, `Munger` replaces characters without searching. It replaces `len1` characters starting at the `offset` location with the substitution string.
- If `ptr1` is `NULL` and `len1` is negative, `Munger` replaces all of the characters from the `offset` location to the end of the string with the substitution string.
- If `len1` is 0, `Munger` inserts the substitution string without replacing anything. `Munger` inserts the string at the `offset` location and returns the offset of the first byte past where the insertion occurred.
- If `ptr2` is `NULL`, `Munger` searches but does not replace. In this case, `Munger` returns the offset at which the string was found.
- If `len2` is 0 and `ptr2` is not `NULL`, `Munger` searches and deletes. In this case, `Munger` returns the offset at which it deleted.

- If the portion of the string from the `offset` location to its end matches the beginning of the string that `Munger` is searching for, `Munger` replaces that portion with the substitution string.

Be careful not to specify an offset with a value that is greater than the length of the destination string. Unpredictable results may occur.

`Munger` calls the `GetHandleSize` and `SetHandleSize` functions to access or modify the length of the string it is manipulating.

Special Considerations

`Munger` may move memory; your application should not call this function at interrupt time.

The destination string must be in a relocatable block that was allocated by the Memory Manager.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TextUtils.h`

NewIndexToStringUPP

Creates a new universal procedure pointer (UPP) to an index-to-string callback. (Deprecated in Mac OS X v10.4.)

```
IndexToStringUPP NewIndexToStringUPP (
   IndexToStringProcPtr userRoutine
);
```

Parameters

`userRoutine`
A pointer to your index-to-string callback.

Return Value
On return, a UPP to the index-to-string callback.

Discussion
See the callback `IndexToStringProcPtr` (page 1691) for more information.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TypeSelect.h`

NewString

Allocates memory in the heap for a string, copies its contents, and produces a handle for the heap version of the string. (Deprecated in Mac OS X v10.4. Use `CFStringCreateCopy` instead.)

```
StringHandle NewString (
   ConstStr255Param theString
);
```

Parameters

theString

A Pascal string that you want copied onto the heap.

Return Value

A handle to the newly allocated string. If the string cannot be allocated, `NewString` returns `NULL`. The size of the allocated string is based on the actual length of `theString`, which may not be 255 bytes.

Discussion

Before using Pascal string functions that can change the length of the string, it is a good idea to maximize the size of the string object on the heap. You can call either the `SetString` (page 1677) function or the Memory Manager function `SetHandleSize` to modify the string's size.

Special Considerations

`NewString` may move memory; your application should not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`TextUtils.h`

NumToString

Converts a long integer value into a Pascal string. (Deprecated in Mac OS X v10.4. Use `CFStringCreateWithFormat` instead.)

```
void NumToString (
   long theNum,
   Str255 theString
);
```

Parameters

theNum

A long integer value. If the value of the number in the parameter `theNum` is negative, the string begins with a minus sign; otherwise, the sign is omitted.

theString

On output, contains the Pascal string representation of the number. Leading zeros are suppressed, except that a value of 0 produces the string "0". `NumToString` does not include thousand separators or decimal points in its formatted output.

Discussion

`NumToString` creates a string representation of `theNum` as a base-10 value and returns the result in `theString`.

Unless patched by a script system with different rules, this function assumes that you are using standard numeric token processing, meaning that the Roman script system number processing rules are used.

For functions that make use of the token-processing information that is found in the tokens (`'itl4'`) resource of script systems for converting numbers, see the sections "Using Number Format Specification Strings for International Number Formatting" and "Converting Between Strings and Floating-Point Numbers".

Special Considerations

`NumToString` may move memory; your application should not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`NumberFormatting.h`

p2cstr

Converts a Pascal string to a C string. (Deprecated in Mac OS X v10.4. You should store strings as Core Foundation CFStrings instead. See *CFString Reference*.)

```
char * p2cstr (
   StringPtr aStr
);
```

Availability
Available in Mac OS X v10.4 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextUtils.h`

P2CStr

Converts a Pascal string to a C string. (Deprecated in Mac OS X v10.4. You should store strings as Core Foundation CFStrings instead. See *CFString Reference*.)

```
Ptr P2CStr (
    StringPtr pString
);
```

Availability
Available in Mac OS X v10.4 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextUtils.h`

p2cstrcpy

Converts a Pascal string to a C string. (Deprecated in Mac OS X v10.4. You should store strings as Core Foundation CFStrings instead. See *CFString Reference*.)

```
void p2cstrcpy (
   char *dst,
   ConstStr255Param src
);
```

Parameters

dst

On output, the C string.

src

The Pascal string you want to convert.

Discussion

This function allows in-place conversion. That is, the `src` and `dst` parameters can point to the same memory location.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`TextUtils.h`

RelString

Compares two Pascal strings using the string comparison rules of the Macintosh file system and returns a value that indicates the sorting order of the first string relative to the second string. (Deprecated in Mac OS X v10.4. Use `CFStringCompare` or `UCCompareText` instead.)

```
short RelString (
   ConstStr255Param str1,
   ConstStr255Param str2,
   Boolean caseSensitive,
   Boolean diacSensitive
);
```

Parameters

str1

One of the Pascal strings to be compared.

str2

The other Pascal string to be compared.

caseSensitive

A flag that indicates how to handle case-sensitive information during the comparison. If the value of `caseSens` is `TRUE`, uppercase characters are distinguished from the corresponding lowercase characters. If it is `FALSE`, case information is ignored.

diacSensitive
A flag that indicates how to handle information about diacritical marks during the string comparison. If the value of `diacSensitive` is `TRUE`, characters with diacritical marks are distinguished from the corresponding characters without diacritical marks during the comparison. If it is `FALSE`, diacritical marks are ignored.

Return Value
Returns –1 if the first string is less than the second string, 0 if the two strings are equal, and 1 if the first string is greater than the second string. It compares the two strings in the same manner as does the `EqualString` function, by simply looking at the ASCII values of their characters. However, `RelString` provides more information about the two strings—it indicates their relationship to each other, rather than determining if they are exactly equal.

Discussion
This function does not make use of any script or language information; it assumes the original Macintosh character set only.

Special Considerations

The `RelString` function is not localizable and does not work properly with non-Roman script systems.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`StringCompare.h`

relstring

Compares two strings. (Deprecated in Mac OS X v10.4. Use `CFStringCompare` or `UCCompareText` instead.)

Not recommended

```
short relstring (
   const char *str1,
   const char *str2,
   Boolean caseSensitive,
   Boolean diacSensitive
);
```

Parameters
str1
The string to be compared to `str2`.
str2
The string to be compared to `str1`.
caseSensitive
A flag that indicates how to handle case-sensitive information during the comparison.
diacSensitive
A flag that indicates how to handle information about diacritical marks during the string comparison.

Return Value
Returns –1 if the first string is less than the second string, 0 if the two strings are equal, and 1 if the first string is greater than the second string.

Discussion
This function is not recommended. Instead, see the function `RelString` (page 1674).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`StringCompare.h`

ReplaceText

Searches text on a character-by-character basis, replacing all instances of a string in that text with another string. (Deprecated in Mac OS X v10.4. Use `CFStringReplace` instead.)

```
short ReplaceText (
    Handle baseText,
    Handle substitutionText,
    Str15 key
);
```

Parameters

baseText
A handle to the string in which `ReplaceText` is to substitute text.

substitutionText
A handle to the string that `ReplaceText` uses as substitute text.

key
A Pascal string of less than 16 bytes that `ReplaceText` searches for.

Return Value
An integer value; if positive, it indicates the number of substitutions performed; if negative, it indicates an error. The constant `noErr` is returned if there was no error and no substitutions were performed.

Discussion
`ReplaceText` searches the text specified by the `baseText` parameter for instances of the string in the `key` parameter and replaces each instance with the text specified by the `substitutionText` parameter. `ReplaceText` searches on a character-by-character basis (as opposed to byte-by-byte), so it works properly for all script systems, including 2-byte script systems. It recognizes 2-byte characters in script systems that contain them and advances the search appropriately after encountering a 2-byte character.

Special Considerations

`ReplaceText` may move memory; your application should not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`StringCompare.h`

ScriptOrder

Determines the order in which strings in two different scripts should be sorted. (Deprecated in Mac OS X v10.4. Use `CFStringCompare` or `UCCompareText` instead.)

```
short ScriptOrder (
   ScriptCode script1,
   ScriptCode script2
);
```

Parameters

script1
The script code of the first script.

script2
The script code of the second script.

Return Value
A value that indicates the sorting order: –1 if strings in the first script should be sorted before strings in the second script are sorted, 1 if strings in the first script should be sorted after strings in the second script are sorted, or 0 if the sorting order does not matter (that is, if the scripts are the same).

Discussion
Text of the system script is always first in a sorted list, regardless of the result returned by this function. When determining the order in which text from two different script systems should be sorted, the system script always sorts first, and scripts that are not enabled and installed always sort last. Invalid script or language codes always sort after valid ones.

Special Considerations

`ScriptOrder` may move memory; your application should not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`StringCompare.h`

SetString

Changes the contents of a string referenced by a string handle, replacing the previous contents by copying the specified string. (Deprecated in Mac OS X v10.4. Use `CFStringCreateWithPascalString` and `CFStringReplaceAll`.)

```
void SetString (
   StringHandle theString,
   ConstStr255Param strNew
);
```

Parameters

theString

A Pascal string.

strNew

A handle to the string in memory whose contents you are replacing. If the new string (`theString`) is larger than the string originally referenced by `strNew`, `SetString` automatically resizes the handle and copies in the contents of the specified string.

Special Considerations

`SetString` may move memory; your application should not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`TextUtils.h`

StringOrder

Compares two Pascal strings, taking into account the script system and language for each of the strings. (Deprecated in Mac OS X v10.4. Use `CFStringCompare` or `UCCompareText` instead.)

```
short StringOrder (
   ConstStr255Param aStr,
   ConstStr255Param bStr,
   ScriptCode aScript,
   ScriptCode bScript,
   LangCode aLang,
   LangCode bLang
);
```

Parameters

aStr

One of the Pascal strings to be compared.

bStr

The other Pascal string to be compared.

aScript

The script code for the second string.

bScript

The script code for the first string.

aLang

The language code for the first string.

bLang

The language code for the second string.

Return Value
–1 if the first string is less than the second string, 0 if the first string is equal to the second string, and 1 if the first string is greater than the second string. The ordering of script and language codes, which is based on information in the script-sorting resource, is considered in determining the relationship of the two strings.

Discussion
This function takes both primary and secondary sorting orders into consideration and returns a value that indicates whether the first string is less than, equal to, or greater than the second string.

"Implicit Language Codes" (page 1702) are listed in the Constants section. Most applications specify the language code `scriptCurLang` for both the `aLang` and `bLang` values.

`StringOrder` first calls `ScriptOrder` (page 1677); if the result of `ScriptOrder` is not 0 (that is, if the strings use different scripts), `StringOrder` returns the same result.

`StringOrder` next calls `LanguageOrder` (page 1668); if the result of `LanguageOrder` is not 0 (that is, if the strings use different languages), `StringOrder` returns the same result.

At this point, `StringOrder` has two strings that are in the same script and language, so it compares them by using the sorting rules for that script and language, applying both the primary and secondary sorting orders. If that script is not installed and enabled, it uses the sorting rules specified by the system script or the font script, depending on the state of the international resources selection flag.

The `StringOrder` function is primarily used to insert Pascal strings in a sorted list; for sorting, rather than using this function, it may be faster to sort first by script and language by using the `ScriptOrder` and `LanguageOrder` functions, and then to call the `CompareString` (page 1655) function, to sort strings within a script or language group.

Special Considerations

`StringOrder` may move memory; your application should not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`StringCompare.h`

StringToExtended

Converts a string representation of a number into a floating-point number, using a `NumFormatStringRec` structure to specify how the input number string is formatted. (Deprecated in Mac OS X v10.4. Use `CFNumberFormatterCreateStringWithNumber` instead.)

```
FormatStatus StringToExtended (
   ConstStr255Param source,
   const NumFormatString *myCanonical,
   const NumberParts *partsTable,
   extended80 *x
);
```

Parameters

source

A Pascal string that contains the string representation of a number.

myCanonical

A pointer to the internal representation of the formatting information for numbers, as produced by the `StringToFormatRec` function.

partsTable

A pointer to a structure, obtained from the tokens (`'itl4'`) resource, that shows the correspondence between generic number part separators (tokens) and their localized version (for example, a thousand separator is a comma in the United States and a decimal point in France).

x

On output, contains a pointer to the 80-bit SANE representation of the floating-point number.

Return Value

A value that denotes the confidence level for the conversion that it performed. The low byte of the `FormatStatus` value is of type `FormatResultType`. Be sure to cast the result of `StringToExtended` to a type `FormatResultType` before working with it. `StringToExtended` returns an 80-bit, not a 96-bit, representation. See the description of the `FormatStatus` data type.

Discussion

`StringToExtended` uses the internal representation of number formatting information that was created by a prior call to `StringToFormatRec` to parse the input number string. It uses the number parts table to determine the components of the number string that is being converted. `StringToExtended` parses the string and then converts the string to a simple form, stripping nondigits and replacing the decimal point before converting it into a floating-point number. If the input string does not match any of the patterns, then `StringToExtended` parses the string as well as it can and returns a confidence level result that indicates the parsing difficulties.

To obtain a handle to the number parts table from a tokens resource, use the `GetIntlResourceTable` function.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`NumberFormatting.h`

StringToFormatRec

Creates a number format specification string structure from a number format specification string that you supply in a Pascal string. (Deprecated in Mac OS X v10.4. Use `CFNumberFormatterSetFormat` instead.)

```
FormatStatus StringToFormatRec (
   ConstStr255Param inString,
   const NumberParts *partsTable,
   NumFormatString *outString
);
```

Parameters

inString

A Pascal string that contains the number formatting specification.

The `inString` parameter contains a number format specification string that specifies how numbers appear. This string contains up to three specifications, separated by semicolons. The positive number format is specified first, the negative number format is second, and the zero number format is last. If the string contains only one part, that is the format of all three types of numbers. If the string contains two parts, the first part is the format for positive and zero number values, and the second part is the format for negative numbers.

partsTable

A pointer to a structure, usually obtained from the tokens (`'itl4'`) resource, that shows the correspondence between generic number part separators (tokens) and their localized version (for example, a thousand separator is a comma in the United States and a decimal point in France).

outString

On output, a pointer to a `NumFormatStringRec` structure that contains the values that form the internal representation of the format specification. The format of the data in this structure is private.

Return Value

A value that denotes the confidence level for the conversion that was performed. The low byte of the value is of type `FormatResultType`. Be sure to cast the result of `StringToFormatRec` to a type `FormatResultType` before working with it. See the description of the `FormatStatus` data type.

Discussion

`StringToFormatRec` converts a number format specification string into the internal representation contained in a number format string structure. It uses information in the current script's tokens resource to determine the components of the number. `StringToFormatRec` checks the validity both of the input format string and of the number parts table (since this table can be programmed by the application). `StringToFormatRec` ignores spurious characters.

To obtain a handle to the number parts table from a tokens resource, use the `GetIntlResourceTable` function.

Special Considerations

`StringToFormatRec` may move memory; your application should not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`NumberFormatting.h`

StringToNum

Converts the Pascal string representation of a base-10 number into a long integer value. (Deprecated in Mac OS X v10.4. Use `CFStringGetIntValue` instead.)

```
void StringToNum (
   ConstStr255Param theString,
   long *theNum
);
```

Parameters

theString

A Pascal string representation of a base-10 number. The numeric string can be padded with leading zeros or with a sign.

theNum

On output, contains a pointer to the numeric value.

Discussion

Unless patched by a script system with different rules, this function assumes that you are using standard numeric token processing, meaning that the Roman script system number processing rules are used.

For functions that make use of the token-processing information that is found in the tokens (`'itl4'`) resource of script systems for converting numbers, see the sections "Using Number Format Specification Strings for International Number Formatting" and "Converting Between Strings and Floating-Point Numbers".

The 32-bit result is negated if the string begins with a minus sign. Integer overflow occurs if the magnitude is greater than or equal to 2 raised to the 31st power. `StringToNum` performs the negation using the two's complement method: the state of each bit is reversed and then 1 is added to the result. For example, here are possible results produced by `StringToNum`:

- The value of `theString` is "-23". `StringToNum` returns the value -23 in `theNum`.
- The value of `theString` is "-0". `StringToNum` returns the value 0 in `theNum`.
- The value of `theString` is "055". `StringToNum` returns the value 55 in `theNum`.
- The value of `theString` is "2147483648" (magnitude is 2^31). `StringToNum` returns the value –2147483648 in `theNum`.
- The value of `theString` is "–2147483648". `StringToNum` returns the value –2147483648 in `theNum`.
- The value of `theString` is "4294967295" (magnitude is 2^32–1). `StringToNum` returns the value -1 in `theNum`.
- The value of `theString` is "–4294967295". `StringToNum` returns the value 1 in `theNum`.

`StringToNum` does not check whether the characters in the string are between 0 and 9; instead, it takes advantage of the fact that the ASCII values for these characters are $30 through $39, and masks the last four bits for use as a digit. For example, `StringToNum` converts 2: to the number 30 since the character code for the colon (:) is $3A. Because `StringToNum` operates this way, spaces are treated as zeros (the character code for a space is $20), and other characters do get converted into numbers. For example, the character codes for 'C', 'A', and 'T' are $43, $41, and $54 respectively. Hence, the strings 'CAT', '+CAT', and '-CAT' would produce the results 314, 314, and -314.

One consequence of this conversion method is that `StringToNum` does not ignore thousand separators (the "," character in the United States), which can lead to improper conversions. It is a good idea to ensure that all characters in `theString` are valid digits before you call `StringToNum`.

Special Considerations

`StringToNum` may move memory; your application should not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`NumberFormatting.h`

StripDiacritics

Strips any diacritical marks from a text string. (Deprecated in Mac OS X v10.4. Use `CFStringTransform` instead.)

```
void StripDiacritics (
   Ptr textPtr,
   short len,
   ScriptCode script
);
```

Parameters

`textPtr`

A pointer to the text string to be stripped.

`len`

The length, in bytes, of the text string. The text string can be up to 32 KB in length.

`script`

The script code for the script system whose rules are used for determining which character results from stripping a diacritical mark.

The conversion uses tables in the string-manipulation (`'itl2'`) resource of the script specified by the value of the `script` parameter. You can specify `smSystemScript` to use the system script and `smCurrentScript` to use the script of the current font in the current graphics port.

Discussion

`StripDiacritics` traverses the characters starting at the address specified by `textPtr` and continues for the number of characters specified in `len`. It strips any diacritical marks from the text.

If `StripDiacritics` cannot access the specified resource, it generates an error code and does not modify the string. You need to call the `ResError` function to determine which, if any, error occurred.

Special Considerations

`StripDiacritics` may move memory; your application should not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`TextUtils.h`

TextOrder

Compares two text strings, taking into account the script and language for each of the strings. (Deprecated in Mac OS X v10.4. Use `CFStringCompare` or `UCCompareText` instead.)

```
short TextOrder (
   const void *aPtr,
   const void *bPtr,
   short aLen,
   short bLen,
   ScriptCode aScript,
   ScriptCode bScript,
   LangCode aLang,
   LangCode bLang
);
```

Parameters

aPtr
A pointer to the first character of the first text string.

bPtr
A pointer to the first character of the second text string.

aLen
The length, in bytes, of the first text string.

bLen
The length, in bytes, of the second text string.

aScript
The script code for the first text string.

bScript
The script code for the second text string.

aLang
The language code for the first text string.

bLang
The language code for the second text string.

Return Value

Returns –1 if the first string is less than the second string, 0 if the first string is equal to the second string, and 1 if the first string is greater than the second string. The ordering of script and language codes, which is based on information in the script-sorting resource, is considered in determining the relationship of the two strings.

Discussion

This function takes both primary and secondary sorting orders into consideration and returns a value that indicates whether the first string is less than, equal to, or greater than the second string.

"Implicit Language Codes" (page 1702) are listed in the Constants section. Most applications specify the language code `scriptCurLang` for both the `aLang` and `bLang` values.

`TextOrder` first calls `ScriptOrder` (page 1677); if the result of `ScriptOrder` is not 0 (that is, if the strings use different scripts), `TextOrder` returns the same result.

`TextOrder` next calls `LanguageOrder` (page 1668); if the result of `LanguageOrder` is not 0 (that is, if the strings use different languages), `TextOrder` returns the same result.

At this point, `TextOrder` has two strings that are in the same script and language, so it compares them by using the sorting rules for that script and language, applying both the primary and secondary sorting orders. If that script is not installed and enabled, it uses the sorting rules specified by the system script or the font script, depending on the state of the international resources selection flag.

The `TextOrder` function is primarily used to insert text strings in a sorted list; for sorting, rather than using this function, it may be faster to sort first by script and language by using the `ScriptOrder` and `LanguageOrder` functions, and then to call the `CompareText` (page 1656) function, to sort strings within a script or language group.

Special Considerations

`TextOrder` may move memory; your application should not call this function at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`StringCompare.h`

TypeSelectClear

Clears the key list and resets the type select record. (Deprecated in Mac OS X v10.4. Use `UCTypeSelectFlushSelectorData` instead.)

Not recommended.

```
void TypeSelectClear (
   TypeSelectRecord *tsr
);
```

Parameters

tsr

A pointer to the type-select record you want to clear.

Discussion
The use of this function is not recommended in a Unicode-based application. If you want to use this function in an application that uses the Unicode character set, you must first convert Unicode text strings to Macintosh encoded Pascal text strings. You must also provide the encoding type or be able to determine it by extracting it from the text or by examining the system or keyboard script.

Special Considerations

For Unicode-based applications, you should use the UCTypeSelect functions, which manipulate a `UCTypeSelectRef` object. For more detalis, see Unicode Utilities (`UnicodeUtilities.h`).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TypeSelect.h`

TypeSelectCompare

Compares a text buffer to the keystroke buffer. (Deprecated in Mac OS X v10.4. Use `UCTypeSelectCompare` instead.)

Not recommended.

```
short TypeSelectCompare (
   const TypeSelectRecord *tsr,
   ScriptCode testStringScript,
   StringPtr testStringPtr
);
```

Parameters

tsr

A type select record that contains the keystroke buffer.

testStringScript

The script code of the test string.

testStringPtr

A pointer to the text you want to compare to the keystroke buffer.

Return Value

A numerical value that represents the ordering of the characters in the keystroke buffer with respect to the test string buffer. The value -1 is returned if characters in the keystroke buffer sort before those in `testStringPtr`; 0 if characters in the keystroke buffer are the same as those in `testStringPtr`, and 1 if the characters in the keystroke buffer sort after those in `testStringPtr`.

Discussion

The use of this function is not recommended in a Unicode-based application. If you want to use this function in an application that uses the Unicode character set, you must first convert Unicode text strings to Macintosh encoded Pascal text strings. You must also provide the encoding type or be able to determine it by extracting it from the text or by examining the system or keyboard script.

Special Considerations

For Unicode-based applications, you should use the UCTypeSelect functions, which manipulate a `UCTypeSelectRef` object. For more detalis, see Unicode Utilities (`UnicodeUtilities.h`).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`TypeSelect.h`

TypeSelectFindItem

Finds the closest match between a specified list of characters and the keystrokes stored in the type select record. (Deprecated in Mac OS X v10.4. Use `UCTypeSelectFindItem` instead.)

Not recommended.

```
short TypeSelectFindItem (
   const TypeSelectRecord *tsr,
   short listSize,
   TSCode selectMode,
   IndexToStringUPP getStringProc,
   void *yourDataPtr
);
```

Parameters

tsr

A pointer to the type select record that contains the keystrokes you want to compare.

listSize

The size of the list to search through.

selectMode

The select mode. See `Type Select Modes` (page 1703)for a list of the constants you can supply.

getStringProc

A pointer to your index-to-string callback function. See `IndexToStringProcPtr` (page 1691) for more information.

yourDataPtr

A pointer to your data structure. This is passed to your index-to-string callback, and can be `NULL`, depending on how you implement your callback function.

Return Value

Discussion

The use of this function is not recommended in a Unicode-based application. If you want to use this function in an application that uses the Unicode character set, you must first convert Unicode text strings to Macintosh encoded Pascal text strings. You must also provide the encoding type or be able to determine it by extracting it from the text or by examining the system or keyboard script.

Special Considerations

For Unicode-based applications, you should use the UCTypeSelect functions, which manipulate a `UCTypeSelectRef` object. For more detalis, see Unicode Utilities (`UnicodeUtilities.h`).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`TypeSelect.h`

TypeSelectNewKey

Creates a new type select record. (Deprecated in Mac OS X v10.4. Use `UCTypeSelectCreateSelector` instead.)

Not recommended.

```
Boolean TypeSelectNewKey (
   const EventRecord *theEvent,
   TypeSelectRecord *tsr
);
```

Parameters

theEvent

A pointer to an event record.

tsr

A pointer to a type select record.

Return Value

Returns `true` if the function executed successfully; `false` otherwise.

Discussion

The use of this function is not recommended in a Unicode-based application. If you want to use this function in an application that uses the Unicode character set, you must first convert Unicode text strings to Macintosh encoded Pascal text strings. You must also provide the encoding type or be able to determine it by extracting it from the text or by examining the system or keyboard script.

Special Considerations

For Unicode-based applications, you should use the UCTypeSelect functions, which manipulate a `UCTypeSelectRef` object. For more detalis, see Unicode Utilities (`UnicodeUtilities.h`).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`TypeSelect.h`

UppercaseStripDiacritics

Converts any lowercase characters in a text string into their uppercase equivalents and strips any diacritical marks from the text. (Deprecated in Mac OS X v10.4. Use `CFStringTransform` instead.)

```
void UppercaseStripDiacritics (
   Ptr textPtr,
   short len,
   ScriptCode script
);
```

Parameters

textPtr

A pointer to the text string to be converted.

len

The length, in bytes, of the text string. The text string can be up to 32 KB in length.

script

The script code of the script system whose resources are used to determine the results of converting characters.

The conversion uses tables in the string-manipulation (`'itl2'`) resource of the script specified by the value of the `script` parameter. You can specify `smSystemScript` to use the system script and `smCurrentScript` to use the script of the current font in the current graphics port.

Discussion

`UppercaseStripDiacritics` traverses the characters starting at the address specified by `textPtr` and continues for the number of characters specified in `len`. It converts lowercase characters in the text into their uppercase equivalents and also strips diacritical marks from the text string. This function combines the effects of the `UppercaseText` (page 1689) and `StripDiacritics` (page 1683) functions.

If `UppercaseStripDiacritics` cannot access the specified resource, it generates an error code and does not modify the string. You need to call the `ResError` function to determine which, if any, error occurred.

Special Considerations

`UppercaseStripDiacritics` may move memory; your application should not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`TextUtils.h`

UppercaseText

Converts any lowercase characters in a text string into their uppercase equivalents. (Deprecated in Mac OS X v10.4. Use `CFStringUppercase` instead.)

```
void UppercaseText (
   Ptr textPtr,
   short len,
   ScriptCode script
);
```

Parameters

textPtr

A pointer to the text string to be converted.

len

The length, in bytes, of the text string. The text string can be up to 32 KB in length.

script

The script code of the script system whose case conversion rules are used for determining uppercase character equivalents.

The conversion uses tables in the string-manipulation (`'itl2'`) resource of the specified `script`. You can specify `smSystemScript` to use the system script and `smCurrentScript` to use the script of the current font in the current graphics port.

Discussion

`UppercaseText` traverses the characters starting at the address specified by `textPtr` and continues for the number of characters specified in `len`. It converts any lowercase characters in the text into uppercase.

If `UppercaseText` cannot access the specified resource, it generates an error code and does not modify the string. You need to call the `ResError` function to determine which, if any, error occurred.

Special Considerations

`UppercaseText` may move memory; your application should not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`TextUtils.h`

UpperString

Converts any lowercase letters in a Pascal string to their uppercase equivalents, using the Macintosh file system rules. (Deprecated in Mac OS X v10.4. Use `CFStringUppercase` instead.)

```
void UpperString (
   Str255 theString,
   Boolean diacSensitive
);
```

Parameters

theString

On input, this is the Pascal string to be converted. On output, this contains the string resulting from the conversion. `UpperString` traverses the characters in `theString` and converts any lowercase characters with character codes in the range 0x00 through 0xD8 into their uppercase equivalents. `UpperString` places the converted characters in `theString`.

diacSensitive

A flag that indicates whether the case conversion is to strip diacritical marks. If the value of this parameter is `TRUE`, diacritical marks are considered in the conversion; if it is `FALSE`, any diacritical marks are stripped.

Discussion

Only a subset of the Roman character set (character codes with values through $D8) are converted. Use this function to emulate the behavior of the Macintosh file system.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`TextUtils.h`

upperstring

Converts any lowercase letters in a Pascal string to their uppercase equivalents. (Deprecated in Mac OS X v10.4. Use `CFStringUppercase` instead.)

Not recommended

```
void upperstring (
   char *theString,
   Boolean diacSensitive
);
```

Parameters

`theString`

On input, this is the Pascal string to be converted. On output, this contains the string resulting from the conversion.

`diacSensitive`

A flag that indicates whether the case conversion is to strip diacritical marks. If the value of this parameter is `TRUE`, diacritical marks are considered in the conversion; if it is `FALSE`, any diacritical marks are stripped.

Discussion
You should use the function `CFStringUppercase` instead of this one.

Carbon Porting Notes

Use `UpperString` instead.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextUtils.h`

Callbacks

IndexToStringProcPtr

Defines a pointer to your index-to-string callback function that retrieves the string associated with an index value.

Not recommended.

```
typedef Boolean (*IndexToStringProcPtr)
(
    short item,
    ScriptCode * itemsScript,
    StringPtr * itemsStringPtr,
    void * yourDataPtr
);
```

If you name your function `MyIndexToStringProc`, you would declare it like this:

```
Boolean MyIndexToStringProcPtr (
    short item,
    ScriptCode * itemsScript,
    StringPtr * itemsStringPtr,
    void * yourDataPtr
);
```

Parameters

item
The index value for which the `TypeSelect` function requests a string.

itemsScript
The script code of the string specified by `itemsStringPtr`.

itemsStringPtr
On return, points to the string that matches the index specify by the `item` parameter.

yourDataPtr
A pointer to your data structure. This is passed to your index-to-string callback, and can be `NULL`, depending on how you implement your callback function.

Return Value
Returns `true` if a string matching that index value was found; `false` otherwise.

Discussion
The use of this function is not recommended in a Unicode-based application. If you want to use this function in an application that uses the Unicode character set, you must first convert Unicode text strings to Macintosh encoded Pascal text strings. You must also provide the encoding type or be able to determine it by extracting it from the text or by examining the system or keyboard script.

Availability
Not recommended. Available in CarbonLib 1.0 and later.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Declared In
`TypeSelect.h`

Data Types

BreakTable

Contains information used to determine the boundaries of a word.

```
struct BreakTable {
    char charTypes[256];
    short tripleLength;
    short triples[1];
};
typedef struct BreakTable BreakTable;
typedef BreakTable * BreakTablePtr;
```

Discussion
You can supply a `BreakTable` as a parameter to the function `FindWordBreaks` (page 1661).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`TextUtils.h`

FormatClass

Defines a data type used to access entries in a triple integer array.

```
typedef SInt8 FormatClass;
```

Discussion
Each of the three `FVector` entries in a triple integer array is accessed by one of the values of the `FormatClass` type. See `FVector` (page 1693) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NumberFormatting.h`

FormatStatus

Defines a data type used to denote the confidence level for a conversion.

```
typedef short FormatStatus;
```

Discussion
A `FormatStatus` value is returned by the functions `ExtendedToString` (page 1659), `StringToExtended` (page 1679), `FormatRecToString` (page 1663), and `StringToFormatRec` (page 1680).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NumberFormatting.h`

FVector

Contains position and length information for one portion of a formatted numeric string.

```
struct FVector {
    short start;
    short length;
};
typedef struct FVector FVector;
typedef FVector TripleInt[3];
```

Fields

`start`

The starting byte position in the string of the specification information.

`length`

The number of bytes used in the string for the specification information.

Discussion

The `FVector` data structure is used in the `TripleInt` (page 1698) array.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NumberFormatting.h`

IndexToStringUPP

Defines a universal procedure pointer to an index-to-string callback.

```
typedef IndexToStringProcPtr IndexToStringUPP;
```

Discussion

For more information, see the description of the `IndexToStringProcPtr` (page 1691) callback function.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`TypeSelect.h`

NBreakTable

Contains information used by the `FindWordBreaks` function to determine word boundaries.

```
struct NBreakTable {
    SInt8 flags1;
    SInt8 flags2;
    short version;
    short classTableOff;
    short auxCTableOff;
    short backwdTableOff;
    short forwdTableOff;
    short doBackup;
    short length;
    char charTypes[256];
    short tables[1];
};
typedef struct NBreakTable NBreakTable;
typedef NBreakTable * NBreakTablePtr;
```

Fields

`flags1`
: The high-order byte of the break table format flags. If the high-order bit of this byte is set to 1, this break table is in the format used by `FindWordBreaks`.

`flags2`
: The low-order byte of the break table format flags. If the value in this byte is 0, the break table belongs to a 1-byte script system; in this case `FindWordBreaks` does not check for 2-byte characters.

`version`
: The version of this break table.

`classTableOff`
: The offset in bytes from the beginning of the break table to the beginning of the class table.

`auxCTableOff`
: The offset in bytes from the beginning of the break table to the beginning of the auxiliary class table.

`backwdTableOff`
: The offset in bytes from the beginning of the break table to the beginning of the backward-processing table.

`forwdTableOff`
: The offset in bytes from the beginning of the break table to the beginning of the forward-processing table.

`doBackup`
: The minimum byte offset into the buffer for doing backward processing. If the selected character for `FindWordBreaks` has a byte offset less than `doBackup`, `FindWordBreaks` skips backward processing altogether and starts from the beginning of the buffer.

`length`
: The length in bytes of the entire break table, including the individuals tables.

`charTypes`
: The class table.

`tables`
: The data of the auxiliary class table, backward table, and forward table.

Discussion

The tables have this format and content:

- The class table is an array of 256 signed bytes. Offsets into the table represent byte values; if the entry at a given offset in the table is positive, it means that a byte whose value equals that offset is a single-byte character, and the entry at that offset is the class number for the character. If the entry is negative, it means that the byte is the first byte of a 2-byte character code, and the auxiliary class table must be used to determine the character class. Odd negative numbers are handled differently from even negative numbers.
- The auxiliary class table assigns character classes to 2-byte characters. It is used when the class table determines that a byte value represents the first byte of a 2-byte character.
  - The auxiliary class table handles odd negative values from the class table as follows. If the first word of the auxiliary class table is equal to or greater than zero, it represents the default class number for 2-byte character codes—the class assigned to every odd negative value from the class table. If the first word is less than zero, it is the negative of the offset from the beginning of the auxiliary class table to a first-byte class table (a table of 2-byte character classes that can be determined from just the first byte). The value from the class table is negated, 1 is subtracted from it to obtain an even offset, and the value at that offset into the first-byte class table is the class to be assigned.
  - The auxiliary class table handles even negative values from the class table as follows. The auxiliary class table begins with a variable-length word array. The words that follow the first word are offsets to row tables. Row tables have the same format as the class table, but are used to map the second byte of a 2-byte character code to a class number. If the entry in the class table for a given byte is an even negative number, `FindWordBreaks` negates this value to obtain the offset from the beginning of the auxiliary class table to the appropriate word, which in turn contains an offset to the appropriate row table. That row table is then used to map the second byte of the character to a class number.
- The backward-processing table is a state table used by `FindWordBreaks` for backward searching. Using the backward-processing table, `FindWordBreaks` starts at a specified character, moving backward as necessary until it encounters a word boundary.
- The forward-processing table is a state table used by `FindWordBreaks` for forward searching. Using the forward-processing table, `FindWordBreaks` starts at one word boundary and moves forward until it encounters another word boundary.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`TextUtils.h`

NumFormatString

Contains data that represents the internal number formatting specification.

```
struct NumFormatString {
    UInt8 fLength;
    UInt8 fVersion;
    char data[254];
};
typedef struct NumFormatString NumFormatString;
typedef NumFormatString NumFormatStringRec;
```

Fields

`fLength`
The number of bytes in the data actually used for this number formatting specification.

`fVersion`
The version number of the number formatting specification.

`data`
The data that comprises the number formatting specification.

Discussion

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NumberFormatting.h`

NumFormatStringRec

Defines an internal numeric representation that is independent of region, language, and other multicultural consideration.

```
typedef NumFormatString NumFormatStringRec;
```

Discussion
To allow for all of the international variations in numeric presentation styles, you need to include in your function calls a number parts table from a tokens (`'itl4'`) resource. You can usually use the number parts table in the standard tokens resource that is supplied with the system. You also need to define the format of input and output numeric strings, including which characters (if any) to use as thousand separators, whether to indicate negative values with a minus sign or by enclosing the number in parentheses, and how to display zero values.

To make it possible to map a number that was formatted for one specification into another format, the Mac OS defines an internal numeric representation that is independent of region, language, and other multicultural considerations: the `NumFormatStringRec` structure. This structure is created from a number format specification string that defines the appearance of numeric strings.

Four of the numeric string functions use the number formatting specification, defined by the `NumFormatStringRec` data type: `StringToFormatRec` (page 1680), `FormatRecToString` (page 1663), `StringToExtended` (page 1679), and `ExtendedToString` (page 1659). The number format specification structure contains the data that represents the internal number formatting specification information. This data is stored in a private format.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`NumberFormatting.h`

ScriptRunStatus

Contains script-specific information for a script run.

```
struct ScriptRunStatus {
    SInt8 script;
    SInt8 runVariant;
};
typedef struct ScriptRunStatus ScriptRunStatus;
```

Fields

`script`

The script code of the subscript run. Zero indicates the Roman script system.

`runVariant`

Script-specific information about the run, in the same format as that returned by the `CharacterType` function. This information includes the type of subscript—for example, Kanji, Katakana, or Hiragana for a Japanese script system.

Discussion

The `FindScriptRun` (page 1660) function returns the script run status structure, defined by the `ScriptRunStatus` data type, when it completes its processing, which is to find a run of subscript text in a string.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`TextUtils.h`

TripleInt

Defines a data type used to return the position and length information for three different portions of a formatted numeric string.

```
typedef FVector TripleInt[3];
```

Discussion

The `FormatRecToString` (page 1663) function uses the triple-integer array, defined by the `TripleInt` data type, to return the starting position and length in a string of three different portions of a formatted numeric string: the positive value string, the negative value string, and the zero value string. Each element of the triple integer array is an `FVector` structure. Each of the three `FVector` entries in the triple integer array is accessed by one of the values of the `FormatClass` type.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`NumberFormatting.h`

TypeSelectRecord

Contains a buffer of keystrokes, the script code associated with the keystrokes, and timer information.

```
struct TypeSelectRecord {
    unsigned long tsrLastKeyTime;
    ScriptCode tsrScript;
    Str63 tsrKeyStrokes;
};
typedef struct TypeSelectRecord TypeSelectRecord;
```

Fields

`tsrLastKeyTime`
A value that indicates timeout information.

`tsrScript`
A script code.

`tsrKeyStrokes`
The keystroke buffer.

Discussion

The `TypeSelectRecord` data structure is passed as a parameter to the functions `TypeSelectNewKey` (page 1687), `TypeSelectFindItem` (page 1686), `TypeSelectCompare` (page 1686), and `TypeSelectClear` (page 1685).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`TypeSelect.h`

Constants

Format Result Types

Specify values that can be returned in the low byte of a format status (`FormatStatus`) value.

```
enum {
    fFormatOK = 0,
    fBestGuess = 1,
    fOutOfSynch = 2,
    fSpuriousChars = 3,
    fMissingDelimiter = 4,
    fExtraDecimal = 5,
    fMissingLiteral = 6,
    fExtraExp = 7,
    fFormatOverflow = 8,
    fFormStrIsNAN = 9,
    fBadPartsTable = 10,
    fExtraPercent = 11,
    fExtraSeparator = 12,
    fEmptyFormatString = 13
};
typedef SInt8 FormatResultType;
```

Constants

`fFormatOK`
Specifies format is okay.
Available in Mac OS X v10.0 and later.
Declared in `NumberFormatting.h`.

`fBestGuess`
Specifies the format is the best guess.
Available in Mac OS X v10.0 and later.
Declared in `NumberFormatting.h`.

`fOutOfSynch`
Specifies the format is out of sync.
Available in Mac OS X v10.0 and later.
Declared in `NumberFormatting.h`.

`fSpuriousChars`
Specifies the format contains spurious characters.
Available in Mac OS X v10.0 and later.
Declared in `NumberFormatting.h`.

`fMissingDelimiter`
Specifies a missing delimiter.
Available in Mac OS X v10.0 and later.
Declared in `NumberFormatting.h`.

`fExtraDecimal`
Specifies the format contains an extra decimal sign.
Available in Mac OS X v10.0 and later.
Declared in `NumberFormatting.h`.

`fMissingLiteral`
Specifies the format is missing a literal.
Available in Mac OS X v10.0 and later.
Declared in `NumberFormatting.h`.

`fExtraExp`
Available in Mac OS X v10.0 and later.
Declared in `NumberFormatting.h`.

`fFormatOverflow`
Specifies a format overflow.
Available in Mac OS X v10.0 and later.
Declared in `NumberFormatting.h`.

`fFormStrIsNAN`
Available in Mac OS X v10.0 and later.
Declared in `NumberFormatting.h`.

`fBadPartsTable`
Specifies the parts table is bad.
Available in Mac OS X v10.0 and later.
Declared in `NumberFormatting.h`.

`fExtraPercent`
Specifies the format contains an extra percent sign.
Available in Mac OS X v10.0 and later.
Declared in `NumberFormatting.h`.

`fExtraSeparator`
Specifies an extra separator.
Available in Mac OS X v10.0 and later.
Declared in `NumberFormatting.h`.

`fEmptyFormatString`
Specifies the format string is empty.
Available in Mac OS X v10.0 and later.
Declared in `NumberFormatting.h`.

Discussion
A format result type is returned in the low byte of a format status (`FormatStatus`) value. A `FormatStatus` (page 1693) value is returned by the functions `ExtendedToString` (page 1659), `StringToExtended` (page 1679), `FormatRecToString` (page 1663), and `StringToFormatRec` (page 1680). A format status value denotes the confidence level for a conversion.

TripleInt Index Values

Specify an index for a `TripleInt` array.

```
enum {
    fPositive = 0,
    fNegative = 1,
    fZero = 2
};
```

Constants

`fPositive`

Specifies the positive value string.

Available in Mac OS X v10.0 and later.

Declared in `NumberFormatting.h`.

`fNegative`

Specifies the negative value string.

Available in Mac OS X v10.0 and later.

Declared in `NumberFormatting.h`.

`fZero`

Specifies the zero value string.

Available in Mac OS X v10.0 and later.

Declared in `NumberFormatting.h`.

Discussion

See `TripleInt` (page 1698) for more information.

NumFormatString Version

Specifies the first version of the `NumFormatString` data structure.

```
enum {
    fVNumber = 0
};
```

Discussion

See `NumFormatString` (page 1696) for more information.

Implicit Language Codes

Specify implicit language codes.

```
enum {
    systemCurLang = -2,
    systemDefLang = -3,
    currentCurLang = -4,
    currentDefLang = -5,
    scriptCurLang = -6,
    scriptDefLang = -7
};
```

Constants

`systemCurLang`
Specifies the current language for system script (from `'itlb'`).
Available in Mac OS X v10.0 and later.
Declared in `StringCompare.h`.

`systemDefLang`
Specifies the default language for system script (from `'itlm'`).
Available in Mac OS X v10.0 and later.
Declared in `StringCompare.h`.

`currentCurLang`
Specifies the current language for current script (from `'itlb'`).
Available in Mac OS X v10.0 and later.
Declared in `StringCompare.h`.

`currentDefLang`
Specifies the default language for current script (from `'itlm'`).
Available in Mac OS X v10.0 and later.
Declared in `StringCompare.h`.

`scriptCurLang`
Specifies the current language for specified script (from `'itlb'`)
Available in Mac OS X v10.0 and later.
Declared in `StringCompare.h`.

`scriptDefLang`
Specifies the default language for specified script (from `'itlm'`)
Available in Mac OS X v10.0 and later.
Declared in `StringCompare.h`.

Discussion
The functions `LanguageOrder` (page 1668), `StringOrder` (page 1678), and `TextOrder` (page 1684) accept as parameters implicit language codes listed here, as well as explicit language codes.

Type Select Modes

Contains type-select code information.

```
typedef SInt16 TSCode;
enum {
    tsPreviousSelectMode = -1,
    tsNormalSelectMode = 0,
    tsNextSelectMode = 1
};
```

Constants

`tsPreviousSelectMode`
Specifies previous-select mode.
Available in Mac OS X v10.0 and later.
Declared in `TypeSelect.h`.

`tsNormalSelectMode`
Specifies normal-select mode.
Available in Mac OS X v10.0 and later.
Declared in `TypeSelect.h`.

`tsNextSelectMode`
Specifies next-select mode.
Available in Mac OS X v10.0 and later.
Declared in `TypeSelect.h`.

Discussion
This structure is passed as a parameter to the function `TypeSelectFindItem` (page 1686).

Obsolete Language Code Values

Specify language code values that are no longer used.

```
enum {
    iuSystemCurLang = systemCurLang,
    iuSystemDefLang = systemDefLang,
    iuCurrentCurLang = currentCurLang,
    iuCurrentDefLang = currentDefLang,
    iuScriptCurLang = scriptCurLang,
    iuScriptDefLang = scriptDefLang
};
```

Translation Manager Reference

Framework:	Carbon/Carbon.h
Declared in	Translation.h TranslationExtensions.h

Overview

You can use the Translation Manager to direct the translation of documents from one format to another. For example, Macintosh Easy Open uses the Translation Manager to provide

- automatic translation of a document, if the application that created the document is not available
- automatic translation of documents drop-launched onto an application
- enhanced file-opening dialog boxes and (when necessary) automatic translation of documents the user selects in those dialog boxes
- batch desktop translation of documents
- automatic translation of data pasted from the clipboard

These services allow your application to open documents created by other applications (possibly running on other operating systems) and to import data from other applications with better fidelity than previously possible.

Macintosh Easy Open does not do any translating itself, and it does not have any knowledge of translation data models. Instead, it delegates these functions to translation extensions or to applications with built-in translation capability. Translation extensions and application translation capabilities operate as "black boxes" to Macintosh Easy Open. At system startup (or whenever new translation extensions become available), Macintosh Easy Open catalogs the translation capability of each translation extension and each application, and then invokes each as needed. Macintosh Easy Open can support multiple translation systems.

Carbon supports the Translation Manager in Mac OS 9, with the exception of the functions declared in `TranslationExtensions.h`.

Important: The Translation Manager was deprecated in Mac OS X v10.3. In Mac OS X, Carbon includes the Translation Manager headers but does not implement any of the functionality. If you call the functions declared in the API, they do nothing. You should use Translation Services instead.

Functions

CanDocBeOpened

Determines whether a specified application can open a particular document. (Deprecated in Mac OS X v10.3. Use Launch Services to determine whether a document can be opened. See *Launch Services Programming Guide*.)

Not Recommended

```
OSErr CanDocBeOpened (
   const FSSpec *targetDocument,
   short appVRefNumHint,
   OSType appSignature,
   const FileType *nativeTypes,
   Boolean onlyNative,
   DocOpenMethod *howToOpen,
   FileTranslationSpec *howToTranslate
);
```

Parameters

targetDocument

A pointer to the document to check.

appVRefNumHint

The volume reference number of the volume containing the application. The search for the specified application begins on this volume if the application isn't found there, the search continues to other mounted volumes.

appSignature

The signature of the application.

nativeTypes

A pointer to the zero-terminated list of file types that the application can open without translation; if this parameter contains `NULL`, the default list of file types returned by the `GetFileTypesThatAppCanNativelyOpen` (page 1710) function is used.

onlyNative

If `TRUE`, determine only whether the application can open the document without translation; otherwise, determine whether the application can open the document after translation.

howToOpen

On return, a pointer to a constant indicating the method of opening the document. This field contains a meaningful value only if the function returns `noErr` (indicating that the specified document can be opened). See "DocOpenMethod" (page 1731) for a description of the values you can use here.

howToTranslate

On return, if the document needs to be translated before it can be opened, a pointer to a buffer of information (in a private format) indicating how to translate the document. You pass the information returned in this parameter to the `TranslateFile` (page 1714) function.

Return Value
A result code. If the application can open the document, the function returns `noErr`.

Discussion
A preference must have already been set (using the Document Converter tool) on how to open the document.

Special Considerations

This function might cause memory to be moved or purged; you should not call it at interrupt time.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.3.

Not available to 64-bit applications.

Declared In
`Translation.h`

DisposeGetScrapDataUPP

Disposes of a universal procedure pointer (UPP) to a source-data fetcher callback. (Deprecated in Mac OS X v10.3. There is no replacement function. You should adopt Translation Services instead.)

```
void DisposeGetScrapDataUPP (
   GetScrapDataUPP userUPP
);
```

Parameters

userUPP

The universal procedure pointer.

Discussion
See the callback `GetScrapDataProcPtr` (page 1726) for more information.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.3.

Declared In
`Translation.h`

ExtendFileTypeList

Creates a list of file types that can be translated into a type in a given list. (Deprecated in Mac OS X v10.3. Use the Translation Services function `TranslationCreateWithSourceArray` instead.)

Not Recommended

```
OSErr ExtendFileTypeList (
   const FileType *originalTypeList,
   short numberOriginalTypes,
   FileType *extendedTypeList,
   short *numberExtendedTypes
);
```

Parameters

originalTypeList

A pointer to a list of file types that can be opened.

numberOriginalTypes

The number of file types in the `originalTypeList` parameter.

extendedTypeList

On return, a pointer to a buffer filled with file types that can be translated into the types in `originalTypeList`.

numberExtendedTypes

On input, a pointer to the maximum number of file types that can be put into the buffer passed in the `extendedTypeList` parameter. On return, a pointer to the actual number of file types put into the extended type list.

Return Value

A result code. See "Translation Manager Result Codes" (page 1732).

Discussion

Note that the number of types specified in the parameters `numberOriginalTypes` and `numberExtendedTypes` is limited only by available memory.

The Standard File Package calls this function internally your application probably won't need to use it.

Special Considerations

This function might cause memory to be moved or purged; you should not call it at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.3.

Not available to 64-bit applications.

Declared In

`Translation.h`

GetDocumentKindString

Allows you to get a document kind string. (Deprecated in Mac OS X v10.3. There is no replacement function, but you can use Launch Services to obtain similar information. See *Launch Services Programming Guide*.)

Not Recommended

```
OSErr GetDocumentKindString (
   short docVRefNum,
   OSType docType,
   OSType docCreator,
   Str63 kindString
);
```

Parameters

docVRefNum

The volume containing the document. This is a hint to the Translation Manager. If it doesn't find the string on that volume, it will use an internal search path to look on other volumes for the string.

docType

The type code of the document you want to query.

docCreator

The creator code of the document you want to query.

kindString

Upon return, contains the kind string to display for the specified document type and creator.

Return Value
A result code. See "Translation Manager Result Codes" (page 1732).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.3.
Not available to 64-bit applications.

Declared In
`Translation.h`

GetFileTranslationPaths

A low-level routine that allows you to get all the translation capabilities of the Translation Manager. (Deprecated in Mac OS X v10.3. Use the Translation Services function `TranslationCreateWithSourceArray` instead.)

Not Recommended

```
short GetFileTranslationPaths (
   const FSSpec *srcDocument,
   FileType dstDocType,
   unsigned short maxResultCount,
   FileTranslationSpecArrayPtr resultBuffer
);
```

Parameters

srcDocument

A source document (optional), or `NULL`.

dstDocType

The desired document type to which you would like `srcDocument` translated.

resultBuffer

The requested translation information.

Return Value
Returns the number of translation paths, or a result code if the value is negative.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.3.
Not available to 64-bit applications.

Declared In
`Translation.h`

GetFileTypesThatAppCanNativelyOpen

Obtains a list of file types that an application can open by itself. (Deprecated in Mac OS X v10.3. There is no replacement function. Applications should use Translation Services instead.)

Not Recommended

```
OSErr GetFileTypesThatAppCanNativelyOpen (
   short appVRefNumHint,
   OSType appSignature,
   FileType *nativeTypes
);
```

Parameters

appVRefNumHint
The volume reference number of the volume containing the application. The search for the specified application begins on this volume if the application isn't found there, the search continues to other mounted volumes.

appSignature
The signature of the application.

nativeTypes
On return, a pointer to an array of file types that the application can open without translation. If successful, the array contains up to 64 file types. If fewer than 64 types are returned, the end of the list is indicated by an entry whose value is 0.

Return Value
A result code. See "Translation Manager Result Codes" (page 1732).

Special Considerations
This function might cause memory to be moved or purged; you should not call it at interrupt time.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.3.
Not available to 64-bit applications.

Declared In
`Translation.h`

GetPathFromTranslationDialog

Displays the Translation Dialog, which attempts to generate a list of translation paths resulting in a document readable by the target application. (Deprecated in Mac OS X v10.3. There is no replacement function. However, applications can obtain this information using Launch Services, Translation Services, and uniform type identifiers.)

Not Recommended

```
OSErr GetPathFromTranslationDialog (
   const FSSpec *theDocument,
   const FSSpec *theApplication,
   TypesBlockPtr typeList,
   DocOpenMethod *howToOpen,
   FileTranslationSpec *howToTranslate
);
```

Parameters

theDocument
The target file.

theApplication
The target application.

typeList
Specifies a list of file types into which to translate the target document.

howToOpen
Upon return, contains the translation open method.

howToTranslate
Upon return, contains the translation specification.

Return Value

A result code. See "Translation Manager Result Codes" (page 1732).

Special Considerations

For information about using Launch Services and uniform type identifiers, see *Launch Services Programming Guide* and *Uniform Type Identifiers Overview*.

Availability

Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.3.
Not available to 64-bit applications.

Declared In

`Translation.h`

GetTranslationExtensionName

Finds the name of the extension performing the translation. (Deprecated in Mac OS X v10.3. There is no replacement function. However, you can obtain useful user-level information by calling the Translation Manager function `TranslationCopyDestinationType` and the uniform type identifiers function `UTTypeCopyDescription`.)

Not Recommended

```
OSErr GetTranslationExtensionName (
   const FileTranslationSpec *translationMethod,
   Str31 extensionName
);
```

Parameters

translationMethod

A file translation method obtained by calling `CanDocBeOpened` or `GetFileTranslationPaths`.

extensionName

Upon return, contains the name of the extension performing the translation.

Return Value

A result code. See "Translation Manager Result Codes" (page 1732).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.3.

Not available to 64-bit applications.

Declared In

`Translation.h`

InvokeGetScrapDataUPP

Calls a source-data fetcher callback. (Deprecated in Mac OS X v10.3. There is no replacement function. You should adopt Translation Services instead.)

```
OSErr InvokeGetScrapDataUPP (
   ScrapType requestedFormat,
   Handle dataH,
   void *srcDataGetterRefCon,
   GetScrapDataUPP userUPP
);
```

Discussion

You should not need to use the function `InvokeGetScrapDataUPP`, as the system calls your source-data fetcher callback for you. See the callback `GetScrapDataProcPtr` (page 1726) for more information.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.3.

Declared In

`Translation.h`

NewGetScrapDataUPP

Creates a new universal procedure pointer (UPP) to a source-data fetcher callback. (Deprecated in Mac OS X v10.3. There is no replacement function. You should adopt Translation Services instead.)

```
GetScrapDataUPP NewGetScrapDataUPP (
   GetScrapDataProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your source-data fetcher callback.

Return Value

On return, a UPP to the source-data fetcher callback.

Discussion

See the callback `GetScrapDataProcPtr` (page 1726) for more information.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.3.

Declared In

`Translation.h`

SetTranslationAdvertisement

Allows your translation extension to install an advertisement into the upper portion of the progress dialog box. (Deprecated in Mac OS X v10.3. There is no replacement function. You should adopt Translation Services instead.)

Unsupported

```
OSErr SetTranslationAdvertisement (
   TranslationRefNum refNum,
   PicHandle advertisement
);
```

Parameters

refNum

A translation reference number. You should set this parameter to the translation reference number passed to your `DoTranslateFileProcPtr` (page 1723) or `DoTranslateScrapProcPtr` (page 1724) callback functions. The Translation Manager uses that number internally.

advertisement

A handle to a picture to display in the upper portion of the dialog box. If this parameter is `NULL`, no advertisement is displayed and the upper portion of the dialog box is removed before the box is displayed to the user. After the function installs the specified advertisement, it then displays the dialog box.

Your translation extension can read the picture data from its resource fork, but it should detach the resource from the resource fork (by calling the `DetachResource` function) and make the handle unpurgeable before calling this function. Because you'll usually load the picture data into the temporary heap provided for the translation extension, the picture data is automatically disposed of when that heap is destroyed. If your translation extension loads the picture data elsewhere in memory, you are responsible for disposing of it before returning from your `DoTranslateFile` or `DoTranslateScrap` callback function.

The size of the picture to display can be no larger than 280 by 50 pixels. If the picture you specify is smaller than that, it is automatically centered (both vertically and horizontally) in the available space.

Return Value
A result code. See "Translation Manager Result Codes" (page 1732).

Discussion
Your translation extension should call this function only in response to the `kTranslateTranslateFile` or `kTranslateTranslateScrap` request code (that is, you should only call this function in your `DoTranslateFile` or `DoTranslateScrap` callback function). Do not call this function in response to any other request code or from any code that isn't a translation extension.

You must call this function before you call the `UpdateTranslationProgress` (page 1716) function for the first time.

Special Considerations

This function might cause memory to be moved or purged; you should not call it at interrupt time.

Carbon Porting Notes

The functions contained in TranslationExtensions.h were originally written to be used only by someone implementing a Mac Easy Open translation component. Carbon, however, is for applications and not extensions. Therefore, this function is not supported.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.3.
Not available to 64-bit applications.

Declared In
`TranslationExtensions.h`

TranslateFile

Translates a document from one format to another. (Deprecated in Mac OS X v10.3. Use the Translation Services function `TranslationPerformForFile` or `TranslationPerformForURL` instead.)

Not Recommended

```
OSErr TranslateFile (
   const FSSpec *sourceDocument,
   const FSSpec *destinationDocument,
   const FileTranslationSpec *howToTranslate
);
```

Parameters

sourceDocument

A pointer to the document to translate.

destinationDocument

A pointer to the file to put the translated document into. Note that your application only specifies the name and location for the file; the function creates the file and puts the translated data into it. The destination file must not exist before you call this function.

howToTranslate

A pointer to a buffer of information indicating how to translate the document. Usually, you'll get this information by calling the `CanDocBeOpened` (page 1706) function.

Return Value

A result code. See "Translation Manager Result Codes" (page 1732).

Special Considerations

This function might cause memory to be moved or purged; you should not call it at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.3.

Not available to 64-bit applications.

Declared In

`Translation.h`

TranslateScrap

Uses a client-specified callback function to get the source data to translate. (Deprecated in Mac OS X v10.3. Use the Translation Manager function `TranslationPerformForData` instead.)

Not Recommended

```
OSErr TranslateScrap (
   GetScrapDataUPP sourceDataGetter,
   void *sourceDataGetterRefCon,
   ScrapType destinationFormat,
   Handle destinationData,
   short progressDialogID
);
```

Parameters

sourceDataGetter

The callback that provides the data to translate.

sourceDataGetterRefCon

A generic pointer to private information for your callback.

destinationFormat
The desired translation format.

destinationData
A handle you provide. The Translation Extension will automatically re-size it as necessary during translation. Upon exit, if the routine successfully executes, it will contain the translated information.

progressDialogID
This parameter should always be assigned the value `TranslationScrapProgressDialogID`.

Return Value
A result code. See "Translation Manager Result Codes" (page 1732).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.3.
Not available to 64-bit applications.

Declared In
`Translation.h`

UpdateTranslationProgress

Allows your translation extension to update the progress dialog box that is displayed during file and scrap translation and give the user a chance to cancel. (Deprecated in Mac OS X v10.3. There is no replacement function. You should adopt Translation Services instead.)

Unsupported

```
OSErr UpdateTranslationProgress (
   TranslationRefNum refNum,
   short percentDone,
   Boolean *canceled
);
```

Parameters

refNum
A translation reference number. You should set this parameter to the translation reference number passed to your `DoTranslateFileProcPtr` (page 1723) or `DoTranslateScrapProcPtr` (page 1724) callback function. The Translation Manager uses that number internally.

percentDone
A value in the range 0–100 that indicates the percentage of the translation that has been completed (the approximate percentage of time elapsed until completion). When the translation is complete, you should call this function with this parameter set to 100 so that the user can see that the translation is complete.

canceled
On return, a pointer to a value which indicates whether or not the user canceled the translation. `TRUE` if the user clicked the Cancel button in the progress dialog box, or typed Command-period while the box is displayed; otherwise, `FALSE`. When `TRUE`, you should stop the translation, and your `DoTranslateFile` or `DoTranslateScrap` callback function should return the result code `userCancelledErr`.

Return Value
A result code. See "Translation Manager Result Codes" (page 1732).

Discussion
Your translation extension should call this function only in response to the `kTranslateTranslateFile` or `kTranslateTranslateScrap` request code (that is, you should only call this function in your `DoTranslateFile` or `DoTranslateScrap` callback function). Do not call this function in response to any other request code or from any code that isn't a translation extension.

You should already have called `SetTranslationAdvertisement` before calling `UpdateTranslationProgress`.

Special Considerations

This function might cause memory to be moved or purged; you should not call it at interrupt time.

Carbon Porting Notes

The functions contained in TranslationExtensions.h were originally written to be used only by someone implementing a Mac Easy Open translation component. Carbon, however, is for applications and not extensions. Therefore, this function is not supported.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.3.
Not available to 64-bit applications.

Declared In
`TranslationExtensions.h`

Callbacks

DoGetFileTranslationListProcPtr

Defines a pointer to a get file translation list callback function that returns a list of the file types which your extension can translate.

Unsupported

```
typedef ComponentResult (*DoGetFileTranslationListProcPtr)
(
    ComponentInstance self,
    FileTranslationListHandle translationList
);
```

If you name your function `MyDoGetFileTranslationListProc`, you would declare it like this:

```
ComponentResult DoGetFileTranslationListProcPtr
(
    ComponentInstance self,
    FileTranslationListHandle translationList
);
```

Parameters

self
A component instance that identifies the component containing the translation extension.

translationList

On entry to your function, this parameter contains a handle to a structure of type `FileTranslationList` (page 1727). If your translation extension can translate any files at all, your function should resize that handle and fill the block with a list of the file types it can translate. If the translation list whose handle you return in this parameter has the `groupCount` field set to 0, Macintosh Easy Open assumes that your extension cannot translate any file types.

For improved performance, Macintosh Easy Open remembers each translation extension's most recently returned file translation list and passes that list to your get file translation list callback function in this parameter. If you determine that the list hasn't changed, you should simply return the same handle to Macintosh Easy Open.

Return Value
If successful, your function should return `noErr`. Otherwise, your function should return an appropriate result code. The Component Manager requires this function to return a value of type `ComponentResult` to simplify dispatching. See the Component Manager documentation for a description of the `ComponentResult` data type.

Discussion
A file translation extension must respond to the `kTranslateGetFileTranslationList` request code. Whenever it first notices the extension, Macintosh Easy Open calls your extension with this request code to obtain a list of the file types that the extension can translate. You can handle this request by calling the Component Manager function `CallComponentFunctionWithStorage` and passing it a pointer to a your get file translation list callback function.

Carbon Porting Notes

The functions contained in TranslationExtensions.h were originally written to be used only by someone implementing a Mac Easy Open translation component. Carbon, however, is for applications and not extensions. Therefore, this function is not supported.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TranslationExtensions.h`

DoGetScrapTranslationListProcPtr

Defines a pointer to a get-scrap-translation-list callback function that returns a list of the scrap types that your extension can translate.

Unsupported

```
typedef ComponentResult (*DoGetScrapTranslationListProcPtr)
(
    ComponentInstance self,
    ScrapTranslationListHandle list
);
```

If you name your function `MyDoGetScrapTranslationListProc`, you would declare it like this:

```
ComponentResult DoGetScrapTranslationListProcPtr
(
    ComponentInstance self,
    ScrapTranslationListHandle list
```

```
);
```

Parameters

self

A component instance that identifies the component containing the translation extension.

list

A handle to a `ScrapTranslationList` (page 1729). Your function should return, through this parameter, a handle to a list of the scrap types from and into which your translation extension can translate. If your translation extension can translate any scrap types at all, your function should resize this handle and fill the block with a list of the scrap types it can translate. If the translation list whose handle you return in this parameter has the `groupCount` field set to 0, Macintosh Easy Open assumes that your extension cannot translate any scrap types.

On entry to your function, this parameter contains a handle to a structure of type `ScrapTranslationList`. When it first becomes aware of your extension, Macintosh Easy Open calls your translation extension's get scrap translation list function. For improved performance, Macintosh Easy Open remembers each translation extension's most recently returned scrap translation list and passes that list to your function in this parameter. If you determine that the list hasn't changed, you should simply return the same handle to Macintosh Easy Open.

Return Value

If successful, your function should return `noErr`. Otherwise, your function should return an appropriate result code. The Component Manager requires this function to return a value of type `ComponentResult` to simplify dispatching. See the Component Manager documentation for a description of the `ComponentResult` data type.

Discussion

A scrap translation extension must respond to the `kTranslateGetScrapTranslationList` request code. At system startup time, the Translation Manager calls your extension with this code. You can handle this request by calling the Component Manager function `CallComponentFunctionWithStorage` and passing it a pointer to your get scrap translation list callback function.

Carbon Porting Notes

The functions contained in TranslationExtensions.h were originally written to be used only by someone implementing a Mac Easy Open translation component. Carbon, however, is for applications and not extensions. Therefore, this function is not supported.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`TranslationExtensions.h`

DoGetTranslatedFilenameProcPtr

Defines a pointer to a get-translated-filename callback function.

Unsupported

```
typedef ComponentResult (*DoGetTranslatedFilenameProcPtr)
(
    ComponentInstance self,
    FileType dstType,
    long dstTypeHint,
    FSSpec * theDocument
);
```

If you name your function `MyDoGetTranslatedFilenameProc`, you would declare it like this:

```
ComponentResult DoGetTranslatedFilenameProcPtr
(
    ComponentInstance self,
    FileType dstType,
    long dstTypeHint,
    FSSpec * theDocument
);
```

Parameters

self

dstType

theDocument

Return Value

See the Component Manager documentation for a description of the `ComponentResult` data type.

Carbon Porting Notes

The functions contained in TranslationExtensions.h were originally written to be used only by someone implementing a Mac Easy Open translation component. Carbon, however, is for applications and not extensions. Therefore, this function is not supported.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`TranslationExtensions.h`

DoIdentifyFileProcPtr

Defines a pointer to a file identification callback function that identifies a file as having a format which your extension can translate.

Unsupported

```
typedef ComponentResult (*DoIdentifyFileProcPtr)
(
    ComponentInstance self,
    const FSSpec * theDocument,
    FileType * docType
);
```

If you name your function `MyDoIdentifyFileProc`, you would declare it like this:

```
ComponentResult DoIdentifyFileProcPtr (
    ComponentInstance self,
```

```
    const FSSpec * theDocument,
    FileType * docType
);
```

Parameters

self

A component instance that identifies the component containing the translation extension.

theDocument

A pointer to a file system specification structure that specifies the document that the translation extension must identify.

docType

Your function should return, in this parameter, the file format type of the file specified in the `theDocument` parameter.

Your function should not return `'TEXT'` as a file type unless you determine that the document consists solely of a plain, unformatted stream of ASCII characters.

Return Value

If successful, your function should return `noErr`. Otherwise, your function should return an appropriate result code. If your translation extension does not recognize the type of the specified file, your function should return the result code `noTypeErr`. The Component Manager requires this function to return a value of type `ComponentResult` to simplify dispatching. See the Component Manager documentation for a description of the `ComponentResult` data type.

Discussion

A file translation extension must respond to the `kTranslateIdentifyFile` request code. The Translation Manager uses this request code to allow the translation extension to identify a file as having a format that the extension can translate. You can handle this request by calling the Component Manager function `CallComponentFunctionWithStorage` and passing it a pointer to your file identification callback function.

Carbon Porting Notes

The functions contained in TranslationExtensions.h were originally written to be used only by someone implementing a Mac Easy Open translation component. Carbon, however, is for applications and not extensions. Therefore, this function is not supported.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`TranslationExtensions.h`

DoIdentifyScrapProcPtr

Defines a pointer to a scrap identification callback function that identifies a scrap as one that your scrap translation extension can translate.

Unsupported

```
typedef ComponentResult (*DoIdentifyScrapProcPtr)
(
    ComponentInstance self,
    const void * dataPtr,
    Size dataLength,
    ScrapType * dataFormat
);
```

If you name your function `MyDoIdentifyScrapProc`, you would declare it like this:

```
ComponentResult DoIdentifyScrapProcPtr
(
    ComponentInstance self,
    const void * dataPtr,
    Size dataLength,
    ScrapType * dataFormat
);
```

Parameters

self

A component instance that identifies the component containing the translation extension.

dataPtr

A pointer to the scrap to translate.

dataLength

The size of the scrap to translate.

dataFormat

On entry, the type of the scrap format. Your function returns, through this parameter, the type of the scrap format of the scrap specified by the `dataPtr` parameter, as recognized by your translation extension.

In general, the scrap that your `DoIdentifyScrap` function is asked to identify is always in one of the formats listed among the source formats in the translation groups contained in your extension's scrap translation list. Your scrap translation extension therefore needs only to verify that the indicated scrap is of the specified format.

Return Value

If successful, your function should return `noErr`. Otherwise, your function should return an appropriate result code. If your translation extension does not recognize the type of the specified scrap, your function should return the result code `noTypeErr`. The Component Manager requires this function to return a value of type `ComponentResult` to simplify dispatching. See the Component Manager documentation for a description of the `ComponentResult` data type.

Discussion

A scrap translation extension must respond to the `kTranslateIdentifyScrap` request code. You can handle this request by calling the Component Manager function `CallComponentFunctionWithStorage` and passing it a pointer to a your scrap identification callback function.

Carbon Porting Notes

The functions contained in TranslationExtensions.h were originally written to be used only by someone implementing a Mac Easy Open translation component. Carbon, however, is for applications and not extensions. Therefore, this function is not supported.

Availability

Available in Mac OS X v10.0 and later.

Declared In
`TranslationExtensions.h`

DoTranslateFileProcPtr

Defines a pointer to a file translation callback function that translates a document from one format into another.

Unsupported

```
typedef ComponentResult (*DoTranslateFileProcPtr)
(
    ComponentInstance self,
    TranslationRefNum refNum,
    const FSSpec * sourceDocument,
    FileType srcType,
    long srcTypeHint,
    const FSSpec * dstDoc,
    FileType dstType,
    long dstTypeHint
);
```

If you name your function `MyDoTranslateFileProc`, you would declare it like this:

```
ComponentResult DoTranslateFileProcPtr
(
    ComponentInstance self,
    TranslationRefNum refNum,
    const FSSpec * sourceDocument,
    FileType srcType,
    long srcTypeHint,
    const FSSpec * dstDoc,
    FileType dstType,
    long dstTypeHint
);
```

Parameters

self
: A component instance that identifies the component containing your translation extension.

refNum
: The translation reference number for this translation.

  Macintosh Easy Open assigns this reference number to the translation. Each translation is assigned a unique number to distinguish the translation from any other translations that might occur. You need to pass this reference number to any Macintosh Easy Open functions you call from within the file translation extension; for instance, if by calling the `SetTranslationAdvertisement` function you display the progress dialog box, you'll pass that reference number in the `refNum` parameter.

sourceDocument
: A file system specification structure that specifies the document to translate.

srcType
: The format of the file to be translated.

srcTypeHint
: The value in the `hint` field of the source document's file type specification.

dstDoc

A file system specification structure that specifies the destination document.

Your function should put the translated document into the file specified by this parameter. The data fork of the destination file already exists by the time your function is called. In addition, if the `flags` field in the appropriate destination file type specification in your extension's file translation list has the `taDstDocNeedsResourceFork` bit set, the destination file already contains a resource fork. Your function should open the destination file and fill its data or resource fork (or both) with the appropriate translated data.

dstType

The format into which to translate the source document.

dstTypeHint

The value in the `hint` field of the destination document's file type specification.

Return Value

If successful, your function should return `noErr`. Otherwise, your function should return an appropriate result code. If it cannot translate the source file, your function should return a result code different from `noErr`. In that case, Macintosh Easy Open will automatically delete the destination file. The Component Manager requires this function to return a value of type `ComponentResult` to simplify dispatching. See the Component Manager documentation for a description of the `ComponentResult` data type.

Discussion

A file translation extension must respond to the `kTranslateTranslateFile` request code. You can handle this request by calling the Component Manager function `CallComponentFunctionWithStorage` and passing it a pointer to your file translation function.

Your file translation function can translate the source file itself or rely upon external translators.

Your translation extension should call the `SetTranslationAdvertisement` (page 1713) function to display the progress dialog box and the `UpdateTranslationProgress` (page 1716) function to update the dialog box periodically.

Carbon Porting Notes

The functions contained in TranslationExtensions.h were originally written to be used only by someone implementing a Mac Easy Open translation component. Carbon, however, is for applications and not extensions. Therefore, this function is not supported.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`TranslationExtensions.h`

DoTranslateScrapProcPtr

Defines a pointer to a scrap translation callback function that translates a scrap from one format into another.

Unsupported

```
typedef ComponentResult (*DoTranslateScrapProcPtr)
(
    ComponentInstance self,
    TranslationRefNum refNum,
    const void * srcDataPtr,
    Size srcDataLength,
    ScrapType srcType,
    long srcTypeHint,
    Handle dstData,
    ScrapType dstType,
    long dstTypeHint
);
```

If you name your function `MyDoTranslateScrapProc`, you would declare it like this:

```
ComponentResult DoTranslateScrapProcPtr
(
    ComponentInstance self,
    TranslationRefNum refNum,
    const void * srcDataPtr,
    Size srcDataLength,
    ScrapType srcType,
    long srcTypeHint,
    Handle dstData,
    ScrapType dstType,
    long dstTypeHint
);
```

Parameters

self

A component instance that identifies the component containing the translation extension.

refNum

The translation reference number for this translation.

Macintosh Easy Open assigns this reference number to the translation. Each translation is assigned a unique number to distinguish the translation from any other translations that might be occurring. You need to pass this reference number to any Macintosh Easy Open functions you call from within the scrap translation extension; for instance, if you display the progress dialog box by calling the `SetTranslationAdvertisement` function, you'll pass that reference number in the `refNum` parameter.

srcDataPtr

A pointer to the scrap to translate.

srcDataLength

The size of the scrap to translate.

srcType

The format of the scrap to translate.

srcTypeHint

The value in the `hint` field of the source document's scrap type specification.

dstData

A handle to the destination to be filled in. Your function should put the translated data into the block specified here, resizing it as necessary.

dstType
The format into which to translate the source scrap.

dstTypeHint
The value in the `hint` field of the destination document's scrap type specification.

Return Value
If successful, your function should return `noErr`. Otherwise, your function should return an appropriate result code. The Component Manager requires this function to return a value of type `ComponentResult` to simplify dispatching. See the Component Manager documentation for a description of the `ComponentResult` data type.

Discussion
A scrap translation extension must respond to the `kTranslateTranslateScrap` request code. You can handle this request by calling the Component Manager function `CallComponentFunctionWithStorage` and passing it a pointer to your function. Your scrap translation callback function can translate the source file itself or rely upon external translators.

Your translation extension should call the `SetTranslationAdvertisement` (page 1713) function to display the progress dialog box and the `UpdateTranslationProgress` (page 1716) function to update the dialog box periodically.

Carbon Porting Notes

The functions contained in TranslationExtensions.h were originally written to be used only by someone implementing a Mac Easy Open translation component. Carbon, however, is for applications and not extensions. Therefore, this function is not supported.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TranslationExtensions.h`

GetScrapDataProcPtr

Defines a pointer to a source-data fetching callback.

```
typedef OSErr (*GetScrapDataProcPtr)
(
    ScrapType requestedFormat,
    Handle dataH,
    void * srcDataGetterRefCon
);
```

If you name your function `MyGetScrapDataProc`, you would declare it like this:

```
OSErr GetScrapDataProcPtr (
    ScrapType requestedFormat,
    Handle dataH,
    void * srcDataGetterRefCon
);
```

Parameters

requestedFormat

Format of data that `TranslateScrap` (page 1715) needs.

dataH

The handle in which to put the requested data.

Return Value

A result code. See "Translation Manager Result Codes" (page 1732).

Discussion

The first time this function is call ed, you should resize and fill in the handle with a list all the formats that you have available to be translated, and the length of each. When called again, you should supply the data in one of the formats in the list.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Translation.h`

Data Types

FileTranslationList

Defines a list of file formats an extension can translate.

```
struct FileTranslationList {
  unsigned long modDate;
  unsigned long groupCount;
};
typedef struct FileTranslationList  FileTranslationList;
typedef FileTranslationList *       FileTranslationListPtr;
typedef FileTranslationListPtr *    FileTranslationListHandle;
```

Fields

`modDate`

The creation date of the file translation list. If your extension uses external translators, you might set this field to the modification date of a folder containing those translators.

`groupCount`

The number of translation groups that follow.

Discussion

The Translation Manager uses the file translation list that it gets from each translation system to create a master database of format translations it can direct. A file translation list consists of a field indicating the modification date of the list and a count of the number of groups that follow those two fields.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`TranslationExtensions.h`

FileTranslationSpec

Defines a file translation method supported by the Translation Manager.

```
struct FileTranslationSpec {
  OSType          componentSignature;
  const void *    translationSystemInfo;
  FileTypeSpec    src;
  FileTypeSpec    dst;
};
typedef struct FileTranslationSpec      FileTranslationSpec;
typedef FileTranslationSpec *           FileTranslationSpecArrayPtr;
typedef FileTranslationSpecArrayPtr *   FileTranslationSpecArrayHandle;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Translation.h`

FileType

Defines the translation file type of a document.

```
typedef OSType FileType;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TranslationExtensions.h`

FileTypeSpec

Defines a descriptor for the source or destination file type used in a translation method.

```
struct FileTypeSpec {
    FileType format;
    long hint;
    TranslationAttributes flags;
    OSType catInfoType;
    OSType catInfoCreator;
};
typedef struct FileTypeSpec FileTypeSpec;
```

Fields

`format`
: The translation file type of the document. Macintosh Easy Open uses this field as the canonical way to describe the format of a file for translation purposes.

`hint`
: A 4-byte value reserved for use by your translation extension.

`flags`
: A 4-byte value consisting of bit flags that specify how to control the translation. This field is used only for destination file types; you should set it to 0 for all source file type specifications. Currently 2 bits are defined; all other bits should be cleared to 0.

`catInfoType`
: The type of the file as contained in the volume's catalog file.

`catInfoCreator`
: The creator of the file as contained in the volume's catalog file.

Discussion
The `FileTranslationList` (page 1727) structure uses file type specifications to describe document formats. A file type specification is defined by the `FileTypeSpec` data structure.

The interpretation of some of the fields of a file type specification depends on whether the specification occurs in the list of source document types or in the list of destination document types:

In file type specifications occurring in the list of source document types in a file translation list, Macintosh Easy Open uses the `format` and `catInfoCreator` fields to determine the kind string displayed in the "From" format specification of the translation progress dialog box.

In file type specifications occurring in the list of destination document types in a file translation list, Macintosh Easy Open uses the `format` and `catInfoCreator` fields to determine the kind string displayed in the "To" format specification in the translation progress dialog box. The `format` and `catInfoCreator` fields are also used to get the information displayed in the Document Converter dialog box. However, Macintosh Easy Open uses the `catInfoType` and `catInfoCreator` fields to set the catalog type and creator of the destination file.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TranslationExtensions.h`

ScrapTranslationList

Defines the scrap formats an extension can translate.

```
struct ScrapTranslationList {
  unsigned long   modDate;
  unsigned long   groupCount;
};
typedef struct ScrapTranslationList     ScrapTranslationList;
typedef ScrapTranslationList *          ScrapTranslationListPtr;
typedef ScrapTranslationListPtr *       ScrapTranslationListHandle;
```

Fields
`modDate`
: The creation date of the scrap translation list. If your extension uses external translators, you might set this field to the modification date of a folder containing those translators.

`groupCount`
: The number of translation groups that follow. The size of the translation list prepared by an extension is variable, depending upon the number of groups, the scrap specification structure size, and the number of scrap types that the extension knows about.

Discussion
The Translation Manager uses the scrap translation list that it gets from each translation system to create a master database of its translation capability. A scrap translation list consists of a field indicating the modification date of the list and a count of the number of groups that follow those two fields.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TranslationExtensions.h`

ScrapType

Defines the scrap type in a Translation Manager scrap format.

```
typedef ResType ScrapType;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TranslationExtensions.h`

ScrapTypeSpec

Describes a specific scrap format.

```
struct ScrapTypeSpec {
    ScrapType format;
    long hint;
};
typedef struct ScrapTypeSpec ScrapTypeSpec;
```

Fields
`format`
The type of the specified scrap.
`hint`
A 4-byte value reserved for use by your translation extension.

Discussion
The `ScrapTypeSpec` data structure is used by the `ScrapTranslationList` (page 1729) structure.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TranslationExtensions.h`

TypesBlock

Defines a null-terminated array of `OSType` or `FileType` elements.

```
typedef OSType      TypesBlock[64];
typedef OSType *    TypesBlockPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Translation.h`

Constants

DocOpenMethod

Specifies the ways a document can be opened.

```
enum {
  domCannot = 0,
  domNative = 1,
  domTranslateFirst = 2,
  domWildcard = 3
};
typedef short DocOpenMethod;
```

Constants

`domCannot`
Indicates that the application cannot open the document.
Available in Mac OS X v10.0 and later.
Declared in `Translation.h`.

`domNative`
Indicates that the application can open the document natively.
Available in Mac OS X v10.0 and later.
Declared in `Translation.h`.

`domTranslateFirst`
Indicates that the application can open the document only after it's been translated.
Available in Mac OS X v10.0 and later.
Declared in `Translation.h`.

`domWildcard`
Indicates that the application has the file type '****' in its list of the file types that it can open and hence can open any type of document.
Available in Mac OS X v10.0 and later.
Declared in `Translation.h`.

Discussion
The `CanDocBeOpened` (page 1706) function uses the following constants to specify the method for opening a given document.

Result Codes

The most common result codes returned by the Translation Manager are listed in the table below.

Result Code	Value	Description
`invalidTranslationPathErr`	-3025	Source type to destination type not a valid path. Available in Mac OS X v10.0 and later.
`couldNotParseSourceFileErr`	-3026	Source document does not contain source type. Available in Mac OS X v10.0 and later.
`noTranslationPathErr`	-3030	Application cannot open document. Available in Mac OS X v10.0 and later.
`badTranslationSpecErr`	-3031	Translation path is invalid. Available in Mac OS X v10.0 and later.
`noPrefAppErr`	-3032	No translation preference available. Available in Mac OS X v10.0 and later.

Gestalt Constants

You can check for version and feature availability information by using the Translation Manager selectors defined in the Gestalt Manager. For more information, see *Inside Mac OS X: Gestalt Manager Reference*.

URL Access Manager Reference (Not Recommended)

Framework: Carbon/Carbon.h

Declared in URLAccess.h

Overview

Important: URL Access Manager is deprecated as of Mac OS X v10.4. You should use CFNetwork instead (as described in *CFNetwork Programming Guide*).

CFNetwork provides better reliability and performance and is used by Apple's own applications. URL Access Manager is no longer being enhanced or improved.

The URL Access Manager is an API that you can use to perform data transfer to and from a URL from within your application. It includes support for automatic decompression of compressed files and for automatic file extraction from Stuffit archives (with version 5.0 of Stuffit).

The URL Access Manager allows you to use any of the following protocols during download operations: File Transfer Protocol (FTP), Hypertext Transfer Protocol (HTTP), secure Hypertext Transfer Protocol (HTTPS), or a URL representing a local file (begins with `file://`). You might use the latter to test your application on a computer that does not have access to an HTTP or FTP server. For upload operations, you must use an FTP URL.

This document describes the URL Access Manager API through version 2.0.3.

Functions by Task

Getting Information About the URL Access Manager

`URLGetURLAccessVersion` (page 1749) Deprecated in Mac OS X v10.4

Determines the version of URL Access Manager installed on the user's system. (Deprecated. Use CFNetwork instead; see *CFNetwork Programming Guide*.)

Creating and Disposing of a URL Reference

`URLDisposeReference` (page 1741) Deprecated in Mac OS X v10.4

Disposes of the memory associated with a URL reference. (Deprecated. Use CFNetwork instead; see *CFNetwork Programming Guide*.)

URLNewReference (page 1750) Deprecated in Mac OS X v10.4
Creates a URL reference. (Deprecated. Use CFNetwork instead; see *CFNetwork Programming Guide*.)

Getting and Setting Information About a URL

URLGetProperty (page 1747) Deprecated in Mac OS X v10.4
Obtains the value of a URL property. (Deprecated. Use CFNetwork instead; see *CFNetwork Programming Guide*.)

URLGetPropertySize (page 1748) Deprecated in Mac OS X v10.4
Determines the size of a URL property. (Deprecated. Use CFNetwork instead; see *CFNetwork Programming Guide*.)

URLSetProperty (page 1753) Deprecated in Mac OS X v10.4
Sets the value of a URL property. (Deprecated. Use CFNetwork instead; see *CFNetwork Programming Guide*.)

Performing Simple Data Transfer

URLDownload (page 1741) Deprecated in Mac OS X v10.4
Downloads data from a URL specified by a URL reference. (Deprecated. Use CFNetwork instead; see *CFNetwork Programming Guide*.)

URLSimpleDownload (page 1754) Deprecated in Mac OS X v10.4
Downloads data from a URL specified by a character string. (Deprecated. Use CFNetwork instead; see *CFNetwork Programming Guide*.)

URLSimpleUpload (page 1756) Deprecated in Mac OS X v10.4
Uploads a file or directory to an FTP URL specified by a character string. (Deprecated. Use CFNetwork instead; see *CFNetwork Programming Guide*.)

URLUpload (page 1757) Deprecated in Mac OS X v10.4
Uploads a file or directory to an FTP URL specified by a URL reference. (Deprecated. Use CFNetwork instead; see *CFNetwork Programming Guide*.)

Getting More Control Over Data Transfer Operations

URLGetBuffer (page 1743) Deprecated in Mac OS X v10.4
Obtains the next buffer of data in a download operation. (Deprecated. Use CFNetwork instead; see *CFNetwork Programming Guide*.)

URLGetDataAvailable (page 1745) Deprecated in Mac OS X v10.4
Determines the amount of data currently available for retrieval in a download operation. (Deprecated. Use CFNetwork instead; see *CFNetwork Programming Guide*.)

URLOpen (page 1751) Deprecated in Mac OS X v10.4
Opens a URL and starts an asynchronous download or upload operation. (Deprecated. Use CFNetwork instead; see *CFNetwork Programming Guide*.)

URLReleaseBuffer (page 1752) Deprecated in Mac OS X v10.4
Releases a buffer. (Deprecated. Use CFNetwork instead; see *CFNetwork Programming Guide*.)

Terminating Data Transfer Operations

`URLAbort` (page 1740) Deprecated in Mac OS X v10.4
: Terminates a data transfer operation. (Deprecated. Use CFNetwork instead; see *CFNetwork Programming Guide*.)

Getting Data Transfer Information

`URLGetCurrentState` (page 1744) Deprecated in Mac OS X v10.4
: Determines the status of a data transfer operation.

`URLGetError` (page 1746) Deprecated in Mac OS X v10.4
: Determines the error code of a failed data transfer operation. (Deprecated. Use CFNetwork instead; see *CFNetwork Programming Guide*.)

`URLGetFileInfo` (page 1746) Deprecated in Mac OS X v10.4
: Obtains the file type and creator of a file. (Deprecated. Use CFNetwork instead; see *CFNetwork Programming Guide*.)

`URLIdle` (page 1749) Deprecated in Mac OS X v10.4
: Gives the URL Access Manager time to refill its buffers during download operations. (Deprecated. Use CFNetwork instead; see *CFNetwork Programming Guide*.)

Working With URL Access Manager Callbacks

`DisposeURLNotifyUPP` (page 1736) Deprecated in Mac OS X v10.4
: Disposes of a UPP to your data transfer event notification callback. (Deprecated. Use CFNetwork instead; see *CFNetwork Programming Guide*.)

`DisposeURLSystemEventUPP` (page 1736) Deprecated in Mac OS X v10.4
: Disposes of a UPP to your system event notification callback. (Deprecated. Use CFNetwork instead; see *CFNetwork Programming Guide*.)

`InvokeURLNotifyUPP` (page 1737) Deprecated in Mac OS X v10.4
: Invokes your data transfer event notification callback. (Deprecated. Use CFNetwork instead; see *CFNetwork Programming Guide*.)

`InvokeURLSystemEventUPP` (page 1738) Deprecated in Mac OS X v10.4
: Invokes your system event notification callback. (Deprecated. Use CFNetwork instead; see *CFNetwork Programming Guide*.)

`NewURLNotifyUPP` (page 1738) Deprecated in Mac OS X v10.4
: Creates a UPP to your data transfer event notification callback. (Deprecated. Use CFNetwork instead; see *CFNetwork Programming Guide*.)

`NewURLSystemEventUPP` (page 1739) Deprecated in Mac OS X v10.4
: Creates a UPP to your system event notification callback. (Deprecated. Use CFNetwork instead; see *CFNetwork Programming Guide*.)

Functions

DisposeURLNotifyUPP

Disposes of a UPP to your data transfer event notification callback. (Deprecated in Mac OS X v10.4. Use CFNetwork instead; see *CFNetwork Programming Guide*.)

```
void DisposeURLNotifyUPP (
   URLNotifyUPP userUPP
);
```

Parameters

userUPP

A Universal Procedure Pointer (UPP) to your notification callback function.

Discussion

When you are finished with a UPP to your notification callback function, you should dispose of it by calling the `DisposeURLNotifyUPP` function.

Special Considerations

CFNetwork provides better reliability and performance and is used by Apple's own applications. URL Access Manager is no longer being enhanced or improved.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`URLAccess.h`

DisposeURLSystemEventUPP

Disposes of a UPP to your system event notification callback. (Deprecated in Mac OS X v10.4. Use CFNetwork instead; see *CFNetwork Programming Guide*.)

```
void DisposeURLSystemEventUPP (
   URLSystemEventUPP userUPP
);
```

Parameters

userUPP

A UPP to your system event callback function.

Discussion

When you are finished with a UPP to your system event callback function, you should dispose of it by calling the `DisposeURLSystemEventUPP` function.

Special Considerations

CFNetwork provides better reliability and performance and is used by Apple's own applications. URL Access Manager is no longer being enhanced or improved.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In
URLAccess.h

InvokeURLNotifyUPP

Invokes your data transfer event notification callback. (Deprecated in Mac OS X v10.4. Use CFNetwork instead; see *CFNetwork Programming Guide*.)

```
OSStatus InvokeURLNotifyUPP (
   void *userContext,
   URLEvent event,
   URLCallbackInfo *callbackInfo,
   URLNotifyUPP userUPP
);
```

Parameters

userContext

A pointer to application-defined storage. The URL Access Manager passes this value in the `userContext` parameter of your notification callback function. Your application can use this to set up its context when your data transfer event notification callback is called.

event

The data transfer events you want your application to receive. See "Data Transfer Event Constants" (page 1764) for a description of possible values. The URL Access Manager tests the bitmask you pass in the `eventRegister` parameter of the function `URLOpen` (page 1751) to determine which events to pass to your callback function. See "Data Transfer Event Mask Constants" (page 1766) for a description of this bitmask.

callbackInfo

A pointer to a structure of type `URLCallbackInfo` (page 1762) that provides information about the data transfer event to your callback function. The URL Access Manager passes a pointer to this structure in the `callbackInfo` parameter of your notification callback function.

userUPP

A Universal Procedure Pointer to your data transfer notification callback. For information on how to write this function, see `URLNotifyProcPtr` (page 1759).

Return Value
A result code. See "URL Access Manager Result Codes" (page 1779).

Discussion
The URL Access Manager calls the `InvokeURLNotifyUPP` function when you pass a UPP to your callback function in the `notifyProc` parameter of the function `URLOpen` (page 1751), and the data transfer event that you specified in the `eventRegister` parameter occurs.

Special Considerations

CFNetwork provides better reliability and performance and is used by Apple's own applications. URL Access Manager is no longer being enhanced or improved.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
URLAccess.h

InvokeURLSystemEventUPP

Invokes your system event notification callback. (Deprecated in Mac OS X v10.4. Use CFNetwork instead; see *CFNetwork Programming Guide*.)

```
OSStatus InvokeURLSystemEventUPP (
   void *userContext,
   EventRecord *event,
   URLSystemEventUPP userUPP
);
```

Parameters

userContext

A pointer to application-defined storage. The URL Access Manager passes this value in the `userContext` parameter of your system event callback function. Your application can use this to set up its context when your system event notification callback is called.

event

A pointer to an event record that provides information about the system event to your callback function.

userUPP

A Universal Procedure Pointer to your system event notification callback. For information on how to write this function, see `URLSystemEventProcPtr` (page 1760).

Return Value
A result code. See "URL Access Manager Result Codes" (page 1779).

Discussion
The URL Access Manager calls the `InvokeURLSystemEventUPP` function when you pass a UPP to your callback function in the eventProc parameter of the functions `URLSimpleDownload` (page 1754), `URLSimpleUpload` (page 1756), `URLDownload` (page 1741), or `URLUpload` (page 1757), and a system event occurs while a progress indicator or authentication dialog box is being displayed.

Special Considerations

CFNetwork provides better reliability and performance and is used by Apple's own applications. URL Access Manager is no longer being enhanced or improved.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
URLAccess.h

NewURLNotifyUPP

Creates a UPP to your data transfer event notification callback. (Deprecated in Mac OS X v10.4. Use CFNetwork instead; see *CFNetwork Programming Guide*.)

```
URLNotifyUPP NewURLNotifyUPP (
   URLNotifyProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your data transfer event notification callback. For information on how to write your callback, see URLNotifyProcPtr (page 1759).

Return Value

A UPP to your data transfer event notification callback. You can register your callback by passing this UPP in the notifyProc parameter of the function URLOpen (page 1751). See the description of the URLNotifyUPP data type.

Discussion

The NewURLNotifyUPP function creates a pointer to your data transfer event notification callback. You pass a pointer to your callback in the notifyProc parameter of the function URLOpen (page 1751) if you want your application to receive data transfer events. Pass a bitmask in the eventRegister parameter of URLOpen (page 1751) indicating which data transfer events you want to receive.

Special Considerations

CFNetwork provides better reliability and performance and is used by Apple's own applications. URL Access Manager is no longer being enhanced or improved.

When you are finished with your data transfer event notification callback, you should dispose of the UPP by calling the function DisposeURLNotifyUPP (page 1736).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

URLAccess.h

NewURLSystemEventUPP

Creates a UPP to your system event notification callback. (Deprecated in Mac OS X v10.4. Use CFNetwork instead; see *CFNetwork Programming Guide*.)

```
URLSystemEventUPP NewURLSystemEventUPP (
   URLSystemEventProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your system event notification callback. For information on how to write your callback, see URLSystemEventProcPtr (page 1760).

Return Value

A UPP to your system event notification callback. You can register your callback by passing this UPP in the eventProc parameter of the function URLSimpleDownload (page 1754), URLSimpleUpload (page 1756), URLDownload (page 1741), or URLUpload (page 1757). See the description of the URLSystemEventUPP data type.

Discussion

The `NewURLSystemEventUPP` function creates a pointer to your system event callback function. You pass a pointer to your callback function in the `eventProc` parameter of the functions `URLSimpleDownload` (page 1754), `URLSimpleUpload` (page 1756), `URLDownload` (page 1741), and `URLUpload` (page 1757) if you want update events to be passed to your application while a dialog box is displayed. (In Mac OS X, this is not necessary, since all dialog boxes are moveable). In order for these functions to display a dialog box, you must set the mask constant `kURLDisplayProgressFlag` or `kURLDisplayAuthFlag` in the bitmask passed in the `openFlags` parameter.

Special Considerations

CFNetwork provides better reliability and performance and is used by Apple's own applications. URL Access Manager is no longer being enhanced or improved.

When you are finished with your system event notification callback, you should dispose of the UPP by calling the function `DisposeURLNotifyUPP` (page 1736).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`URLAccess.h`

URLAbort

Terminates a data transfer operation. (Deprecated in Mac OS X v10.4. Use CFNetwork instead; see *CFNetwork Programming Guide*.)

```
OSStatus URLAbort (
   URLReference urlRef
);
```

Parameters

urlRef

A reference to the URL whose data transfer operation you wish to terminate.

Return Value

A result code. See "URL Access Manager Result Codes" (page 1779).

Discussion

The `URLAbort` function terminates any data transfer operation started by the functions `URLSimpleDownload` (page 1754), `URLDownload` (page 1741), `URLSimpleUpload` (page 1756), `URLUpload` (page 1757), or `URLOpen` (page 1751). When your application calls `URLAbort`, the URL Access Manager changes the state returned by the function `URLGetCurrentState` (page 1744) to `kURLAbortingState` and passes the constant `kURLAbortInitiatedEvent` to your notification callback function. When data transfer is terminated, the URL Access Manager changes the state returned by `URLGetCurrentState` (page 1744) to `kURLCompletedState` and passes the constant `kURLCompletedEvent` in the `event` parameter of your notification callback function.

Special Considerations

CFNetwork provides better reliability and performance and is used by Apple's own applications. URL Access Manager is no longer being enhanced or improved.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`URLAccess.h`

URLDisposeReference

Disposes of the memory associated with a URL reference. (Deprecated in Mac OS X v10.4. Use CFNetwork instead; see *CFNetwork Programming Guide*.)

```
OSStatus URLDisposeReference (
   URLReference urlRef
);
```

Parameters

urlRef

A reference to the URL whose associated memory you wish to dispose of. You should call the `URLDisposeReference` function to release the memory occupied by a URL reference when you are finished with it.

Return Value
A result code. See "URL Access Manager Result Codes" (page 1779).

Special Considerations

CFNetwork provides better reliability and performance and is used by Apple's own applications. URL Access Manager is no longer being enhanced or improved.

You must call the `URLDisposeReference` function to dispose of the reference associated with a URL reference even if the data transfer operation fails. Failure to call `URLDisposeReference` may result in thread or memory leaks.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`URLAccess.h`

URLDownload

Downloads data from a URL specified by a URL reference. (Deprecated in Mac OS X v10.4. Use CFNetwork instead; see *CFNetwork Programming Guide*.)

```
OSStatus URLDownload (
   URLReference urlRef,
   FSSpec *destination,
   Handle destinationHandle,
   URLOpenFlags openFlags,
   URLSystemEventUPP eventProc,
   void *userContext
);
```

Parameters

urlRef

A reference to the URL from which data is to be downloaded. Once you call `URLDownload`, you cannot use the same reference if you call `URLDownload` again. Instead, you must create a new URL reference by calling the function `URLNewReference` (page 1750).

destination

A pointer to a file specification structure that identifies the file or directory into which data is to be downloaded. If you wish to download data into memory, pass `NULL` in this parameter and a valid handle in the `destinationHandle` parameter. If you pass a file specification that does not identify a file or directory, the name of the file or directory specified by the pathname in the `urlRef` parameter is used. If you pass a file or directory that already exists, and do not specify `kURLReplaceExistingFlag` in the `openFlags` parameter, `URLDownload` creates a new file or directory whose name has a number appended before the extension. For example, if the URL specifies a file named `file.txt`, `URLDownload` changes the filename to `file1.txt`.

destinationHandle

A handle to the destination in memory where you want the data downloaded. Before calling `URLDownload`, create a zero-sized handle. If you wish to download data into a file or directory, pass `NULL` in this parameter and a valid file specification in the `destination` parameter.

openFlags

A bitmask that indicates the data transfer options to use. You can specify any of the following masks for downloading options: `kURLReplaceExistingFlag`, `kURLExpandFileFlag`, `kURLExpandAndVerifyFlag`, `kURLDisplayProgressFlag`, `kURLDisplayAuthFlag`, `kURLIsDirectoryHintFlag`, `kURLDoNotTryAnonymousFlag`, `kURLDebinhexOnlyFlag`, `kURLNoAutoRedirect`, and `kURLDirectoryListingFlag`. See "Data Transfer Options Mask Constants" (page 1770) for a description of possible values.

eventProc

A Universal Procedure Pointer (UPP) to your system event callback function, if one exists. For information on how to write a system event callback, see `URLSystemEventProcPtr` (page 1760). If you want to handle events that occur while a progress indicator or authentication dialog box is being displayed, specify the appropriate mask (either `kURLDisplayProgressFlag` or `kURLDisplayAuthFlag`) in the `openFlags` parameter and pass a UPP to your callback function in this parameter. Pass `NULL` if you do not want to receive notification of these events. In this case, the URL Access Manager displays a nonmovable modal progress indicator or authentication dialog box.

userContext

A pointer to application-defined storage that will be passed to your system event callback function, if one exists. Your application can use this to set up its context when your system event callback function is called.

Return Value

A result code. See "URL Access Manager Result Codes" (page 1779). If your application is multi-threaded, and more than one thread calls `URLDownload` simultaneously, `URLDownload` returns the result code `kURLProgessAlreadyDisplayedError` if you specify `kURLDisplayProgressFlag` in the `openFlags` parameter and the URL Access Manager is already displaying a progress indicator.

Discussion
The `URLDownload` function downloads data from a URL specified by a URL reference to a file, directory, or memory. It does not return until the download is complete. If you want to download data from a URL identified by a URL string rather than a reference, call the function `URLSimpleDownload` (page 1754). The difference between the two functions is that URLDownload allows you to access other URL Access Manager functions before, after, or during the download. If you want more control over a data transfer operation, call the function `URLOpen` (page 1751).

If you wish to download data to a file or directory, pass a valid file specification in the `destination` parameter. If you instead wish to download data to memory, pass a valid handle in the `destinationHandle` parameter. If the URL specified in the `urlRef` parameter points to a file, the file is downloaded regardless of whether the bit specified by the mask constant `kURLDirectoryListingFlag` or `KURLIsDirectoryHintFlag` is set in the `openFlags` parameter.

When `URLDownload` downloads data from a URL that represents a local file (that is, a URL that begins with `file://`), the data fork is downloaded but the resource fork is not.

Special Considerations

CFNetwork provides better reliability and performance and is used by Apple's own applications. URL Access Manager is no longer being enhanced or improved.

`URLDownload` yields time to other threads. Your application should call `URLDownload` from a thread other than the main thread so that other processes have time to run.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`URLAccess.h`

URLGetBuffer

Obtains the next buffer of data in a download operation. (Deprecated in Mac OS X v10.4. Use CFNetwork instead; see *CFNetwork Programming Guide*.)

```
OSStatus URLGetBuffer (
   URLReference urlRef,
   void **buffer,
   Size *bufferSize
);
```

Parameters

urlRef
A reference to the URL whose next buffer you wish to obtain.

buffer
On return, a handle to a buffer containing the downloaded data.

bufferSize
On return, a pointer to the number of bytes of data in the buffer.

Return Value
A result code. See "URL Access Manager Result Codes" (page 1779).

Discussion
The `URLGetBuffer` function obtains the next buffer of data in a download operation. `URLGetBuffer` does not enable you to retain or modify the transferred data. If you pass `NULL` in the `fileSpec` parameter of the function `URLOpen` (page 1751), you should call `URLGetBuffer` to retrieve data as it is downloaded.

You should call `URLGetBuffer` repeatedly until URL Access Manager passes the event constant `kURLCompletedEvent` or `kURLAbortInitiatedEvent` in the `event` parameter of your notification callback function, or until the function `URLGetCurrentState` (page 1744) returns the state constant `kURLTransactionComplete` or `kURLAbortingState`. Between calls to `URLGetBuffer`, you should call the function `URLIdle` (page 1749) to allow time for the URL Access Manager to refill its buffers.

To determine the number of bytes remaining in the buffer, call the function `URLGetDataAvailable` (page 1745). The size returned by `URLGetDataAvailable` (page 1745) does not include the number of bytes in transit to a buffer, nor does it include the amount of data not yet transferred from the URL.

Special Considerations

CFNetwork provides better reliability and performance and is used by Apple's own applications. URL Access Manager is no longer being enhanced or improved.

You should release the returned buffer as soon as possible after a call to `URLGetBuffer` by calling the function `URLReleaseBuffer` (page 1752). This prevents the URL Access Manager from running out of buffers.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`URLAccess.h`

URLGetCurrentState

Determines the status of a data transfer operation. (Deprecated in Mac OS X v10.4.)

```
OSStatus URLGetCurrentState (
   URLReference urlRef,
   URLState *state
);
```

Parameters

urlRef
A reference to the URL whose data transfer state you want to determine.

state
On return, a pointer to the state of data transfer. See "Data Transfer State Constants" (page 1773) for a description of possible values.

Return Value
A result code. See "URL Access Manager Result Codes" (page 1779).

Discussion
The `URLGetCurrentState` function determines the current status of a data transfer operation. You may wish to call `URLGetCurrentState` periodically to monitor the status of a download or upload operation.

Special Considerations

CFNetwork provides better reliability and performance and is used by Apple's own applications. URL Access Manager is no longer being enhanced or improved.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

URLAccess.h

URLGetDataAvailable

Determines the amount of data currently available for retrieval in a download operation. (Deprecated in Mac OS X v10.4. Use CFNetwork instead; see *CFNetwork Programming Guide*.)

```
OSStatus URLGetDataAvailable (
   URLReference urlRef,
   Size *dataSize
);
```

Parameters

urlRef

A reference to the URL for which you wish to determine the amount od data currently available for retrieval in a download operation.

dataSize

On return, a pointer to the size (in bytes) of data available for retrieval in a download operation.

Return Value

A result code. See "URL Access Manager Result Codes" (page 1779).

Discussion

The URLGetDataAvailable (page 1745) function determines the amount of data remaining in the buffer of the URL Access Manager that you will obtain from a call to the function URLGetBuffer (page 1743). You should only call this function if you passed an invalid destination file to the function URLOpen (page 1751). This does not include the number of bytes in transit to your buffer, nor does it include the amount of data not yet transferred from the URL Access Manager. To calculate the amount of data remaining to be downloaded, pass the name constant kURLResourceSize in the property parameter of the function URLGetProperty (page 1747) and subtract the amount of data copied.

Special Considerations

CFNetwork provides better reliability and performance and is used by Apple's own applications. URL Access Manager is no longer being enhanced or improved.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

URLAccess.h

URLGetError

Determines the error code of a failed data transfer operation. (Deprecated in Mac OS X v10.4. Use CFNetwork instead; see *CFNetwork Programming Guide*.)

```
OSStatus URLGetError (
   URLReference urlRef,
   OSStatus *urlError
);
```

Parameters

urlRef

A reference to the URL whose data transfer operation failed.

urlError

A pointer to a C string representing the name of the error code returned by the failed operation.

Return Value

A result code. See "URL Access Manager Result Codes" (page 1779).

Discussion

The `URLGetError` function determines the error code returned when a data transfer operation fails. The error code may be a system error code, a protocol-specific error code, or one of the error codes listed in "URL Access Manager Result Codes" (page 1779).

Special Considerations

CFNetwork provides better reliability and performance and is used by Apple's own applications. URL Access Manager is no longer being enhanced or improved.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`URLAccess.h`

URLGetFileInfo

Obtains the file type and creator of a file. (Deprecated in Mac OS X v10.4. Use CFNetwork instead; see *CFNetwork Programming Guide*.)

```
OSStatus URLGetFileInfo (
   StringPtr fName,
   OSType *fType,
   OSType *fCreator
);
```

Parameters

fName

A pointer to a Pascal string representing the name of the file for which you want information.

fType

On return, a pointer to the file type code of the specified filename.

fCreator

On return, a pointer to the file creator code of the specified filename.

Return Value
A result code. See "URL Access Manager Result Codes" (page 1779).

Discussion
The `URLGetFileInfo` function obtains the file type and creator codes for a specified filename. The type and creator codes are determined by the Internet configuration mapping table and are based on the filename extension. For example, if you pass the filename `"jane.txt"`, `URLGetFileInfo` will return `'TEXT'` in the `type` parameter and `'ttxt'` in the `creator` parameter.

Special Considerations

CFNetwork provides better reliability and performance and is used by Apple's own applications. URL Access Manager is no longer being enhanced or improved.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`URLAccess.h`

URLGetProperty

Obtains the value of a URL property. (Deprecated in Mac OS X v10.4. Use CFNetwork instead; see *CFNetwork Programming Guide*.)

```
OSStatus URLGetProperty (
   URLReference urlRef,
   const char *property,
   void *propertyBuffer,
   Size bufferSize
);
```

Parameters

urlRef

A reference to the URL whose property value you want to determine.

property

A pointer to a C string representing the name of the property value you want to determine. For a description of property name constants and their corresponding data types, see "Universal URL Property Name Constants" (page 1776) and "HTTP and HTTPS URL Property Name Constants" (page 1775).

propertyBuffer

A pointer to a buffer containing the property value you want to obtain. You must also pass the correct data type of the property value you wish to obtain. Before calling `URLGetProperty`, allocate enough memory in this buffer to contain the property value you wish to obtain. On return, a pointer to a buffer containing the property value. If you do not allocate enough memory for the buffer, `URLGetProperty` does not pass back the property value in this parameter and returns the result code `kURLPropertyBufferTooSmallError`.

bufferSize

The size (in bytes) of the buffer pointed to by `propertyBuffer`. To determine the buffer size, call the function `URLGetPropertySize` (page 1748). If the buffer size is too small, `URLGetProperty` returns the result code `kURLPropertyBufferTooSmallError` and does not pass back the property value in the `propertyBuffer` parameter.

Return Value
A result code. See "URL Access Manager Result Codes" (page 1779). The result code `kURLPropertyBufferTooSmallError` indicates that you did not allocate enough memory for the buffer in the `propertyBuffer` parameter. The result code `kURLPropertyNotYetKnownError` indicates that the value of the property is not yet available.

Discussion
The `URLGetProperty` function obtains the value of a URL property identified by the property name constant specified in the property parameter.

Special Considerations

CFNetwork provides better reliability and performance and is used by Apple's own applications. URL Access Manager is no longer being enhanced or improved.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`URLAccess.h`

URLGetPropertySize

Determines the size of a URL property. (Deprecated in Mac OS X v10.4. Use CFNetwork instead; see *CFNetwork Programming Guide*.)

```
OSStatus URLGetPropertySize (
   URLReference urlRef,
   const char *property,
   Size *propertySize
);
```

Parameters

urlRef

A reference to the URL whose property size you want to determine.

property

A pointer to a C string representing the name of the property value whose size you want to determine. For a description of property name constants, see "Universal URL Property Name Constants" (page 1776) and "HTTP and HTTPS URL Property Name Constants" (page 1775).

propertySize

On return, a pointer to the size (in bytes) of the specified property value. If the size is not available, `URLGetPropertySize` passes back `-1` in this parameter and returns the result code `kURLPropertyNotYetKnownError`.

Return Value
A result code. See "URL Access Manager Result Codes" (page 1779).

Discussion
The `URLGetProperty` function obtains the size of the property value identified by the property name constant passed in the property parameter. For a description of property name constants and data types of the corresponding property values, see "Universal URL Property Name Constants" (page 1776) and "HTTP and HTTPS URL Property Name Constants" (page 1775).

You should call the `URLGetPropertySize` function before calling the function `URLGetProperty` (page 1747) to determine the size of the buffer containing the property value you wish to obtain. Pass the value passed back in the `propertySize` parameter in the `bufferSize` parameter of `URLGetProperty` (page 1747).

Special Considerations

CFNetwork provides better reliability and performance and is used by Apple's own applications. URL Access Manager is no longer being enhanced or improved.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`URLAccess.h`

URLGetURLAccessVersion

Determines the version of URL Access Manager installed on the user's system. (Deprecated in Mac OS X v10.4. Use CFNetwork instead; see *CFNetwork Programming Guide*.)

```
OSStatus URLGetURLAccessVersion (
   UInt32 *returnVers
);
```

Parameters

returnVers
On return, a pointer to the version number of the URL Access Manager installed on the user's system.

Return Value
A result code. See "URL Access Manager Result Codes" (page 1779).

Special Considerations

CFNetwork provides better reliability and performance and is used by Apple's own applications. URL Access Manager is no longer being enhanced or improved.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`URLAccess.h`

URLIdle

Gives the URL Access Manager time to refill its buffers during download operations. (Deprecated in Mac OS X v10.4. Use CFNetwork instead; see *CFNetwork Programming Guide*.)

```
OSStatus URLIdle (
   void
);
```

Return Value
A result code. See "URL Access Manager Result Codes" (page 1779).

Discussion
The `URLIdle` function gives the URL Access Manager time to refill its buffers during download operations. You should call URLIdle periodically after you call the function `URLOpen` (page 1751) to allow time for the URL Access Manager to refill its buffers.

Special Considerations

CFNetwork provides better reliability and performance and is used by Apple's own applications. URL Access Manager is no longer being enhanced or improved.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`URLAccess.h`

URLNewReference

Creates a URL reference. (Deprecated in Mac OS X v10.4. Use CFNetwork instead; see *CFNetwork Programming Guide*.)

```
OSStatus URLNewReference (
   const char *url,
   URLReference *urlRef
);
```

Parameters

url

A pointer to a C string representing the name of the URL you want to create.

urlRef

On return, a pointer to the newly-created URL reference.

Return Value
A result code. See "URL Access Manager Result Codes" (page 1779).

Discussion
The `URLNewReference` function creates a URL reference that you can use in subsequent calls to the URL Access Manager. When you no longer need a URL reference, you should dispose of its memory by calling the function `URLDisposeReference` (page 1741).

Special Considerations

CFNetwork provides better reliability and performance and is used by Apple's own applications. URL Access Manager is no longer being enhanced or improved.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
URLAccess.h

URLOpen

Opens a URL and starts an asynchronous download or upload operation. (Deprecated in Mac OS X v10.4. Use CFNetwork instead; see *CFNetwork Programming Guide*.)

```
OSStatus URLOpen (
   URLReference urlRef,
   FSSpec *fileSpec,
   URLOpenFlags openFlags,
   URLNotifyUPP notifyProc,
   URLEventMask eventRegister,
   void *userContext
);
```

Parameters

urlRef

A reference to the URL to or from which you wish to transfer data. You cannot use the same reference if you call URLOpen again. Instead, you must create a new URL reference by calling the function URLNewReference (page 1750). If the URL refers to a file, the file is downloaded regardless of whether you specify kURLDirectoryListingFlag or KURLIsDirectoryHintFlag in the openFlags parameter. See "Naming Your Destination File" for more information.

fileSpec

A pointer to a file specification that identifies the file or directory from which data is to be uploaded or downloaded. For upload operations, you must pass a valid file specification. For download operations, you can pass NULL. In this case, you must call the function URLGetBuffer (page 1743) to retrieve the data as it is downloaded. For more information, see the function discussion.

openFlags

A bitmask that indicates the data transfer options to use. You can specify any of the following masks for upload operations: kURLUploadFlag, kURLReplaceExistingFlag, kURLBinHexFileFlag, and kURLDoNotTryAnonymousFlag. You can specify any of the following masks for download operations: kURLReplaceExistingFlag, kURLIsDirectoryHintFlag, kURLDoNotTryAnonymousFlag, kURLDebinhexOnlyFlag, kURLNoAutoRedirect, and kURLDirectoryListingFlag. See "Data Transfer Options Mask Constants" (page 1770) for a description of possible values.

notifyProc

A Universal Procedure Pointer (UPP) to a data transfer event notification callback, as described in URLNotifyProcPtr (page 1759). You should create a notification callback function if you wish to receive notification of certain data transfer events. In this case, you should also pass a bitmask of the events you wish to receive in the eventRegister parameter. The data transfer events that you receive will vary depending upon whether the destination file you specify is valid. Pass NULL if you do not want to receive notification of data transfer events.

eventRegister

A bitmask that URLOpen will test to determine the data transfer events that you wish to receive notification of. To receive data transfer events, you should also pass a UPP to your callback in the notifyProc parameter. See "Data Transfer Event Mask Constants" (page 1766) for a description of this mask.

userContext
A pointer to application-defined storage that will be passed to your notification callback function. Your application can use this to set up its context when your notification callback function is called.

Return Value
A result code. See "URL Access Manager Result Codes" (page 1779).

Discussion
The `URLOpen` function starts a download or upload operation and returns control to your application immediately. For download operations, you do not have to specify a valid destination file. In this case, you should call the function `URLGetBuffer` (page 1743) repeatedly to get the next buffer of data. Between calls to `URLGetBuffer` (page 1743), you should call the function `URLIdle` (page 1749) to allow time for the URL Access Manager to refill its buffers during download operations. After each call to `URLGetBuffer` (page 1743), you call the function `URLReleaseBuffer` (page 1752) to prevent the URL Access Manager from running out of buffers. You can call the function `URLGetDataAvailable` (page 1745) to determine the amount of data remaining in the buffer of the URL Access Manager that you will obtain from a call to the function `URLGetBuffer` (page 1743).

If you pass a valid destination file, you should not call the functions `URLGetBuffer` (page 1743), `URLReleaseBuffer` (page 1752), or `URLGetDataAvailable` (page 1745).

If you wish to be notified of certain data transfer events, you can specify a data transfer event callback and pass a pointer to it in the `URLEventMask` parameter of URLOpen. The data transfer events that you receive will vary depending upon whether the destination file you specify is valid. In addition, you should pass a bitmask representing the events you wish to be notified of in the `eventRegister` parameter.

When `URLOpen` downloads data from a URL that represents a local file (that is, a URL that begins with `file://`), the data fork is downloaded but the resource fork is not.

Special Considerations

CFNetwork provides better reliability and performance and is used by Apple's own applications. URL Access Manager is no longer being enhanced or improved.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`URLAccess.h`

URLReleaseBuffer

Releases a buffer. (Deprecated in Mac OS X v10.4. Use CFNetwork instead; see *CFNetwork Programming Guide*.)

```
OSStatus URLReleaseBuffer (
   URLReference urlRef,
   void *buffer
);
```

Parameters

urlRef
A reference to the URL whose buffer you want to release.

buffer
A pointer to the buffer you want to release.

Return Value
A result code. See "URL Access Manager Result Codes" (page 1779).

Discussion
The `URLReleaseBuffer` function releases the buffer obtained by calling the function `URLGetBuffer` (page 1743). To prevent the URL Access Manager from running out of buffers, you should call `URLReleaseBuffer` after each call to `URLGetBuffer` (page 1743).

Special Considerations

CFNetwork provides better reliability and performance and is used by Apple's own applications. URL Access Manager is no longer being enhanced or improved.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`URLAccess.h`

URLSetProperty

Sets the value of a URL property. (Deprecated in Mac OS X v10.4. Use CFNetwork instead; see *CFNetwork Programming Guide*.)

```
OSStatus URLSetProperty (
   URLReference urlRef,
   const char *property,
   void *propertyBuffer,
   Size bufferSize
);
```

Parameters

urlRef
A reference to the URL whose property value you want to set.

property
A pointer to a C string representing the name of the property value you want to set. You can only set property values identified by the constants `kURLPassword`, `kURLUserName`, `kURLHTTPRequestMethod`, `kURLHTTPRequestHeader`, `kURLHTTPRequestBody`, and `kURLHTTPUserAgent`. For a description of these property name constants and their corresponding data types, see "Universal URL Property Name Constants" (page 1776) and "HTTP and HTTPS URL Property Name Constants" (page 1775).

propertyBuffer
A pointer to a buffer containing the data you would like the property to be set to. The data must be of the correct type.

bufferSize
The size (in bytes) of the data you want to set.

Return Value
A result code. See "URL Access Manager Result Codes" (page 1779). The result code `kURLUnsettablePropertyError` indicates that a property value cannot be set.

Discussion

The `URLSetProperty` function enables you to set those property values identified by the following constants: `kURLPassword`, `kURLUserName`, `kURLPassword`, `kURLHTTPRequestMethod`, `kURLHTTPRequestHeader`, `kURLHTTPRequestBody`, and `kURLHTTPUserAgent`. For a description of these property name constants and their corresponding data types, see "Universal URL Property Name Constants" (page 1776) and "HTTP and HTTPS URL Property Name Constants" (page 1775).

You may wish to call `URLSetProperty` before calling the function `URLDownload` (page 1741) or `URLUpload` (page 1757) to set a URL property before a data transfer operation.

Special Considerations

CFNetwork provides better reliability and performance and is used by Apple's own applications. URL Access Manager is no longer being enhanced or improved.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`URLAccess.h`

URLSimpleDownload

Downloads data from a URL specified by a character string. (Deprecated in Mac OS X v10.4. Use CFNetwork instead; see *CFNetwork Programming Guide*.)

```
OSStatus URLSimpleDownload (
   const char *url,
   FSSpec *destination,
   Handle destinationHandle,
   URLOpenFlags openFlags,
   URLSystemEventUPP eventProc,
   void *userContext
);
```

Parameters

url

A pointer to a C string representing the pathname of the URL from which data is to be downloaded. If the pathname specifies a file, the file is downloaded regardless of whether you specify `kURLDirectoryListingFlag` or `KURLIsDirectoryHintFlag` in the `openFlags` parameter.

destination

A pointer to a file specification structure that identifies the file or directory into which data is to be downloaded. If you wish to download data into memory, pass `NULL` in this parameter and a valid handle in the `destinationHandle` parameter. If you pass a file specification that does not identify a file or directory, the name of the file or directory specified by the pathname in the `url` parameter is used. If you pass a file or directory that already exists, and do not specify `kURLReplaceExistingFlag` in the `openFlags` parameter, `URLSimpleDownload` creates a new file or directory whose name has a number appended before the extension. For example, if the URL specifies a file named `file.txt`, `URLSimpleDownload` changes the filename to `file1.txt`.

destinationHandle

A handle to the destination in memory where you want the data downloaded. Before calling `URLDownload`, create a zero-sized handle. If you wish to download data into a file or directory, pass `NULL` in this parameter and a valid file specification in the `destination` parameter.

openFlags

A bitmask that indicates the data transfer options to use. You can specify any of the following masks for downloading options: `kURLReplaceExistingFlag`, `kURLExpandFileFlag`, `kURLExpandAndVerifyFlag`, `kURLDisplayProgressFlag`, `kURLDisplayAuthFlag`, `kURLIsDirectoryHintFlag`, `kURLDoNotTryAnonymousFlag`, `kURLDebinhexOnlyFlag`, `kURLNoAutoRedirect`, and `kURLDirectoryListingFlag`. See "Data Transfer Options Mask Constants" (page 1770) for a description of possible values.

eventProc

A Universal Procedure Pointer (UPP) to your system event callback function, if one exists. For information on how to write a system event callback, see `URLSystemEventProcPtr` (page 1760). If you want to handle events that occur while a progress indicator or authentication dialog box is being displayed, specify the appropriate mask (either `kURLDisplayProgressFlag` or `kURLDisplayAuthFlag`) in the `openFlags` parameter and pass a UPP to your callback function in this parameter. Pass `NULL` if you do not want to receive notification of these events. In this case, the URL Access Manager displays a nonmovable modal progress indicator or authentication dialog box.

userContext

A pointer to application-defined storage that will be passed to your system event callback function, if one exists. Your application can use this to set up its context when your system event callback function is called.

Return Value

A result code. See "URL Access Manager Result Codes" (page 1779). If your application is multi-threaded, and more than one thread calls `URLSimpleDownload` simultaneously, `URLSimpleDownload` returns the result code `kURLProgressAlreadyDisplayedError` if you specify `kURLDisplayProgressFlag` in the `openFlags` parameter and the URL Access Manager is already displaying a progress indicator.

Discussion

The `URLSimpleDownload` function downloads data from a URL specified by a pathname to a specified file, directory, or memory. It does not return until the download is complete. If you want to download data from a URL identified by a reference rather than a pathname, call the function `URLDownload` (page 1741). The difference between the two functions is that `URLDownload` (page 1741) allows you to access other URL Access Manager functions before, after, or during the download. If you want more control over a data transfer operation, call the function `URLOpen` (page 1751).

If you wish to download data to a file or directory, pass a valid file specification in the `destination` parameter. If you instead wish to download data to memory, pass a valid handle in the `destinationHandle` parameter.

When `URLSimpleDownload` downloads data from a URL that represents a local file (that is, a URL that begins with `file://`), the data fork is downloaded but the resource fork is not.

Special Considerations

CFNetwork provides better reliability and performance and is used by Apple's own applications. URL Access Manager is no longer being enhanced or improved.

`URLSimpleDownload` yields time to other threads. Your application should call URLSimpleDownload from a thread other than the main thread so that other processes have time to run.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In
URLAccess.h

URLSimpleUpload

Uploads a file or directory to an FTP URL specified by a character string. (Deprecated in Mac OS X v10.4. Use CFNetwork instead; see *CFNetwork Programming Guide*.)

```
OSStatus URLSimpleUpload (
   const char *url,
   const FSSpec *source,
   URLOpenFlags openFlags,
   URLSystemEventUPP eventProc,
   void *userContext
);
```

Parameters

url

A pointer to a C string representing the URL to which a file or directory is to be uploaded. If you wish to replace the destination directory of this URL with the file or directory that you pass in the source parameter, terminate the string with a slash character (/), and set the mask constant kURLReplaceExistingFlag in the openFlags parameter. If you specify a name that already exists on the server and do not specify kURLReplaceExistingFlag, URLSimpleUpload returns the result code kURLDestinationExistsError. If you do not specify a name, do not specify kURLReplaceExistingFlag in the openFlags parameter, and the name already exists on the server, the URL Access Manager creates a unique name by appending a number to the original name before the extension, if any. For example, if the URL specifies a file named file.txt, URLSimpleUpload changes the filename to file1.txt. See "Naming Your Destination File" for more information.

source

A pointer to a file specification structure that describes the file or directory you want to upload.

openFlags

A bitmask that indicates the data transfer options to use. You can specify any of the following masks for uploading options: kURLReplaceExistingFlag, kURLBinHexFileFlag, kURLDisplayProgressFlag, kURLDisplayAuthFlag, and kURLDoNotTryAnonymousFlag. See "Data Transfer Options Mask Constants" (page 1770) for a description of possible values.

eventProc

A Universal Procedure Pointer (UPP) to your system event callback function, if one exists. For information on how to write a system event callback, see URLSystemEventProcPtr (page 1760). If you want to handle events that occur while a progress indicator or authentication dialog box is being displayed, specify the appropriate mask (either kURLDisplayProgressFlag or kURLDisplayAuthFlag) in the openFlags parameter and pass a UPP to your callback function in this parameter. Pass NULL if you do not want to receive notification of these events. In this case, the URL Access Manager displays a nonmovable modal progress indicator or authentication dialog box.

userContext

A pointer to application-defined storage that will be passed to your system event callback function, if one exists. Your application can use this to set up its context when your system event callback function is called.

Return Value
A result code. See "URL Access Manager Result Codes" (page 1779). The result code `kURLDestinationExistsError` indicates that you specified a pathname that already exists on the server but did not set the bit specified by the mask constant `kURLReplaceExistingFlag` in the `openFlags` parameter. If your application is multi-threaded, and more than one thread calls `URLSimpleUpload` simultaneously, `URLSimpleUpload` returns the result code `kURLProgessAlreadyDisplayedError` if you specify `kURLDisplayProgressFlag` in the `openFlags` parameter and the URL Access Manager is already displaying a progress indicator.

Discussion
The `URLSimpleUpload` function uploads a file or directory to an FTP URL specified by a pathname. It does not return until the upload is complete. If you want to upload data from a URL identified by a reference rather than a pathname, call the function `URLUpload` (page 1757). The difference between the two functions is that `URLUpload` (page 1757) allows you to access other URL Access Manager functions before, after, or during the download. If you want more control over a data transfer operation, call the function `URLOpen` (page 1751).

Special Considerations

CFNetwork provides better reliability and performance and is used by Apple's own applications. URL Access Manager is no longer being enhanced or improved.

`URLSimpleUpload` yields time to other threads. Your application should call `URLSimpleUpload` from a thread other than the main thread so that other processes have time to run.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`URLAccess.h`

URLUpload

Uploads a file or directory to an FTP URL specified by a URL reference. (Deprecated in Mac OS X v10.4. Use CFNetwork instead; see *CFNetwork Programming Guide*.)

```
OSStatus URLUpload (
   URLReference urlRef,
   const FSSpec *source,
   URLOpenFlags openFlags,
   URLSystemEventUPP eventProc,
   void *userContext
);
```

Parameters

urlRef

A reference to the URL to which a file or directory is to be uploaded. Once you have called `URLUpload`, you cannot use the same reference again. If you wish to replace the destination directory of this URL with the file or directory that you pass in the source parameter, set the mask constant `kURLReplaceExistingFlag` in the `openFlags` parameter. If you specify a name that already exists on the server and do not specify `kURLReplaceExistingFlag`, `URLUpload` returns the result code `kURLDestinationExistsError`. If you do not specify a name, do not specify `kURLReplaceExistingFlag` in the `openFlags` parameter, and the name already exists on the server, the URL Access Manager creates a unique name by appending a number to the original name before the extension, if any. For example, if the URL specifies a file named `file.txt`, `URLUpload` changes the filename to `file1.txt`. See "Naming Your Destination File" for more information.

source

A pointer to a file specification structure that describes the file or directory you want to upload.

openFlags

A bitmask that indicates the data transfer options you want used. You can specify any of the following masks for uploading options: `kURLReplaceExistingFlag`, `kURLBinHexFileFlag`, `kURLDisplayProgressFlag`, `kURLDisplayAuthFlag`, and `kURLDoNotTryAnonymousFlag`. See "URL Access Manager Reference" (page 1736) for a description of possible values.

eventProc

A Universal Procedure Pointer (UPP) to your system event callback function, if one exists. For information on how to write a system event callback, see `URLSystemEventProcPtr` (page 1760). If you want to handle events that occur while a progress indicator or authentication dialog box is being displayed, specify the appropriate mask (either `kURLDisplayProgressFlag` or `kURLDisplayAuthFlag`) in the `openFlags` parameter and pass a UPP to your callback function in this parameter. Pass `NULL` if you do not want to receive notification of these events. In this case, the URL Access Manager displays a nonmovable modal progress indicator or authentication dialog box.

userContext

A pointer to application-defined storage that will be passed to your system event callback function, if one exists. Your application can use this to set up its context when your system event callback function is called.

Return Value

A result code. See "URL Access Manager Result Codes" (page 1779). The result code `kURLDestinationExistsError` indicates that you specified a pathname that already exists on the server but did not specify `kURLReplaceExistingFlag` in the `openFlags` parameter. If your application is multi-threaded, and more than one thread calls `URLUpload` simultaneously, `URLUpload` returns the result code `kURLProgessAlreadyDisplayedError` if you specify `kURLDisplayProgressFlag` in the `openFlags` parameter and the URL Access Manager is already displaying a progress indicator.

Discussion
The `URLUpload` function uploads a file or directory to an FTP URL specified by a URL reference. It does not return until the upload is complete. If you want to upload data from a URL identified by a pathname rather than a reference, call the function `URLSimpleUpload` (page 1756). The difference between the two functions is that URLUpload allows you to access other URL Access Manager functions before, after, or during the download. If you want more control over a data transfer operation, call the function `URLOpen` (page 1751).

Special Considerations

CFNetwork provides better reliability and performance and is used by Apple's own applications. URL Access Manager is no longer being enhanced or improved.

`URLUpload` yields time to other threads. Your application should call `URLUpload` from a thread other than the main thread so that other processes have time to run.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`URLAccess.h`

Callbacks

URLNotifyProcPtr

Defines a pointer to your notification callback function that handles certain data transfer events that occur during data transfer operations.

```
typedef OSStatus (*URLNotifyProcPtr)
(
    void * userContext,
    URLEvent event,
    URLCallbackInfo * callbackInfo
);
```

If you name your function `MyURLNotifyProc`, you would declare it like this:

```
OSStatus MyURLNotifyProc (
    void * userContext,
    URLEvent event,
    URLCallbackInfo * callbackInfo
);
```

Parameters

`userContext`

A pointer to application-defined storage that your application previously passed to the function `URLOpen` (page 1751). Your application can use this to set up its context when your notification callback function is called.

event

The data transfer event that your application wishes to be notified of. See "Data Transfer Event Constants" (page 1764) for a description of possible values. The type of event that can trigger your callback depends on the event mask you passed in the `eventRegister` parameter of the function `URLOpen` (page 1751), and whether you pass a valid file specification in the `fileSpec` parameter of `URLOpen` (page 1751). For more information, see the discussion.

callbackInfo

A pointer to a structure of type `URLCallbackInfo` (page 1762). On return, the structure contains information about the data transfer event that occurred. The URL Access Manager passes this information to your callback function via the `callbackInfo` parameter of the function `InvokeURLNotifyUPP` (page 1737).

Return Value

A result code. See "URL Access Manager Result Codes" (page 1779). Your notification callback function should process the data transfer event and return `noErr`.

Discussion

Your notification callback function handles certain data transfer events that occur during data transfer operations performed by the function `URLOpen` (page 1751). You can define an event notification function and the events for which you want to receive notification only if you do not specify a file in which to store the data for download operations. In order to be notified of these events, you must pass a UPP to your notification callback function in the `notifyProc` parameter. You indicate the type of data transfer events you want to receive via a bitmask in the `eventRegister` parameter.

Note that if you pass a valid file specification to `URLOpen` (page 1751), your callback function will not be notified of data available and transaction completed events as identified by the constants `kURLDataAvailableEvent` and `kURLTransactionCompleteEvent`. If you pass a valid file specification to `URLOpen` (page 1751), your callback function notified if any of the following events occur: `kURLPercentEvent`, `kURLPeriodicEvent`, `kURLPropertyChangedEvent`, `kURLSystemEvent`, `kURLInitiatedEvent`, `kURLResourceFoundEvent`, `kURLDownloadingEvent`, `kURLUploadingEvent`, `kURLAbortInitiatedEvent`, `kURLCompletedEvent`, and `kURLErrorOccurredEvent`.

When your callback is called, it should process the event immediately and return 0. You may wish your callback function to update its user interface, allocate and deallocate memory, or call the Thread Manager function `NewThread`.

Special Considerations

Do not call the function `URLDisposeReference` (page 1741) from your notification callback function. Doing so may cause your application to stop working.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`URLAccess.h`

URLSystemEventProcPtr

Defines a pointer to your system event notification callback that handles update events that occur while a dialog box is displayed during a data transfer operation.

```
typedef OSStatus (*URLSystemEventProcPtr)
(
    void * userContext,
    EventRecord * event
);
```

If you name your function `MyURLSystemEventProc`, you would declare it like this:

```
OSStatus MyURLSystemEventProc (
    void * userContext,
    EventRecord * event
);
```

Parameters

userContext

A pointer to application-defined storage that your application previously passed to the function `URLSimpleDownload` (page 1754), `URLDownload` (page 1741), `URLSimpleUpload` (page 1756), or `URLUpload` (page 1757). Your application can use this value to set up its context when the system event callback function is called.

event

A pointer to an event record containing information about the system event that occurred during the data transfer operation.

Return Value

A result code. See "URL Access Manager Result Codes" (page 1779). Your system event callback function should process the system event and return `noErr`.

Discussion

You pass a pointer to your callback function in the `eventProc` parameter of the function `URLSimpleDownload` (page 1754), `URLSimpleUpload` (page 1756), `URLDownload` (page 1741), or `URLUpload` (page 1757) if you want update events to be passed to your application while a dialog box is displayed by these functions. (In Mac OS X, this is not necessary, since all dialog boxes are moveable). In order for these functions to display a dialog box, you must set the mask constant `kURLDisplayProgressFlag` or `kURLDisplayAuthFlag` in the bitmask passed in the `openFlags` parameter. Call the function `NewURLSystemEventUPP` (page 1739) to create a UPP to your system event notification callback. If you do not write your own system event notification callback, these functions will display a nonmovable modal dialog box.

When your callback is called, it should process the event immediately and return 0. You may wish your callback function to update its user interface, allocate and deallocate memory, or call the Thread Manager function `NewThread`.

Special Considerations

Do not call the function `URLDisposeReference` (page 1741) from your callback function. Doing so may cause your application to stop working.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`URLAccess.h`

Data Types

URLCallbackInfo

Contains information about a data transfer event.

```
struct URLCallbackInfo {
    UInt32 version;
    URLReference urlRef;
    const char * property;
    UInt32 currentSize;
    EventRecord * systemEvent;
};
typedef struct URLCallbackInfo URLCallbackInfo;
```

Fields

`version`
The version of this structure. This value is currently 0.

`urlRef`
A reference to the URL associated with the data transfer event.

`property`
A pointer to a C string representing the name of the URL property that has changed, if relevant. This field is only valid if a property change event occurs as identified by the event constant `kURLPropertyChangedEvent`, described in "Data Transfer Event Constants" (page 1764). or a description of name constants and data types of the corresponding property values, see "Universal URL Property Name Constants" (page 1776) and "HTTP and HTTPS URL Property Name Constants" (page 1775). You should specify this field if the event involves a change in a property value.

`currentSize`
The current total size (in bytes) of the data that has been downloaded and processed by the client.

`systemEvent`
A pointer to an event record containing information about the system event that occurred, if relevant. If the event is not a system event, as identified by the event constant `kURLSystemEvent`, described in "Data Transfer Event Constants" (page 1764), this field is not valid.

Discussion

The `URLCallbackInfo` type represents a structure that contains information about the data transfer event that you want notification of. The URL Access Manager passes a pointer to this structure in the `callbackInfo` parameter of your notification callback function. For information on how to write a notification callback function, see `URLNotifyProcPtr` (page 1759).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`URLAccess.h`

URLNotifyUPP

```
typedef URLNotifyProcPtr URLNotifyUPP;
```

Discussion
For more information, see the description of the callback function `URLNotifyProcPtr` (page 1759).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`URLAccess.h`

URLReference

Represents a reference to a URL.

```
typedef struct OpaqueURLReference * URLReference;
```

Discussion
The `URLReference` type represents a reference to an opaque structure that identifies a URL. You should call the function `URLNewReference` (page 1750) to create a URL reference. The function `URLDisposeReference` (page 1741) disposes of a URL reference when no longer needed. You pass a reference of this type to URL Access Manager functions that operate on a URL in some way.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`URLAccess.h`

URLSystemEventUPP

```
typedef URLSystemEventProcPtr URLSystemEventUPP;
```

Discussion
For more information, see the description of the callback function `URLSystemEventProcPtr` (page 1760).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`URLAccess.h`

Constants

Authentication Type Constant

Represents the default value of the property value identified by the property name constant `kURLAuthType`.

```
enum {
    kUserNameAndPasswordFlag = 0x00000001
};
```

Constants

`kUserNameAndPasswordFlag`

Represents the default value of the property value identified by the property name constant `kURLAuthType`, described in "Universal URL Property Name Constants" (page 1776). This value indicates that both the user name and password are used for authentication.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

Discussion

This constant represents the default value of the authentication type property value. The authentication type property value is identified by the property name constant `kURLAuthType`, described in "Universal URL Property Name Constants" (page 1776). If you do not set the `kURLAuthType` property, the default value will be used for the authentication type. In this case, both the user name and password are used for authentication purposes.

Data Transfer Event Constants

Identify data transfer events that occur during a data transfer operation.

```
typedef UInt32 URLEvent;
enum {
    kURLInitiatedEvent = kURLInitiatingState,
    kURLResourceFoundEvent = kURLResourceFoundState,
    kURLDownloadingEvent = kURLDownloadingState,
    kURLAbortInitiatedEvent = kURLAbortingState,
    kURLCompletedEvent = kURLCompletedState,
    kURLErrorOccurredEvent = kURLErrorOccurredState,
    kURLDataAvailableEvent = kURLDataAvailableState,
    kURLTransactionCompleteEvent = kURLTransactionCompleteState,
    kURLUploadingEvent = kURLUploadingState,
    kURLSystemEvent = 29,
    kURLPercentEvent = 30,
    kURLPeriodicEvent = 31,
    kURLPropertyChangedEvent = 32
};
```

Constants

`kURLInitiatedEvent`

Indicates the function `URLOpen` (page 1751) has been called but the location specified by the URL reference has not yet been accessed.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLResourceFoundEvent`

Indicates that the location specified by the URL reference has been accessed and is valid.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLDownloadingEvent`
Indicates that a download operation is in progress.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLAbortInitiatedEvent`
Indicates that a data transfer operation has been aborted. When your application calls the function `URLAbort` (page 1740), the URL Access Manager changes the state returned by the function `URLGetCurrentState` (page 1744) to `kURLAbortingState` and passes the constant `kURLAbortInitiatedEvent` to your notification callback function. When data transfer is terminated, the URL Access Manager changes the state returned by `URLGetCurrentState` (page 1744) to `kURLCompletedState` and passes the constant `kURLCompletedEvent` in the `event` parameter of your notification callback function.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLCompletedEvent`
Indicates that all operations associated with a call to `URLOpen` (page 1751) have been completed. This includes the successful completion of a download or upload operation or the completion of cleanup work after aborting a download or upload operation. For example, when a data transfer operation is aborted, the URL Access Manager changes the state returned by the function `URLGetCurrentState` (page 1744) to `kURLCompletedState` and passes the constant `kURLCompletedEvent` in the `event` parameter of your notification callback function.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLErrorOccurredEvent`
Indicates that an error occurred during data transfer. If you receive this event, you may wish to call the function `URLGetError` (page 1746) to determine the nature of the error.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLDataAvailableEvent`
Indicates that data is available in buffers. If you receive this event, you can call the function `URLGetBuffer` (page 1743) to obtain the next buffer of data. You may wish to call the function `URLGetDataAvailable` (page 1745) to determine the amount of data available for retrieval in a download operation. Note that if you pass a valid file specification in the `fileSpec` parameter of `URLOpen` (page 1751), your notification callback function will not be called for data available events.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLTransactionCompleteEvent`
Indicates that a download operation is complete because there is no more data to retrieve from buffers. Note that if you pass a valid file specification in the `fileSpec` parameter of `URLOpen` (page 1751), your notification callback function will not be called for transaction completed events.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLUploadingEvent`
Indicates that an upload operation is in progress.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLSystemEvent`
: Indicates that a system event has occurred.

  Available in Mac OS X v10.0 and later.

  Declared in `URLAccess.h`.

`kURLPercentEvent`
: Indicates that the size of the data being downloaded is known. In this case, an increment of one percent of the data was transferred into buffers.

  Available in Mac OS X v10.0 and later.

  Declared in `URLAccess.h`.

`kURLPeriodicEvent`
: Indicates that a time interval of approximately one quarter of a second has passed.

  Available in Mac OS X v10.0 and later.

  Declared in `URLAccess.h`.

`kURLPropertyChangedEvent`
: Indicates that a property such as a filename has become known or changed. In this case, the name of the changed property will be passed to your notification function via the `property` field of the `callbackInfo` structure.

  Available in Mac OS X v10.0 and later.

  Declared in `URLAccess.h`.

Discussion
The `URLEvent` enumeration defines constants that identify data transfer events that occur during a data transfer operation performed by `URLOpen` (page 1751). In order to be notified of these events, you must pass a UPP to your data transfer notification callback in the `notifyProc` parameter. You indicate the type of data transfer events you want to receive via a bitmask in the `eventRegister` parameter. For a description of this bitmask, see "Data Transfer Event Mask Constants" (page 1766).

Data Transfer Event Mask Constants

Represent a mask that identifies the data transfer events occurring during a data transfer operation that your application wants notification of.

```
typedef unsigned long URLEventMask;
enum {
    kURLInitiatedEventMask = 1 << (kURLInitiatedEvent - 1),
    kURLResourceFoundEventMask = 1 << (kURLResourceFoundEvent
- 1),
    kURLDownloadingMask = 1 << (kURLDownloadingEvent - 1),
    kURLUploadingMask = 1 << (kURLUploadingEvent - 1),
    kURLAbortInitiatedMask = 1 << (kURLAbortInitiatedEvent
- 1),
    kURLCompletedEventMask = 1 << (kURLCompletedEvent - 1),
    kURLErrorOccurredEventMask = 1 << (kURLErrorOccurredEvent
- 1),
    kURLDataAvailableEventMask = 1 << (kURLDataAvailableEvent
- 1),
    kURLTransactionCompleteEventMask = 1 << (kURLTransactionCompleteEvent
- 1),
    kURLSystemEventMask = 1 << (kURLSystemEvent - 1),
    kURLPercentEventMask = 1 << (kURLPercentEvent - 1),
    kURLPeriodicEventMask = 1 << (kURLPeriodicEvent - 1),
    kURLPropertyChangedEventMask = 1 << (kURLPropertyChangedEvent
- 1),
    kURLAllBufferEventsMask = kURLDataAvailableEventMask +
kURLTransactionCompleteEventMask,
    kURLAllNonBufferEventsMask = kURLInitiatedEventMask + kURLDownloadingMask
+ kURLUploadingMask + kURLAbortInitiatedMask + kURLCompletedEventMask
+ kURLErrorOccurredEventMask + kURLPercentEventMask + kURLPeriodicEventMask
+ kURLPropertyChangedEventMask,
    kURLAllEventsMask = 0xFFFFFFFF
};
```

Constants

`kURLInitiatedEventMask`

If the bit specified by this mask is set, your notification callback function will be notified when the function `URLOpen` (page 1751) has been called but the location specified by the URL reference has not yet been accessed.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLResourceFoundEventMask`

If the bit specified by this mask is set, your notification callback function will be notified when the location specified by a URL reference has been accessed and is valid.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLDownloadingMask`

If the bit specified by this mask is set, your notification callback function will be notified when a download operation is in progress.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLUploadingMask`

If the bit specified by this mask is set, your notification callback function will be notified when an upload operation is in progress.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLAbortInitiatedMask`

If the bit specified by this mask is set, your notification callback function will be notified when a download or upload operation has been aborted. When your application calls the function `URLAbort` (page 1740), the URL Access Manager changes the state returned by the function `URLGetCurrentState` (page 1744) to `kURLAbortingState` and passes the constant `kURLAbortInitiatedEvent` to your notification callback function. When data transfer is terminated, the URL Access Manager changes the state returned by `URLGetCurrentState` (page 1744) to `kURLCompletedState` and passes the constant `kURLCompletedEvent` in the `event` parameter of your notification callback function.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLCompletedEventMask`

If the bit specified by this mask is set, your notification callback function will be notified when all operations associated with a call to the function `URLOpen` (page 1751) have been completed. This indicates either the successful completion of an operation or the completion of cleanup work after aborting the operation. For example, when a data transfer operation is aborted, the URL Access Manager changes the state returned by the function `URLGetCurrentState` (page 1744) to `kURLCompletedState` and passes the constant `kURLCompletedEvent` in the `event` parameter of your notification callback function.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLErrorOccurredEventMask`

If the bit specified by this mask is set, your notification callback function will be notified when an error has occurred. If you receive this event, you may wish to call the function `URLGetError` (page 1746) to determine the nature of the error.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLDataAvailableEventMask`

If the bit specified by this mask is set, your notification callback function will be notified when data is available in buffers. If you receive this event, you may wish to call the function `URLGetDataAvailable` (page 1745) to determine the amount of data available for retrieval in a download operation. Note that if you pass a valid file specification in the `fileSpec` parameter of the function `URLOpen` (page 1751), your notification callback function will not be called for data available events.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLTransactionCompleteEventMask`

If the bit specified by this mask is set, your notification callback function will be notified when the operation is complete because there is no more data to retrieve from buffers. Note that if you pass a valid file specification in the `fileSpec` parameter of the function `URLOpen` (page 1751), your notification callback function will not be called for transaction completed events.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLSystemEventMask`

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLPercentEventMask`
: If the bit specified by this mask is set, your notification callback function will be notified when an increment of one percent of the data has been transferred into buffers. This occurs only when the size of the data being transferred is known. This information is useful if you want the URL Access Manager to display a progress indicator.

  Available in Mac OS X v10.0 and later.

  Declared in `URLAccess.h`.

`kURLPeriodicEventMask`
: If the bit specified by this mask is set, your notification callback function will be notified when a time interval of approximately one quarter of a second has passed. You can use this event to report the progress of the download operation when the size of the data is unknown or for other processing that you wish to perform at regular intervals.

  Available in Mac OS X v10.0 and later.

  Declared in `URLAccess.h`.

`kURLPropertyChangedEventMask`
: If the bit specified by this mask is set, your notification callback function will be notified when the value of a URL property, such as a filename or user name, has become known or changes.

  Available in Mac OS X v10.0 and later.

  Declared in `URLAccess.h`.

`kURLAllBufferEventsMask`
: If the bit specified by this mask is set, your notification callback function will be notified when a buffer-related event indicated by the event constants `kURLDataAvailableEvent` or `kURLTransactionCompleteEvent` occurred. If you pass a file specification in the `fileSpec` parameter of the function `URLOpen` (page 1751), your notification callback function will not be called for buffer-related events.

  Available in Mac OS X v10.0 and later.

  Declared in `URLAccess.h`.

`kURLAllNonBufferEventsMask`
: If the bit specified by this mask is set, your notification callback function will be notified when an event unrelated to a buffer occurred. This includes all events except those represented by the constants `kURLDataAvailableEvent` and `kURLTransactionCompleteEvent`.

  Available in Mac OS X v10.0 and later.

  Declared in `URLAccess.h`.

`kURLAllEventsMask`
: If the bit specified by this mask is set, your notification callback function will be notified when any of the above data transfer events occur. If you pass a file specification in the `fileSpec` parameter of the function `URLOpen` (page 1751), your notification callback function will not be called for buffer-related events.

  Available in Mac OS X v10.0 and later.

  Declared in `URLAccess.h`.

Discussion

The `URLEventMask` enumeration defines masks that identify the data transfer events occurring during a call to the function `URLOpen` (page 1751) that your application wants notification of. For a description of data transfer events, see "Data Transfer Event Constants" (page 1764). You can define an event notification function and the events for which you want to receive notification only if you do not specify a file in which to store the data for downloads. You can indicate which events you want to receive notification of via a bitmask in the `eventRegister` parameter of `URLOpen` (page 1751).

Data Transfer Options Mask Constants

Represent a mask that identifies the data transfer options to use when uploading or downloading data.

```
typedef UInt32 URLOpenFlags;
enum {
    kURLReplaceExistingFlag = 1 << 0,
    kURLBinHexFileFlag = 1 << 1,
    kURLExpandFileFlag = 1 << 2,
    kURLDisplayProgressFlag = 1 << 3,
    kURLDisplayAuthFlag = 1 << 4,
    kURLUploadFlag = 1 << 5,
    kURLIsDirectoryHintFlag = 1 << 6,
    kURLDoNotTryAnonymousFlag = 1 << 7,
    kURLDirectoryListingFlag = 1 << 8,
    kURLExpandAndVerifyFlag = 1 << 9,
    kURLNoAutoRedirectFlag = 1 << 10,
    kURLDebinhexOnlyFlag = 1 << 11,
    kURLDoNotDeleteOnErrorFlag = 1 << 12,
    kURLResumeDownloadFlag = 1 << 13,
    kURLReservedFlag = (unsigned long) 1 << 31
};
```

Constants

`kURLReplaceExistingFlag`

If the bit specified by this mask is set and the destination file or directory exists, the file or directory contents are replaced by the newly downloaded or uploaded data. If this bit is not set and the name of the file is specified and does exist, the URL Access Manager returns the result code `kURLDestinationExistsError`. If the name of the file or directory is not specified, the file or directory already exists, and the bit specified by this mask is not set, a number is appended to the name before any extension until a unique name is created, and the data is transferred to the new file or directory name without notifying the calling application that the name has changed. In the case of a download operation, your application can check the `destination` parameter of the functions `URLSimpleDownload` (page 1754) and `URLDownload` (page 1741) to obtain the new filename.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLBinHexFileFlag`

If the bit specified by this mask is set, the URL Access Manager converts a nontext file that has a resource fork to `BinHex` format before uploading it.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLExpandFileFlag`

If the bit specified by this mask is set, files in `BinHex` format are decoded. If version 5.0 of the Stuffit Engine is installed in the System Folder, the URL Access Manager uses it to expand the file.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLDisplayProgressFlag`

If the bit specified by this mask is set, the URL Access Manager displays a nonmovable modal progress indicator during data transfer operations only if you have not provided a system event notification callback. On Mac OS X, dialog boxes will always be moveable. To handle data transfer events that occur while a progress indicator is being displayed, pass a UPP to your data transfer event notification callback in the `eventProc` parameter of the functions `URLSimpleDownload` (page 1754), `URLDownload` (page 1741), `URLSimpleUpload` (page 1756), and `URLUpload` (page 1757).

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLDisplayAuthFlag`

If the bit specified by this mask is set, the URL Access Manager displays a nonmovable modal authentication dialog box when user authentication is required only if you have not provided a system event notification callback. On Mac OS X, dialog boxes will always be moveable. To handle data transfer events that occur while an authentication dialog box is being displayed, pass a UPP to your data transfer event notification callback in the `eventProc` parameter of the functions `URLSimpleDownload` (page 1754), `URLDownload` (page 1741), `URLSimpleUpload` (page 1756), and `URLUpload` (page 1757). If the bit specified by this mask is clear, the user name and password properties of the URL are used for authentication purposes. If these are not set, the URL Access Manager returns the result code `kURLAuthenticationError`.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLUploadFlag`

If the bit specified by this mask is set, the function `URLOpen` (page 1751) will upload the file or directory to the specified URL.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLIsDirectoryHintFlag`

If the bit specified by this mask is set, download operations assume that the URL points to a directory. Note that if you pass a pathname that specifies a file in the `url` parameter of the function `URLSimpleDownload` (page 1754), the file is downloaded regardless of whether you specify `kURLDirectoryListingFlag` or `KURLIsDirectoryHintFlag` in the `openFlags` parameter.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLDoNotTryAnonymousFlag`

If the bits specified by this mask is set, when FTP authentication occurs, the functions `URLSimpleDownload` (page 1754), `URLDownload` (page 1741), `URLSimpleUpload` (page 1756), `URLUpload` (page 1757), and `URLOpen` (page 1751) will not try to log on anonymously. Instead, they will rely on the setting of the mask constant `kURLDisplayAuthFlag`. If the bit specified by the `kURLDoNotTryAnonymousFlag` mask is not set, these functions will first attempt to log on anonymously.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLDirectoryListingFlag`
If the bit specified by this mask is set, a listing of the directory, rather than the entire directory, is downloaded. If the URL points to a file instead of a directory, the file is downloaded. Note that if you pass a pathname that specifies a file in the `url` parameter of the function `URLSimpleDownload` (page 1754), the file is downloaded regardless of whether you specify `kURLDirectoryListingFlag` or `KURLIsDirectoryHintFlag` in the `openFlags` parameter.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLExpandAndVerifyFlag`
If this flag is available (that is, the File Signing shared library is available) and the bit specified by this mask is set, the signature attached to the file is verified. Success indicates that the file was signed by the certificate authority, but the certificate will not be displayed until after the file is downloaded.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLNoAutoRedirectFlag`
If the bit specified by this mask is set, if an HTTP request returns a "redirect" status (300, 301, or 302), the transfer will complete without attempting to redirect to the next URL. Otherwise, redirects are followed until actual data is encountered or an error is returned.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLDebinhexOnlyFlag`
If the bit specified by this mask is set, the internal engine is used to decode files, rather than the external Stuffit Engine, even if Stuffit is installed. This prevents the display of the Stuffit progress user interface. If you set this bit, you must also set the `kURLExpandFileFlag` mask.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLDoNotDeleteOnErrorFlag`
Do not delete the downloaded file if an error or abort occurs. This flag applies to downloading only and should be used if interested in later resuming the download.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLResumeDownloadFlag`
The passed in file is partially downloaded, attempt to resume it. Currently works for HTTP only. If no `FSSpec` passed in, this flag will be ignored. Overridden by `kURLReplaceExistingFlag`.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLReservedFlag`
Reserved for internal use.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

Discussion
The `URLOpenFlags` enumeration defines masks you can use to identify the data transfer options you want used when performing data transfer operations. You pass this mask in the `openFlags` parameter of the functions `URLSimpleDownload` (page 1754), `URLDownload` (page 1741), `URLSimpleUpload` (page 1756),

URLUpload (page 1757), and URLOpen (page 1751). The options that you can specify for upload and download operations differ, as do those that you can specify for the low-level function URLOpen (page 1751). For a description of the options you can specify in each case, see the appropriate function discussions.

Data Transfer State Constants

Identifies the current state of a data transfer operation.

```
typedef UInt32 URLState;
enum {
    kURLNullState = 0,
    kURLInitiatingState = 1,
    kURLLookingUpHostState = 2,
    kURLConnectingState = 3,
    kURLResourceFoundState = 4,
    kURLDownloadingState = 5,
    kURLDataAvailableState = 0x10 + kURLDownloadingState,
    kURLTransactionCompleteState = 6,
    kURLErrorOccurredState = 7,
    kURLAbortingState = 8,
    kURLCompletedState = 9,
    kURLUploadingState = 10
};
```

Constants

kURLNullState
: Indicates that the function URLOpen (page 1751) has not yet been called.

  Available in Mac OS X v10.0 and later.

  Declared in URLAccess.h.

kURLInitiatingState
: Indicates that the function URLOpen (page 1751) has been called, but the location specified by the URL reference has not yet been accessed. The stream enters this state from the kURLNullState state.

  Available in Mac OS X v10.0 and later.

  Declared in URLAccess.h.

kURLLookingUpHostState
: Indicates that the function URLOpen (page 1751) has been called, and that the host is being looked up. The stream enters this state from the kURLInitiatingState state.

  Available in Mac OS X v10.0 and later.

  Declared in URLAccess.h.

kURLConnectingState
: Indicates that the function URLOpen (page 1751) has been called, and a connection is being made. The stream enters this state from the kURLLookingUpHostState state.

  Available in Mac OS X v10.0 and later.

  Declared in URLAccess.h.

kURLResourceFoundState
: Indicates that the location specified by the URL reference has been accessed and is valid. The stream enters this state from the kURLConnectingState state.

  Available in Mac OS X v10.0 and later.

  Declared in URLAccess.h.

`kURLDownloadingState`
Indicates that the download operation is in progress but there is currently no data in the buffers. The stream enters this state initially from the `kURLResourceFoundState` state. During a download operation, the stream's state may alternate between the `kURLDownloadingState` and the `kURLDataAvailableState` states.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLDataAvailableState`
Indicates that the download operation is in progress and data is available in the buffers. The stream initially enters this state from the `kURLDownloadingState` state. During a download operation, the stream's state may alternate between the `kURLDownloadingState` and the `kURLDataAvailableState` states. If the stream is in the data available state, you may want to call the function `URLGetDataAvailable` (page 1745) to determine the amount of data available for download. If you pass `NULL` in the `fileSpec` parameter of the function `URLOpen` (page 1751), you will need to call the function `URLGetBuffer` (page 1743) to obtain the next buffer of data.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLTransactionCompleteState`
Indicates that a download or upload operation is complete. The stream can enter this state from the `kURLDownloadingState` state.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLErrorOccurredState`
Indicates that an error occurred during data transfer. The stream can enter this state from any state except the `kURLAbortingState` state. If the stream is in this state, you may wish to call the function `URLGetError` (page 1746) to determine the nature of the error.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLAbortingState`
Indicates that a download or upload operation is aborting. The stream enters this state from the `kURLErrorOccurredState` state or as a result of calling the function `URLOpen` (page 1751) when the stream is in any other state. When your application calls the function `URLAbort` (page 1740), the URL Access Manager changes the state returned by the function `URLGetCurrentState` (page 1744) to `kURLAbortingState` and passes the constant `kURLAbortInitiatedEvent` to your notification callback function.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLCompletedState`
Indicates that there is no more activity to be performed on this stream. In this case, the data transfer has either completed successfully or been aborted. The stream enters this state from the `kURLTransactionCompleteState` or the `kURLAbortingState` state. When data transfer is terminated after a data transfer operation is aborted, the URL Access Manager changes the state returned by `URLGetCurrentState` (page 1744) to `kURLCompletedState` and passes the constant `kURLCompletedEvent` in the `event` parameter of your notification callback function.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLUploadingState`
Indicates that an upload operation is in progress.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

Discussion
The `URLState` enumeration defines constants that identify the status of a data transfer operation with respect to a URL. The function `URLGetCurrentState` (page 1744) passes back one of these constants in the `state` parameter to indicate the status of a data transfer operation. All constants except `kURLDataAvailableState` and `kURLCompletedState` can be returned at any time. If you pass a valid file specification in the `fileSpec` parameter of the function `URLOpen` (page 1751), your notification callback function will not be notified of data available and transaction completed states as identified by the constants `kURLDataAvailableState` and `kURLTransactionCompleteState`.

HTTP and HTTPS URL Property Name Constants

Identify property values specific to HTTP and HTTPS URLs.

```
#define kURLHTTPRequestMethod "URLHTTPRequestMethod"
#define kURLHTTPRequestHeader "URLHTTPRequestHeader"
#define kURLHTTPRequestBody "URLHTTPRequestBody"
#define kURLHTTPRespHeader "URLHTTPRespHeader"
#define kURLHTTPUserAgent "URLHTTPUserAgent"
#define kURLHTTPRedirectedURL "URLHTTPRedirectedURL"
#define kURLSSLCipherSuite "URLSSLCipherSuite"
```

Constants
`kURLHTTPRequestMethod`
Identifies the HTTP request method property value. You use this name constant to set or obtain a C string that represents the HTTP method to be used in the request. If you are posting a form, you must set this property to the string "POST".

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLHTTPRequestHeader`
Identifies the HTTP request header property value. You use this name constant to set or obtain a C string that represents the HTTP header to be used in the request. You may set this property to contain all headers needed for the request. If you are posting a form and have set the properties identified by the name constants `kURLHTTPRequestMethod` and `kURLHTTPRequestBody`, you do not need to set the property identified by this tag.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLHTTPRequestBody`
Identifies the HTTP request body property value. You use this name constant to set or obtain a buffer of data that represents the HTTP body to be provided in the request. If you set the property identified by this tag but not that identified by the name constant `kURLHTTPHeader`, a body-length header is automatically added to the request. If you are posting a form, you must set this property to the form data you want sent.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLHTTPRespHeader`
: Identifies the HTTP response header property value. You use this name constant to obtain a C string that represents the HTTP response header that was received.

  Available in Mac OS X v10.0 and later.

  Declared in `URLAccess.h`.

`kURLHTTPUserAgent`
: Identifies the user agent property value. You use this name constant to set or obtain a C string that represents the HTTP user agent string that is embedded in HTTP requests. By default, the URL Access Manager sets the user agent string to "URL Access 1.0 (Macintosh; PPC)".

  Available in Mac OS X v10.0 and later.

  Declared in `URLAccess.h`.

`kURLHTTPRedirectedURL`
: Identifies the redirected URL property value. You use this name constant to obtain a C string that represents the URL that you were redirected to.

  Available in Mac OS X v10.0 and later.

  Declared in `URLAccess.h`.

`kURLSSLCipherSuite`
: Identifies the SSL cipher suite property value.

  Available in Mac OS X v10.0 and later.

  Declared in `URLAccess.h`.

Discussion
These constants represent Apple-defined name constants that identify property values specific to HTTP and HTTPS URLs. For a description of the name constants that identify property values universal to all URLs, see `Universal URL Property Name Constants` (page 1776).

You pass one of these name constants in the property parameter of the functions `URLSetProperty` (page 1753) and `URLGetProperty` (page 1747), respectively, to set or obtain a particular property value. Note that you can only set HTTP and HTTPS property values identified by the constants `kURLHTTPRequestMethod`, `kURLHTTPRequestHeader`, `kURLHTTPRequestBody`, and `kURLHTTPUserAgent`. You must also pass the correct data type corresponding to the property value in the `propertyBuffer` parameter of these functions.

Version Notes
Prior to version 2.0.3 of the URL Access Manager, the data type of the property value identified by the name constant `kURLHTTPRequestBody` was a C string. In 2.0.3 and later, the data type is a buffer of data.

Universal URL Property Name Constants

Identify property values universal to all URLs.

```
#define kURLURL "URLString"
#define kURLResourceSize "URLResourceSize"
#define kURLLastModifiedTime "URLLastModifiedTime"
#define kURLMIMEType "URLMIMEType"
#define kURLFileType "URLFileType"
#define kURLFileCreator "URLFileCreator"
#define kURLCharacterSet "URLCharacterSet"
#define kURLResourceName "URLResourceName"
#define kURLHost "URLHost"
#define kURLAuthType "URLAuthType"
#define kURLUserName "URLUserName"
#define kURLPassword "URLPassword"
#define kURLStatusString "URLStatusString"
#define kURLIsSecure "URLIsSecure"
#define kURLCertificate "URLCertificate"
#define kURLTotalItems "URLTotalItems"
#define kURLConnectTimeout "URLConnectTimeout"
```

Constants

`kURLURL`

Identifies the name string property value. You use this name constant to obtain a C string that represents the URL.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLResourceSize`

Identifies the resource size property value. You use this name constant to obtain a value of type `Size` that represents the total size of the data at the location specified by the URL.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLLastModifiedTime`

Identifies the modification time property value. You use this name constant to obtain a value of type `UInt32` that represents the last time the data was modified.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLMIMEType`

Identifies the MIME type property value. You use this name constant to obtain a Pascal string that represents the MIME type of the URL.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLFileType`

Identifies the file type property value. You use this name constant to set or obtain a value of type `OSType` that represents the file type as specified in a call to the function `URLOpen` (page 1751). If the file type was not specified, `kURLFileType` obtains the file type compatible with the MIME type.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLFileCreator`
Identifies the file creator property value. You use this name constant to set or obtain a value of type `OSType` that represents the file creator as specified in a call to the function `URLOpen` (page 1751). If the file creator was not specified, `kURLFileType` obtains the file type compatible with the MIME type.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLCharacterSet`
Identifies the character set property value. You use this name constant to obtain a Pascal string that represents the character set used by the URL, as returned by the HTTP server.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLResourceName`
Identifies the resource name property value. You use this name constant to obtain a Pascal string that represents the name associated with the data to be downloaded.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLHost`
Identifies the host property value. You use this name constant to obtain a Pascal string that represents the host on which the data is located.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLAuthType`
Identifies the authentication type property value. You use this name constant to obtain a value that represents the type of authentication that the download operation requires. The default authentication type is `kUserNameAndPasswordFlag`, described in `Authentication Type Constant` (page 1763).

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLUserName`
Identifies the user name property value. You use this name constant to set or obtain a Pascal string that represents the user name used for authentication.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLPassword`
Identifies the password property value. You use this name constant to set or obtain a Pascal string that represents the password used for authentication.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLStatusString`
Identifies the status property value. You use this name constant to obtain a Pascal string that represents the current status of the data stream. You can use this property to display the status of the data transfer operation.

Available in Mac OS X v10.0 and later.

Declared in `URLAccess.h`.

`kURLIsSecure`
: Identifies the security property value. You use this name constant to get a Boolean value that indicates whether the download operation is secure.

  Available in Mac OS X v10.0 and later.

  Declared in `URLAccess.h`.

`kURLCertificate`
: Identifies the certificate property value. You use this name constant to obtain a buffer of data that represents the certificate provided by a remote server.

  Available in Mac OS X v10.0 and later.

  Declared in `URLAccess.h`.

`kURLTotalItems`
: Identifies the total items property value. You use this name constant to obtain a value of type `UInt32` that represents the total number of items being uploaded or downloaded.

  Available in Mac OS X v10.0 and later.

  Declared in `URLAccess.h`.

`kURLConnectTimeout`
: Identifies the connection timeout property value.

  Available in Mac OS X v10.0 and later.

  Declared in `URLAccess.h`.

Discussion
These constants represent Apple-defined name constants that identify property values universal to all URLs. For a description of the name constants that identify property values specific to HTTP and HTTPS URLs, see `HTTP and HTTPS URL Property Name Constants` (page 1775).

You pass one of these name constants in the property parameter of the functions `URLSetProperty` (page 1753) and `URLGetProperty` (page 1747), respectively, to set or obtain a particular property value. Note that you can only set the universal property values identified by the constants `kURLPassword` and `kURLUserName`. You must also pass the correct data type corresponding to the property value in the `propertyBuffer` parameter of these functions.

Result Codes

The most common result codes returned by URL Access Manager are listed in the table below. The following result codes may also be returned; `noErr` (0), `nsvErr` (-35), `fnfErr` (-43), `paramErr` (-50), and `dirNFErr` (-120).

Result Code	Value	Description
`kURLInvalidURLReferenceError`	-30770	Returned by functions that operate on URL references to indicate that a reference is invalid. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
kURLProgressAlreadyDisplayedError	-30771	Returned by the functions URLSimpleDownload, URLDownload, URLSimpleUpload, and URLUpload to indicate that a progress indicator is already displayed. Available in Mac OS X v10.0 and later.
kURLDestinationExistsError	-30772	Returned by the functions URLSimpleUpload and URLUpload to indicate that the destination file already exists. Available in Mac OS X v10.0 and later.
kURLInvalidURLError	-30773	Returned by functions that operate on URL strings to indicate that the format of the URL is invalid. Available in Mac OS X v10.0 and later.
kURLUnsupportedSchemeError	-30774	Returned by functions that operate on URL strings to indicate that the transfer protocol is not supported. Available in Mac OS X v10.0 and later.
kURLServerBusyError	-30775	Indicates a failed data transfer operation. Available in Mac OS X v10.0 and later.
kURLAuthenticationError	-30776	Returned by URLSimpleDownload, URLDownload, URLSimpleUpload, and URLUpload functions if no authentication dialog box is allowed and the user name and password properties of the URL are set incorrectly or don't exist. Available in Mac OS X v10.0 and later.
kURLPropertyNotYetKnownError	-30777	Returned by the functions URLGetProperty and URLGetPropertySize to indicate that the value or size of a URL property is not available. Available in Mac OS X v10.0 and later.
kURLUnknownPropertyError	-30778	Returned by functions URLSetProperty, URLGetProperty, and URLGetPropertySize to indicate that the property is invalid or undefined. Available in Mac OS X v10.0 and later.
kURLPropertyBufferTooSmallError	-30779	Returned by the function URLGetProperty to indicate that the buffer is too small to receive the requested property. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
kURLUnsettablePropertyError	-30780	Returned by the function URLSetProperty to indicate that the property cannot be set. Available in Mac OS X v10.0 and later.
kURLInvalidCallError	-30781	Returned by the functions URLGetDataAvailable, URLGetBuffer, and URLReleaseBuffer to indicate that the call is invalid. Available in Mac OS X v10.0 and later.
kURLFileEmptyError	-30783	Indicates a failed data transfer operation. Available in Mac OS X v10.0 and later.
kURLExtensionFailureError	-30785	Indicates that your extension failed to load. Available in Mac OS 9 and earlier.
kURLInvalidConfigurationError	-30786	Indicates a failed data transfer operation. This is returned when you attempt to upload through an HTTP proxy, since upload through proxies is not supported. Available in Mac OS X v10.0 and later.
kURLAccessNotAvailableError	-30787	Returned by the function URLGetURLAccessVersion to indicate that the URL Access Manager is not available. Available in Mac OS X v10.0 and later.
kURL68kNotSupportedError	-30788	Indicates that URL Access Manager was called from within a 68K context. Available in Mac OS 9 and earlier.

Window Manager Reference

Framework: Carbon/Carbon.h

Declared in MacWindows.h

Overview

Your application uses the Window Manager to create and manage windows. For example, your application uses the Window Manager to create and display a new window when the user creates a new document or opens an existing document. When the user clicks or holds down the mouse button while the cursor is in a window created by your application, you use the Window Manager to determine the location of the mouse action and to alter the window display as appropriate. When the user closes a window, you use the Window Manager to remove the window from the screen.

A Macintosh application uses windows for most communication with the user, from discrete interactions such as presenting and acknowledging alert boxes to open-ended interactions such as creating and editing documents. Users generally type words and formulas, draw pictures, or otherwise enter data in a window on the screen. Your application typically lets the user save this data in a file, open saved files, and view the saved data in a window.

A window can be any size or shape, and the user can display any number of windows, within the limits of available memory, on the screen at once.

The Window Manager defines a set of standard windows and provides a set of routines for managing them. The Window Manager helps your application display windows that are consistent with the Macintosh user interface.

Note: Historically, the Window Manager has offered different successive methods for creating and manipulating windows. Many of the older functions have been deprecated and, in most cases, this reference provides a recommended replacement. For the most up-to-date information about creating windows, see the document *Handling Carbon Windows and Controls*.

Carbon supports the Window Manager. Be aware, however, that if you use custom window definition procedures (also known as WDEFs), you must move them out of resources and compile them into your application. In addition:

- Your application must use the functions defined by the Window Manager whenever it creates and disposes of Window Manager data structures. For example, instead of directly creating and disposing of window records, applications must call Window Manager functions such as `CreateNewWindow` and `DisposeWindow`.
- You must revise your application so that it accesses Window Manager data structures only through accessor functions.

- You are encouraged to adopt the standard Mac OS window definition procedures in your application. Applications that use the standard Mac OS window definition procedures inherit the Mac OS human interface appearance on Mac OS 9 and Mac OS X. Applications that use custom window definition procedures work correctly, but because custom definition procedures invoke their own drawing routines, the Mac OS can't draw these applications with the current appearance (unless you specifically use Appearance Manager drawing primitives).

Functions by Task

Accessing Information About a Window

`IsValidWindowPtr` (page 1902)
Reports whether a pointer is a valid window pointer.

`GetWindowGreatestAreaDevice` (page 1848)
Returns the graphics device with the greatest area of intersection with a specified window region.

`HIWindowGetGreatestAreaDisplay` (page 1887)
Finds the display with the greatest area of intersection with a window region.

`HIWindowCopyShape` (page 1879)
Retrieves a shape that describes a region of a window.

`HIWindowGetScaleMode` (page 1889)
Obtains the window's scale mode and the application's display scale factor.

`GetWindowList` (page 1856)
Obtains the first window in a window list.

`GetWindowWidgetHilite` (page 1868)
Obtains the window part code of the window widget that is currently highlighted.

`IsWindowModified` (page 1907)
Obtains the modification state of the specified window.

`SetWindowModified` (page 1946)
Sets the modification state of the specified window.

`HIWindowGetCGWindowID` (page 1886)
Returns the Quartz window ID assigned to a window.

`HIWindowFromCGWindowID` (page 1884)
Returns the window in the current process with a specified Quartz window ID.

`GetWindowRegion` (page 1864) Deprecated in Mac OS X v10.5
Obtains a handle to a specific window region.

Activating Window Path Pop-Up Menus

`IsWindowPathSelectEvent` (page 1908)
Determines whether a Carbon event describing a click on a window's title should cause a path selection menu to be displayed.

`WindowPathSelect` (page 1970)
Displays a window path pop-up menu.

`IsWindowPathSelectClick` (page 1907) Deprecated in Mac OS X v10.5
Reports whether a mouse click should activate the window path pop-up menu. (Deprecated. Use `IsWindowPathSelectEvent` (page 1908) instead.)

Associating Data With Windows

`GetWindowProperty` (page 1859)
Obtains a piece of data that is associated with a window.

`SetWindowProperty` (page 1948)
Associates an arbitrary piece of data with a window.

`GetWindowPropertySize` (page 1861)
Obtains the size of a piece of data that is associated with a window.

`RemoveWindowProperty` (page 1923)
Removes a piece of data that is associated with a window.

`ChangeWindowPropertyAttributes` (page 1806)
Changes attributes associated with a window property.

`GetWindowPropertyAttributes` (page 1860)
Obtains the attributes of a window property.

Collapsing Windows

`CollapseWindow` (page 1810)
Collapses or expands a window to the dock.

`CollapseAllWindows` (page 1810)
Collapses or expands all collapsable windows in an application.

`IsWindowCollapsed` (page 1904)
Determines whether a window is currently collapsed.

`IsWindowCollapsable` (page 1903)
Determines whether a window can be collapsed.

Creating, Storing, and Closing Windows

`CreateNewWindow` (page 1815)
Creates a window from parameter data.

`CreateCustomWindow` (page 1814)
Creates a custom window based on a registered toolbox object class or a custom window root view.

`HIWindowCreate` (page 1880)
Creates a standard or custom window.

`DisposeWindow` (page 1820)
Removes a window.

`CreateWindowFromCollection` (page 1817) Deprecated in Mac OS X v10.5
Creates a window from collection data. (Deprecated. Use `HIArchiveCopyDecodedCFType` (page 2310) to decode a window from an archive instead.)

`CreateWindowFromResource` (page 1818) Deprecated in Mac OS X v10.5
Creates a window from `'wind'` resource data. (Deprecated. Use nib files and `CreateWindowFromNib` (page 1059) instead.)

`StoreWindowIntoCollection` (page 1960) Deprecated in Mac OS X v10.5
Stores data describing a window into a collection. (Deprecated. Use `HIArchiveEncodeCFType` (page 2314) to encode a window to an archive instead.)

Displaying Floating Windows and Window Animations

`AreFloatingWindowsVisible` (page 1801)
Indicates whether an application's floating windows are currently visible.

`HideFloatingWindows` (page 1871)
Hides an application's floating windows.

`ShowFloatingWindows` (page 1956)
Shows an application's floating windows.

Displaying Windows

`ActivateWindow` (page 1800)
Activates or deactivates a window.

`IsWindowActive` (page 1902)
Indicates whether the specified window is active.

`HiliteWindow` (page 1873)
Sets a window's highlighting status.

`SelectWindow` (page 1929)
Makes a window active.

`ShowWindow` (page 1959)
Makes an invisible window visible.

`HideWindow` (page 1872)
Makes a window invisible.

`ShowHide` (page 1957)
Sets a window's visibility.

`BringToFront` (page 1803)
Brings a window to the front.

`SendBehind` (page 1929)
Moves one window behind another.

`HIWindowInvalidateShadow` (page 1890)
Recalculates a window's shadow.

Dragging Proxy Icons

`BeginWindowProxyDrag` (page 1802)
Creates the drag reference and the drag image when the user drags a proxy icon.

`EndWindowProxyDrag` (page 1826)
Disposes of the drag reference when the user completes the drag of a proxy icon.

`HiliteWindowFrameForDrag` (page 1874)
Sets the highlight state of the window's structure region to reflect the window's validity as a drag-and-drop destination.

`TrackWindowProxyDrag` (page 1963)
Handles all aspects of the drag process when the user drags a proxy icon.

`TrackWindowProxyFromExistingDrag` (page 1964)
Allows custom handling of the drag process when the user drags a proxy icon.

`HIWindowTrackProxyDrag` (page 1897)
Tracks the drag of a window proxy icon.

Establishing Proxy Icons

`GetWindowProxyAlias` (page 1862)
Obtains an alias for the file that is associated with a window.

`SetWindowProxyAlias` (page 1949)
Associates a file with a window.

`GetWindowProxyIcon` (page 1863)
Obtains a window's proxy icon.

`SetWindowProxyIcon` (page 1951)
Overrides the default proxy icon for a window.

`RemoveWindowProxy` (page 1923)
Dissociates a file from a window.

`SetWindowProxyCreatorAndType` (page 1950)
Sets the proxy icon for a window that lacks an associated file.

`HIWindowGetProxyFSRef` (page 1888)
Obtains the `FSRef` used to determine the proxy icon for a window.

`HIWindowSetProxyFSRef` (page 1894)
Sets the proxy icon for a window using an `FSRef` to a file system object.

Getting and Setting Window Structure Fields

`GetNextWindow` (page 1839)
Returns the next window in a window list.

`GetWindowKind` (page 1856)
Returns a window's window kind.

`SetWindowKind` (page 1945)
Sets a window's window kind.

`GetWindowPort` (page 1858)
Gets the window's color graphics port.

`SetPortWindowPort` (page 1932)
Sets the current graphics port to the window's port.

`GetWindowPortBounds` (page 1859)
Obtains the bounds of the window port.

`GetWindowStandardState` (page 1866)
Obtains a window's standard zoom rectangle.

`SetWindowStandardState` (page 1953)
Sets a window's standard zoom rectangle.

`GetWindowUserState` (page 1868)
Returns a window's user zoom rectangle.

`SetWindowUserState` (page 1954)
Sets a window's user zoom rectangle.

`IsWindowHilited` (page 1905)
Indicates whether the window frame is currently highlighted.

`IsWindowLatentVisible` (page 1906)
Indicates whether a window is visible onscreen or is latently visible but not currently onscreen.

`IsWindowVisible` (page 1910)
Indicates whether the window frame is currently visible.

`GetWindowStructurePort` (page 1866)
Obtains a graphics port that is used when drawing a window's structure.

`GetWindowStructureWidths` (page 1867)
Obtains the width of the structure region on each edge of a window.

Handling Mouse Events in Windows

`DragWindow` (page 1824)
Moves a window on the screen when the user drags it by its drag region.

`MoveWindow` (page 1910)
Moves a window on the desktop.

`PinRect` (page 1919)
Returns the point within the specified rectangle that is closest to the specified point.

`SizeWindow` (page 1959)
Sets the size of a window.

`TrackBox` (page 1961)
Tracks clicks in the collapse, close, size, and zoom boxes, and clicks of the toolbar button.

`TrackGoAway` (page 1962)
Tracks the cursor when the user presses the mouse button while the cursor is in the close box.

`ZoomWindow` (page 1971)
Zooms the window when the user has pressed and released the mouse button with the cursor in the zoom box.

Locating Windows

`ActiveNonFloatingWindow` (page 1800)
Returns the currently active nonfloating window.

`FrontNonFloatingWindow` (page 1829)
Returns to the application the frontmost visible window that is not a floating window.

`FindWindow` (page 1827)
Maps the location of the cursor to a part of the screen or a region of a window when your application receives a mouse-down event.

`FindWindowOfClass` (page 1828)
Finds a window of a specific class at the specified point onscreen.

`HIWindowFindAtLocation` (page 1882)
Finds a window in the current process at a specified location.

`GetFrontWindowOfClass` (page 1834)
Obtains the frontmost window of a given class.

`GetNextWindowOfClass` (page 1839)
Obtains the next window in a given window group.

`GetPreviousWindow` (page 1840)
Returns the window above the specified window in the window list.

`FrontWindow` (page 1830) Deprecated in Mac OS X v10.5
Identifies the frontmost visible window. (Deprecated. Use `ActiveNonFloatingWindow` (page 1800), `FrontNonFloatingWindow` (page 1829), or `GetFrontWindowOfClass` (page 1834) instead.)

Maintaining the Update Region

`BeginUpdate` (page 1801)
Starts updating a window when you receive an update event for that window.

`EndUpdate` (page 1826)
Finishes updating a window.

`InvalWindowRect` (page 1898)
Adds a rectangle to a window's update region.

`InvalWindowRgn` (page 1899)
Adds a region to a window's update region.

`IsWindowUpdatePending` (page 1909)
Determines whether a window update is pending.

`ValidWindowRect` (page 1969)
Removes a rectangle from a window's update region.

`ValidWindowRgn` (page 1969)
Removes a region from a window's update region.

Managing Activation Scope

`GetWindowActivationScope` (page 1841)
Obtains a window's activation scope.

`SetWindowActivationScope` (page 1935)
Sets a window's activation scope.

Managing Dock Tiles

`HIWindowCreateCollapsedDockTileContext` (page 1881)
Creates a Quartz graphics context for drawing a collapsed window's Dock tile.

`HIWindowReleaseCollapsedDockTileContext` (page 1892)
Releases a Quartz graphics context for drawing a collapsed window's Dock tile.

`GetWindowDockTileMenu` (page 1847)
Returns the menu to be displayed by a window's dock tile.

`SetWindowDockTileMenu` (page 1940)
Associates a pop-up menu with a window.

`UpdateCollapsedWindowDockTile` (page 1968)
Updates the image of a window in the dock to the current contents of the window.

`CreateQDContextForCollapsedWindowDockTile` (page 1816) Deprecated in Mac OS X v10.5
Obtains a `CGrafPtr` for a collapsed window's tile in the dock. (Deprecated. Use `HIWindowCreateCollapsedDockTileContext` (page 1881) instead.)

`ReleaseQDContextForCollapsedWindowDockTile` (page 1921) Deprecated in Mac OS X v10.5
Releases a port and other state created by `CreateQDContextForCollapsedWindowDockTile`. (Deprecated. Use `HIWindowReleaseCollapsedDockTileContext` (page 1892) instead.)

Managing Modality

`GetWindowModality` (page 1857)
Obtains the modality of a window.

`SetWindowModality` (page 1946)
Sets the modality of a window.

`HIWindowIsDocumentModalTarget` (page 1891)
Determines if a window is currently the target window of another document modal window, such as a sheet.

Managing Themes

`SetThemeWindowBackground` (page 1933)
Sets a window's background theme.

`SetThemeTextColorForWindow` (page 1933)
Sets a text color that contrasts with a theme brush.

`HIWindowGetThemeBackground` (page 1889) Deprecated in Mac OS X v10.4 Deprecated in Mac OS X v10.5
Gets the theme background brush for a window.

Managing Toolbars

`GetWindowToolbar` (page 1867)
Obtains the toolbar associated with a window.

`SetWindowToolbar` (page 1954)
Associates a toolbar with a window.

`ShowHideWindowToolbar` (page 1957)
Shows or hides the toolbar.

`IsWindowToolbarVisible` (page 1909)
Determines whether a window's toolbar is visible.

`HIWindowSetToolbarView` (page 1895) Deprecated in Mac OS X v10.4 Deprecated in Mac OS X v10.5
Sets a custom toolbar view for a window.

Managing Transitions

`TransitionWindow` (page 1965)
Shows, hides, moves, or resizes a window with appropriate animation and sound.

`TransitionWindowAndParent` (page 1966)
Shows or hides a window, potentially also moving a second window, with animation and sound.

`TransitionWindowWithOptions` (page 1967)
Transitions a window from one state to another with appropriate animation and sound.

Managing Transparency

`GetWindowAlpha` (page 1842)
Returns the current alpha channel value for the window.

`SetWindowAlpha` (page 1935)
Sets the window's alpha channel value.

Managing UPPs

`DisposeWindowDefUPP` (page 1821) Deprecated in Mac OS X v10.5
Disposes of the UPP for your window definition. (Deprecated. The WDEF interface is deprecated; use a custom HIView to draw your custom window frame instead.)

`DisposeWindowPaintUPP` (page 1822) Deprecated in Mac OS X v10.5
Disposes of the UPP to your region painting callback function. (Deprecated. The window content painting interface is deprecated; use a `kEventControlDraw` Carbon event handler on a compositing window's content view instead.)

`InvokeWindowDefUPP` (page 1900) Deprecated in Mac OS X v10.5
Invokes the UPP for a window definition. (Deprecated. The WDEF interface is deprecated; use a custom HIView to draw your custom window frame instead.)

`InvokeWindowPaintUPP` (page 1901) Deprecated in Mac OS X v10.5
Invokes the UPP for the specified painting region. (Deprecated. The window content painting interface is deprecated; use a `kEventControlDraw` Carbon event handler on a compositing window's content view instead.)

`NewWindowDefUPP` (page 1917) Deprecated in Mac OS X v10.5
Creates a new UPP for a window definition. (Deprecated. The WDEF interface is deprecated; use a custom HIView to draw your custom window frame instead.)

`NewWindowPaintUPP` (page 1917) Deprecated in Mac OS X v10.5
Creates a new UPP for a painting region. (Deprecated. The window content painting interface is deprecated; use a `kEventControlDraw` Carbon event handler on a compositing window's content view instead.)

Managing Window Attributes

`GetWindowAttributes` (page 1842)
Obtains the attributes of a window.

`GetAvailableWindowAttributes` (page 1830)
Returns the window attributes that are valid for a window class

`ChangeWindowAttributes` (page 1805)
Changes a window's attributes.

`HIWindowTestAttribute` (page 1896)
Returns a Boolean value indicating whether a window has a specified attribute.

`HIWindowChangeAttributes` (page 1874)
Changes the attributes of a window.

`HIWindowIsAttributeAvailable` (page 1890)
Returns a Boolean value indicating whether a window attribute is valid for a specified window class.

Managing Window Availability

`HIWindowChangeAvailability` (page 1875)
Changes the availability of a window during Exposé or in Spaces.

`HIWindowGetAvailability` (page 1884)
Obtains the availability of a window during Exposé or in Spaces.

Managing Window Classes

`GetWindowClass` (page 1844)
Obtains the class of a window.

`HIWindowChangeClass` (page 1876)
Changes the appearance and behavior of a window.

`IsValidWindowClass` (page 1901)
Determines whether a given window class is valid.

`SetWindowClass` (page 1938) Deprecated in Mac OS X v10.5
Sets the class of a window. (Deprecated. Use `HIWindowChangeClass` (page 1876), `SetWindowGroup` (page 1941), or `HIWindowChangeAttributes` (page 1874) instead.)

Managing Window Features

`GetWindowFeatures` (page 1847)
Obtains the features that a window supports.

`HIWindowChangeFeatures` (page 1877)
Changes a window's features.

Managing Window Focus

`SetUserFocusWindow` (page 1934)
Designates a window to receive user focus.

`GetUserFocusWindow` (page 1841)
Returns the current user focus window.

`HIWindowShowsFocus` (page 1896) Deprecated in Mac OS X v10.4 Deprecated in Mac OS X v10.4
Returns a Boolean value indicating whether a window's content should show focus indicators such as focus rings.

Managing Window Groups

`ChangeWindowGroupAttributes` (page 1806)
Changes the attributes of a window group.

`CopyWindowGroupName` (page 1812)
Obtains a copy of the window group name.

`CountWindowGroupContents` (page 1813)
Counts the number of members of a window group.

`CreateWindowGroup` (page 1818)
Creates a window group.

`DebugPrintAllWindowGroups` (page 1819)
Debugging utility function listing all window groups.

`DebugPrintWindowGroup` (page 1819)
Debugging utility functions for use with window groups.

`GetIndexedWindow` (page 1836)
Obtains the window at the given index in the window group.

`GetWindowGroup` (page 1849)
Obtains the window group associated with a window.

`GetWindowGroupAttributes` (page 1849)
Obtains the attributes of a window group.

`GetWindowGroupContents` (page 1850)
Obtains the contents of a window group.

`GetWindowGroupLevel` (page 1851)
Obtains the level of the group in the window class hierarchy.

`GetWindowGroupLevelOfType` (page 1851)
Obtains the Core Graphics window level of a window group.

`GetWindowGroupOfClass` (page 1852)
Obtains the window group corresponding to a given window class.

`GetWindowGroupOwner` (page 1853)
Obtains the window that owns a window group. (if any)

`GetWindowGroupParent` (page 1853)
Obtains the parent group of a window group.

`GetWindowGroupRetainCount` (page 1853)
Determines the current reference count for a window group.

`GetWindowGroupSibling` (page 1854)
Obtains the next or previous group of a window group.

`GetWindowIndex` (page 1855)
Obtains the index number of a specified window in a group.

`IsWindowContainedInGroup` (page 1904)
Determines if a window is a member of a window group or any of its subgroups.

`ReleaseWindowGroup` (page 1922)
Decrements the reference count for a window group.

`RetainWindowGroup` (page 1927)
Increments the reference count for a window group.

`SendWindowGroupBehind` (page 1930)
Orders one window group behind another.

`SetWindowGroup` (page 1941)
Assigns a window to a window group.

`SetWindowGroupLevel` (page 1942)
Sets the level of group in the window class hierarchy.

`SetWindowGroupLevelOfType` (page 1942)
Sets the window level of a window group.

`SetWindowGroupName` (page 1943)
Assigns a name to a window group.

`SetWindowGroupOwner` (page 1944)
Sets a window as the owner of a window group.

`SetWindowGroupParent` (page 1944)
Sets a window group to be the parent of another window group.

Managing Window Titles

`CopyWindowAlternateTitle` (page 1812)
Obtains a copy of the alternate window title.

`SetWindowAlternateTitle` (page 1936)
Sets an alternate window title.

`CopyWindowTitleAsCFString` (page 1813)
Copies the window title into a Core Foundation string.

`SetWindowTitleWithCFString` (page 1953)
Sets the window title to the contents of a Core Foundation string.

Manipulating Drawers

`OpenDrawer` (page 1917)
Opens a drawer.

`CloseDrawer` (page 1809)
Closes a drawer.

`GetDrawerCurrentEdge` (page 1832)
Obtains the current window edge from which the drawer appears.

`GetDrawerPreferredEdge` (page 1833)
Obtains the preferred opening edge for a drawer.

`SetDrawerPreferredEdge` (page 1932)
Set the preferred window edge from which the drawer should appear.

`GetDrawerOffsets` (page 1833)
Obtains the positioning offsets of a drawer.

`SetDrawerOffsets` (page 1931)
Sets the positioning offsets for the drawer with respect to its parent window.

`GetDrawerParent` (page 1833)
Obtains the parent window of a drawer.

`SetDrawerParent` (page 1931)
Sets the parent window for a drawer.

`GetDrawerState` (page 1834)
Determines the current state of the drawer.

`ToggleDrawer` (page 1961)
Toggles the drawer state.

`HIWindowCopyDrawers` (page 1879) Deprecated in Mac OS X v10.4
Obtains an array of the drawers that are attached to a window.

Manipulating Sheets

`GetSheetWindowParent` (page 1841)
Obtains the parent window of a sheet.

`ShowSheetWindow` (page 1958)
Shows a sheet window using appropriate visual effects.

`HideSheetWindow` (page 1872)
Hides a sheet window using appropriate visual effects.

`DetachSheetWindow` (page 1819)
Detaches a sheet window from its parent window.

Manipulating Window Color Information

`GetWindowContentColor` (page 1845)
Obtains the color to which a window's content region is redrawn.

`GetWindowContentPattern` (page 1846)
Obtains the pattern to which a window's content region is redrawn.

`SetWindowContentColor` (page 1938)
Sets the color to which a window's content region is redrawn.

`GetWRefCon` (page 1869)
Returns the reference constant from a window.

`SetWindowContentPattern` (page 1939)
Sets the pattern to which a window's content region is redrawn.

`SetWRefCon` (page 1955)
Sets the `refCon` field of a window.

`GetWindowPic` (page 1858) Deprecated in Mac OS X v10.5
Returns a handle to a window's picture. (Deprecated. Use an HIImageView object to draw a window's content and ask the view for its image instead.)

`SetWindowPic` (page 1947) Deprecated in Mac OS X v10.5
Sets a picture for the Window Manager to draw in a window's content region. (Deprecated. Use an HIImageView object to draw a window's content instead.)

Referencing Windows

`CloneWindow` (page 1808) Deprecated in Mac OS X v10.5
Increments the number of references to a window. (Deprecated. Use `CFRetain` instead.)

`GetWindowOwnerCount` (page 1857) Deprecated in Mac OS X v10.5
Obtains the number of existing references to a window. (Deprecated. Use `CFGetRetainCount` instead.)

`GetWindowRetainCount` (page 1865) Deprecated in Mac OS X v10.5
Returns the retain count of a window. (Deprecated. Use `CFGetRetainCount` instead.)

`ReleaseWindow` (page 1922) Deprecated in Mac OS X v10.5
Decrements the retain count of a window, and destroys the window if the retain count falls to zero. (Deprecated. Use `CFRelease` instead.)

`RetainWindow` (page 1926) Deprecated in Mac OS X v10.5
Increments the retain count of a window. (Deprecated. Use `CFRetain` instead.)

Scrolling

`ScrollWindowRect` (page 1927)
Scroll any area of a window.

`ScrollWindowRegion` (page 1928)
Scrolls a window's region.

Sizing and Positioning Windows

`GetWindowBounds` (page 1843)
Obtains the size and position of the bounding rectangle of the specified window region.

`HIWindowGetBounds` (page 1885)
Gets the bounds of a specified region of a window.

`SetWindowBounds` (page 1936)
Sets a window's size and position from the bounding rectangle of the specified window region.

`HIWindowSetBounds` (page 1893)
Sets the bounds of a window based on either the structure or content region.

`MoveWindowStructure` (page 1911)
Positions a window relative to its structure region.

`RepositionWindow` (page 1924)
Positions a window relative to another window or a display screen.

`ResizeWindow` (page 1925)
Handles all user interaction while a window is being resized.

`GetAvailableWindowPositioningBounds` (page 1831)
Obtains the available window positioning bounds.

`HIWindowGetAvailablePositioningBounds` (page 1885)
Gets the available window positioning bounds on a display.

`GetAvailableWindowPositioningRegion` (page 1832)
Obtains the available window positioning region.

`HIWindowCopyAvailablePositioningShape` (page 1878)
Copies the available window positioning shape on a display.

`HIWindowConstrain` (page 1877)
Moves and resizes a window to be within a specified bounding rectangle.

`GetWindowResizeLimits` (page 1865)
Returns the minimum and maximum content sizes for a window.

`SetWindowResizeLimits` (page 1952)
Sets the maximum and minimum resize limits for windows.

`ConstrainWindowToScreen` (page 1811) Deprecated in Mac OS X v10.4
Moves and resizes a window so that it's contained entirely on a single screen.

Updating the Screen

`EnableScreenUpdates` (page 1825)
Enables screen updates for changes to the current application's windows.

`DisableScreenUpdates` (page 1820)
Disables updates for changes to the current application's windows.

Using Default and Cancel Buttons

You can use these functions to add dialog-like button controls to normal windows.

`SetWindowDefaultButton` (page 1940)
Specifies a default button for a window.

`GetWindowDefaultButton` (page 1846)
Returns the current default button for a window.

`SetWindowCancelButton` (page 1937)
Specifies a Cancel button for a window.

`GetWindowCancelButton` (page 1844)
Returns the current Cancel button for a window.

Zooming Windows

`HIWindowGetIdealUserState` (page 1887)
Gets the bounds of a window's content region in its user state.

`IsWindowInStandardState` (page 1905)
Determines whether a window is currently zoomed in to the user state or zoomed out to the standard state.

`HIWindowIsInStandardState` (page 1891)
Returns a Boolean value indicating whether a window is zoomed out to its standard state.

`HIWindowSetIdealUserState` (page 1894)
Sets the bounds of a window's content region in its user state.

`ZoomWindowIdeal` (page 1972)
Zooms a window in accordance with human interface guidelines.

`GetWindowIdealUserState` (page 1854) Deprecated in Mac OS X v10.4
Obtains the size and position of a window in its user state.

`SetWindowIdealUserState` (page 1945) Deprecated in Mac OS X v10.4
Sets the size and position of a window in its user state.

Miscellaneous

`CreateStandardWindowMenu` (page 1816)
Creates a standard window menu for your application.

`GetWindowFromPort` (page 1848)
Gets a window reference from a `CGrafPtr` data type.

`HIWindowFlush` (page 1883)
Flushes any dirty areas a window might have.

`RegisterWindowDefinition` (page 1920)
Registers a binding between a resource ID and a window definition function.

`ReshapeCustomWindow` (page 1924)
Notifies the Window Manager that a custom window's shape has changed.

`InstallWindowContentPaintProc` (page 1898) Deprecated in Mac OS X v10.5
Installs a window content painting callback. (Deprecated. Use a `kEventControlDraw` Carbon event handler on a window's content view instead.)

Legacy Functions

`CalcVis` (page 1804) Deprecated in Mac OS X v10.5
Calculates the visible region of a window. (Deprecated. There is no replacement function.)

`CalcVisBehind` (page 1804) Deprecated in Mac OS X v10.5
Calculates the visible regions of a series of windows. (Deprecated. There is no replacement function.)

`CheckUpdate` (page 1807) Deprecated in Mac OS X v10.5
Scans the window list for windows that need updating. (Deprecated. Use `FindSpecificEventInQueue` (page 258) or `AcquireFirstMatchingEventInQueue` (page 245) instead.)

`ClipAbove` (page 1808) Deprecated in Mac OS X v10.5
Determines the clip region of the Window Manager port. (Deprecated. There is no replacement function.)

`DragGrayRgn` (page 1822) Deprecated in Mac OS X v10.5
Moves a gray outline of a region on the screen, following the movements of the cursor, until the mouse button is released. (Deprecated. Use an overlay window or other custom drawing instead.)

`DragTheRgn` (page 1824) Deprecated in Mac OS X v10.5
Tracks the mouse as the user drags the outline of a region. (Deprecated. Use an overlay window or other custom drawing instead.)

`GetGrayRgn` (page 1835) Deprecated in Mac OS X v10.5
Returns a region that covers the desktop area of all active displays. (Deprecated. To determine the area in which a window may be positioned, use `HIWindowGetAvailablePositioningBounds` (page 1885) or `HIWindowCopyAvailablePositioningShape` (page 1878).)

`GetNewCWindow` (page 1836) Deprecated in Mac OS X v10.5
Creates a color window from a window resource. (Deprecated. Use nib files and `CreateWindowFromNib` (page 1059) instead.)

`GetNewWindow` (page 1838) Deprecated in Mac OS X v10.5
Creates a window from a window resource. (Deprecated. Use nib files and `CreateWindowFromNib` (page 1059) instead.)

`GetWindowProxyFSSpec` (page 1862) Deprecated in Mac OS X v10.5
Obtains a file system specification structure for the file that is associated with a window. (Deprecated. Use `HIWindowGetProxyFSRef` (page 1888) instead.)

`GetWTitle` (page 1869) Deprecated in Mac OS X v10.5
Retrieves the title of a window as a Pascal string. (Deprecated. Use `CopyWindowTitleAsCFString` (page 1813) instead.)

`GetWVariant` (page 1870) Deprecated in Mac OS X v10.5
Returns a window's variation code. (Deprecated. Use `GetWindowAttributes` (page 1842) to determine aspects of a window's appearance or behavior.)

`GrowWindow` (page 1870) Deprecated in Mac OS X v10.5
Allows the user to change the size of a window. (Deprecated. Use `ResizeWindow` (page 1925) instead.)

`NewCWindow` (page 1912) Deprecated in Mac OS X v10.5
Creates a window with a specified list of characteristics. (Deprecated. Use `CreateNewWindow` (page 1815) instead.)

`NewWindow` (page 1914) Deprecated in Mac OS X v10.5
Creates a window from a parameter list. (Deprecated. Use `CreateNewWindow` (page 1815) instead.)

`PaintBehind` (page 1918) Deprecated in Mac OS X v10.5
Redraws a series of windows in the window list. (Deprecated. Use `InvalWindowRect` (page 1898), `InvalWindowRgn` (page 1899), or `HIViewSetNeedsDisplay` (page 2485) to invalidate a portion of a window.)

`PaintOne` (page 1919) Deprecated in Mac OS X v10.5
: Redraws the invalid, exposed portions of one window on the desktop. (Deprecated. Use `InvalWindowRect` (page 1898), `InvalWindowRgn` (page 1899), or `HIViewSetNeedsDisplay` (page 2485) to invalidate a portion of a window.)

`SetWindowProxyFSSpec` (page 1950) Deprecated in Mac OS X v10.5
: Associates a file with a window. (Deprecated. Use `HIWindowSetProxyFSRef` (page 1894) instead.)

`SetWTitle` (page 1956) Deprecated in Mac OS X v10.5
: Specifies a window's title. (Deprecated. Use `SetWindowTitleWithCFString` (page 1953) instead.)

`DrawGrowIcon` (page 1825) Deprecated in Mac OS X v10.4 Deprecated in Mac OS X v10.5
: Draws a grow icon in the window frame. (Deprecated. There is no replacement function.)

Functions

ActivateWindow

Activates or deactivates a window.

```
OSStatus ActivateWindow (
   WindowRef inWindow,
   Boolean inActivate
);
```

Parameters

inWindow
: The window to activate or deactivate.

inActivate
: Pass `true` to activate the window, `false` otherwise.

Return Value

A result code. See "Window Manager Result Codes" (page 2051).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

BSDLLCTest

Declared In

`MacWindows.h`

ActiveNonFloatingWindow

Returns the currently active nonfloating window.

```
WindowRef ActiveNonFloatingWindow (
   void
);
```

Return Value
A reference to the active window.

Discussion
Note that the active window is not necessarily the frontmost window, and it is not necessarily the window with user focus. Call `GetUserFocusWindow` (page 1841) to get the window that has user focus.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

AreFloatingWindowsVisible

Indicates whether an application's floating windows are currently visible.

```
Boolean AreFloatingWindowsVisible (
   void
);
```

Return Value
A Boolean whose value is `true` if the application's floating windows are currently shown or `false` if the application's floating windows are currently hidden.

Discussion
This function checks the visibility state of an application's floating windows, which are hidden automatically when the application receives a suspend event and are made visible automatically when the application receives a resume event.

Special Considerations

The `AreFloatingWindowsVisible` function operates only upon windows created with the `kFloatingWindowClass` constant; see "Window Class Constants" (page 1988) for more details on this constant.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

BeginUpdate

Starts updating a window when you receive an update event for that window.

```
void BeginUpdate (
   WindowRef window
);
```

Parameters

window

The window that is to be updated when an update event is received. Your application gets this information from the `message` field in the update event structure.

Discussion

The `BeginUpdate` function limits the visible region of the window's graphics port to the intersection of the visible region and the update region it then sets the window's update region to an empty region. After calling `BeginUpdate`, your application redraws either the entire content region or only the visible region. In either case, only the parts of the window that require updating are actually redrawn on the screen.

Every call to `BeginUpdate` must be matched with a subsequent call to `EndUpdate` (page 1826) after your application redraws the content region. `BeginUpdate` and `EndUpdate` can't be nested. That is, you must call `EndUpdate` before the next call to `BeginUpdate`.

In Mac OS X, you only receive one update event. If you don't call `BeginUpdate`, you won't receive any further update events until the window is invalidated again.

Special Considerations

This function should not be used on composited windows. Modifying a composited window's update region does not affect the area of the window to be drawn. A composited window does not use its window update region to control drawing. Instead, a composited window determines what to draw by looking at the invalid regions of the views contained in the window.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

HideMenuBar

Simple DrawSprocket

Declared In

`MacWindows.h`

BeginWindowProxyDrag

Creates the drag reference and the drag image when the user drags a proxy icon.

```
OSStatus BeginWindowProxyDrag (
   WindowRef window,
   DragRef *outNewDrag,
   RgnHandle outDragOutlineRgn
);
```

Parameters

window

The window whose proxy icon is being dragged.

outNewDrag

On input, a pointer to a value of type `DragRef`. On return, the value refers to the current drag process.

outDragOutlineRgn

On input, a value of type `RgnHandle`. Your application can create this handle with a call to the QuickDraw function `NewRgn`. On return, this region is set to the outline of the icon being dragged.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
Typically, if the proxy icon represents a type of object (currently, file system entities such as files, folders, and volumes) supported by the Window Manager, the Window Manager can handle all aspects of the drag process itself, and your application should call the function `TrackWindowProxyDrag` (page 1963). However, if the proxy icon represents a type of data that the Window Manager does not support, or if you want to implement custom dragging behavior, your application should call the function `TrackWindowProxyFromExistingDrag` (page 1964).

The `TrackWindowProxyFromExistingDrag` (page 1964) function accepts an existing drag reference and adds file data if the window contains a file proxy. If your application uses `TrackWindowProxyFromExistingDrag`, you then have the choice of using this function in conjunction with the functions `BeginWindowProxyDrag` and `EndWindowProxyDrag` (page 1826) or simply calling `TrackWindowProxyFromExistingDrag` and handling all aspects of creating and disposing of the drag yourself.

Specifically, your application can call `BeginWindowProxyDrag` to set up the drag image and drag reference. Your application must then track the drag, using `TrackWindowProxyFromExistingDrag`, and do any required moving of data and, finally, call `EndWindowProxyDrag` (page 1826) to dispose of the drag reference. `BeginWindowProxyDrag` should not be used for types handled by the Window Manager unless the application wants to implement custom dragging behavior for those types.

Your application detects a drag when the function `FindWindow` (page 1827) returns the `inProxyIcon` result code.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

BringToFront

Brings a window to the front.

```
void BringToFront (
   WindowRef window
);
```

Parameters

window

The window that is to be brought to the front.

Discussion
The `BringToFront` function puts the specified window at the beginning of the window list and redraws the window in front of all others on the screen. It does not change the window's highlighting or make it active.

Your application does not ordinarily call `BringToFront`. The user interface guidelines specify that the frontmost window should be the active window. To bring a window to the front and make it active, call the function `SelectWindow` (page 1929).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
BSDLLCTest

Declared In
`MacWindows.h`

CalcVis

Calculates the visible region of a window. (Deprecated in Mac OS X v10.5. There is no replacement function.)

```
void CalcVis (
   WindowRef window
);
```

Parameters
window
On input, a pointer to the window's complete window structure.

Discussion
The Window Manager calls the `CalcVis` function; your application does not normally need to. `CalcVis` calculates the visible region of the specified window by starting with its content region and subtracting the structure region of each window in front of it.

Special Considerations
In Mac OS X, the visible region of a window is managed by the window server. Applications never need to call this function.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

CalcVisBehind

Calculates the visible regions of a series of windows. (Deprecated in Mac OS X v10.5. There is no replacement function.)

```
void CalcVisBehind (
   WindowRef startWindow,
   RgnHandle clobberedRgn
);
```

Parameters

startWindow

On input, a pointer to a window structure.

clobberedRgn

On input, a handle to the desktop region that has become invalid.

Discussion

The Window Manager calls the `CalcVisBehind` function; your application does not normally need to. `CalcVisBehind` calculates the visible regions of the window specified by the `startWindow` parameter and all windows behind `startWindow` that intersect `clobberedRgn`. It is called after `PaintBehind`.

Special Considerations

In Mac OS X, the visible region of a window is managed by the window server. Applications never need to call this function.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

ChangeWindowAttributes

Changes a window's attributes.

```
OSStatus ChangeWindowAttributes (
   WindowRef window,
   WindowAttributes setTheseAttributes,
   WindowAttributes clearTheseAttributes
);
```

Parameters

window

The window whose attributes you want to change.

setTheseAttributes

The attributes you want to set. Pass `kWindowNoAttributes` if you do not want to set any attributes. See "Window Attributes" (page 1998) for a list of window attributes.

clearTheseAttributes

The attributes you want to clear (if any). Pass `kWindowNoAttributes` if you do not want to clear any attributes. See "Window Attributes" (page 1998) for a list of window attributes.

Return Value

A result code. See "Window Manager Result Codes" (page 2051).

Discussion
If the changed attributes affect the visible window's frame, the window regions are recalculated and the window is redrawn.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
QTCarbonShell

Declared In
`MacWindows.h`

ChangeWindowGroupAttributes

Changes the attributes of a window group.

```
OSStatus ChangeWindowGroupAttributes (
   WindowGroupRef inGroup,
   WindowGroupAttributes setTheseAttributes,
   WindowGroupAttributes clearTheseAttributes
);
```

Parameters

inGroup
The window group whose attributes you want to set.

setTheseAttributes
The attributes you want to set. See "Window Group Attributes" (page 2031) for a list of possible attributes.

clearTheseAttributes
The attributes you want to clear (if any). See "Window Group Attributes" (page 2031) for a list of possible attributes.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

ChangeWindowPropertyAttributes

Changes attributes associated with a window property.

```
OSStatus ChangeWindowPropertyAttributes (
   WindowRef window,
   OSType propertyCreator,
   OSType propertyTag,
   OptionBits attributesToSet,
   OptionBits attributesToClear
);
```

Parameters

window
The window whose property attributes are to be changed.

propertyCreator
The property creator.

propertyTag
The property tag.

attributesToSet
The attributes to set. For a possible value, see "Window Property Persistent Constant" (page 2024).

attributesToClear
The attributes to clear. For a possible value, see "Window Property Persistent Constant" (page 2024).

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

CheckUpdate

Scans the window list for windows that need updating. (Deprecated in Mac OS X v10.5. Use `FindSpecificEventInQueue` (page 258) or `AcquireFirstMatchingEventInQueue` (page 245) instead.)

```
Boolean CheckUpdate (
   EventRecord *theEvent
);
```

Parameters

theEvent
On input, a pointer to an event structure to be filled in if a window needs updating.

Return Value
A Boolean value. If `CheckUpdate` finds a window structure whose update region is not empty and whose window structure does not contain a picture handle, it stores an update event in the event structure referenced through the parameter `theEvent` and returns `true`. If it finds no such window, it returns `false`.

Discussion
The Event Manager calls the `CheckUpdate` function; your application does not normally need to. `CheckUpdate` scans the window list from front to back, checking for a visible window that needs updating (that is, a visible window whose update region is not empty). If it finds one whose window structure contains a picture handle, it redraws the window itself and continues through the list.

Special Considerations

If you are using a compositing window, the Window Manager never generates update events for the window and you will never find update events in the event queue.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

ClipAbove

Determines the clip region of the Window Manager port. (Deprecated in Mac OS X v10.5. There is no replacement function.)

```
void ClipAbove (
   WindowRef window
);
```

Parameters

window

On input, a pointer to the window.

Discussion

The Window Manager calls the `ClipAbove` function; your application does not normally need to. `ClipAbove` sets the clip region of the Window Manager port to be the area of the desktop that intersects the current clip region, minus the structure regions of all the windows in front of the specified window.

`ClipAbove` retrieves the desktop region from the global variable `GrayRgn`.

Special Considerations

Mac OS X applications never need to call this function.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

CloneWindow

Increments the number of references to a window. (Deprecated in Mac OS X v10.5. Use `CFRetain` instead.)

```
OSStatus CloneWindow (
   WindowRef window
);
```

Parameters

window

The window whose reference count is to be incremented.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
You should call `CloneWindow` if you are using a window and want to ensure that it is not disposed while you are using it. With the Window Manager in Mac OS 8.5 and later, all windows are created with a reference count (owner count) of one. The function `CloneWindow` increments the number of references to a window, and the earlier function `DisposeWindow` decrements the number of references. When the reference count reaches zero, `DisposeWindow` disposes of the window.

In Mac OS X v10.2 and later, you can also call `CFRetain` to increment the reference count of a window.

Special Considerations

To maintain an accurate reference count, you must follow every call to the `CloneWindow` function with a matching call to the `DisposeWindow` function when your application is ready to release its reference to the window.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

CloseDrawer

Closes a drawer.

```
OSStatus CloseDrawer (
   WindowRef inDrawerWindow,
   Boolean inAsync
);
```

Parameters

inDrawerWindow

The drawer window that is to be closed.

inAsync

Pass `true` for asynchronous closing; otherwise, pass `false`.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
`CloseDrawer` may close the drawer either synchronously or asynchronously, depending on the value of the `inAsync` parameter. If `inAsync` is true, `CloseDrawer` installs an event loop timer that closes the drawer after `CloseDrawer` returns to the caller; therefore, the caller must be running its event loop for the drawer to close. If `inAsync` is false, `CloseDrawer` closes the drawer completely before returning to the caller. `CloseDrawer` retains the drawer window while the drawer is closing, and releases it when the drawer is fully closed. `CloseDrawer` sends the `kEventWindowDrawerClosing` event to the drawer, the drawer's parent, and the application before closing the drawer. If an event handler for this event returns `userCanceledErr`, `CloseDrawer` will return immediately without closing the drawer. `CloseDrawer` sends the `kEventWindowDrawerClosed` event to the drawer, the drawer's parent, and the application after the drawer has finished closing.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

CollapseAllWindows

Collapses or expands all collapsable windows in an application.

```
OSStatus CollapseAllWindows (
   Boolean collapse
);
```

Parameters
collapse
Set to `true` to collapse all windows in the application. Set to `false` to expand all windows in the application.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
In Mac OS X, this function works with any window that has the `kWindowCollapseBoxAttribute`. If that attribute is not present, the Window Manager checks for the `kWindowCanCollapse` feature bit.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

CollapseWindow

Collapses or expands a window to the dock.

```
OSStatus CollapseWindow (
   WindowRef window,
   Boolean collapse
);
```

Parameters

window

The window that is to be collapsed or expanded.

collapse

Indicates whether the window should be collapsed or expanded.

Return Value

A result code. See "Window Manager Result Codes" (page 2051).

Discussion

The `CollapseWindow` function tells the Window Manager to collapse or expand a window depending upon the value passed in the `collapse` parameter. In Mac OS X, any window that has the `kWindowCollapseBoxAttribute` can be collapsed. If that attribute is not present, the Window Manager checks for the `kWindowCanCollapse` feature bit.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

ConstrainWindowToScreen

Moves and resizes a window so that it's contained entirely on a single screen.

```
OSStatus ConstrainWindowToScreen (
    WindowRef inWindowRef,
    WindowRegionCode inRegionCode,
    WindowConstrainOptions inOptions,
    const Rect *inScreenRect,
    Rect *outStructure
);
```

Parameters

inWindowRef

The window to constrain.

inRegionCode

The window region to constrain. See "Window Region Constants" (page 2021) for a list of possible constants to pass.

inOptions

Flags controlling how the window is constrained.

inScreenRect

A rectangle, in global coordinates, in which to constrain the window. May be `NULL`. If `NULL`, the window is constrained to the screen with the greatest intersection with the specified window region.

outStructure

On exit, contains the new structure bounds of the window, in global coordinates. May be `NULL`.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

See Also
`HIWindowConstrain` (page 1877)

Declared In
`MacWindows.h`

CopyWindowAlternateTitle

Obtains a copy of the alternate window title.

```
OSStatus CopyWindowAlternateTitle (
   WindowRef inWindow,
   CFStringRef *outTitle
);
```

Parameters

`inWindow`
The window to get the alternate title from.

`outTitle`
Receives the alternate title for the window. If the window does not have an alternate title, `NULL` will be returned in `outTitle`.

Return Value
A result code. See "Window Manager Result Codes" (page 2051). An operating system status code.

Discussion
See the discussion of `SetWindowAlternateTitle` (page 1936) for more information about alternate window titles.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

CopyWindowGroupName

Obtains a copy of the window group name.

```
OSStatus CopyWindowGroupName (
   WindowGroupRef inGroup,
   CFStringRef *outName
);
```

Parameters

inGroup

The window group to query. For information on this data type,

outName

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
MacWindows.h

CopyWindowTitleAsCFString

Copies the window title into a Core Foundation string.

```
OSStatus CopyWindowTitleAsCFString (
   WindowRef inWindow,
   CFStringRef *outString
);
```

Parameters

inWindow

The window whose title is to be copied.

outString

On output, the window's title.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
CarbonSketch
QTCarbonShell

Declared In
MacWindows.h

CountWindowGroupContents

Counts the number of members of a window group.

```
ItemCount CountWindowGroupContents (
   WindowGroupRef inGroup,
   WindowGroupContentOptions inOptions
);
```

Parameters

inGroup

The window group whose members are to be counted.

inOptions

Counting options. See "Window Group Content Options" (page 2032) for possible constants to pass.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
MacWindows.h

CreateCustomWindow

Creates a custom window based on a registered toolbox object class or a custom window root view.

```
OSStatus CreateCustomWindow (
   const WindowDefSpec *def,
   WindowClass windowClass,
   WindowAttributes attributes,
   const Rect *contentBounds,
   WindowRef *outWindow
);
```

Parameters

def

For information on this data type, see `WindowDefSpec` (page 1985).

windowClass

The class the custom window should belong to. This value determines the layer ordering of the custom window.

attributes

Attributes for the window. See "Window Attributes" (page 1998) for a list of possible attributes.

contentBounds

Pointer to a `Rect` structure in global coordinates indicating the dimensions of the window's content region.

outWindow

On return, the newly-created window.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

See Also
`HIWindowCreate` (page 1880)

Declared In
MacWindows.h

CreateNewWindow

Creates a window from parameter data.

```
OSStatus CreateNewWindow (
   WindowClass windowClass,
   WindowAttributes attributes,
   const Rect *contentBounds,
   WindowRef *outWindow
);
```

Parameters

windowClass
A constant that categorizes the class of window to be created. For certain classes, the window class can be altered after the window is created by calling HIWindowChangeClass (page 1876). See "Window Class Constants" (page 1988) for a description of possible values for this parameter.

attributes
Attributes for the window. See "Window Attributes" (page 1998) for a list of possible attributes.

contentBounds
Pointer to a Rect structure in global coordinates indicating the dimensions of the window's content region.

outWindow
On input, a pointer to a value of type WindowRef. On return, the window pointer points to the newly created window.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
The CreateNewWindow function creates a window based on the attributes and class you specify in the attributes and windowClass parameters. CreateNewWindow sets the new window's content region to the size and location specified by the rectangle passed in the bounds parameter, which in turn determines the dimensions of the entire window. The Window Manager creates the window invisibly and places it at the front of the window's window group. After calling CreateNewWindow, you should set any desired associated data—using Window Manager or Control Manager accessor functions—then call the function TransitionWindow (page 1965) or ShowWindow (page 1959) to display the window.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

See Also
HIWindowCreate (page 1880)

Related Sample Code
CarbonSketch

Declared In
MacWindows.h

CreateQDContextForCollapsedWindowDockTile

Obtains a `CGrafPtr` for a collapsed window's tile in the dock. (Deprecated in Mac OS X v10.5. Use `HIWindowCreateCollapsedDockTileContext` (page 1881) instead.)

```
OSStatus CreateQDContextForCollapsedWindowDockTile (
   WindowRef inWindow,
   CGrafPtr *outContext
);
```

Parameters

inWindow

The window whose `CGrafPtr` is to be obtained.

outContext

On output, a pointer to the window's `CGrafPtr`.

Return Value

A result code. See "Window Manager Result Codes" (page 2051).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Related Sample Code

QTCarbonShell

Declared In

`MacWindows.h`

CreateStandardWindowMenu

Creates a standard window menu for your application.

```
OSStatus CreateStandardWindowMenu (
   OptionBits inOptions,
   MenuRef *outMenu
);
```

Parameters

inOptions

Option bits. Pass 0 or `kWindowMenuIncludeRotate`. For information on the `kWindowMenuIncludeRotate` constant, see "Window Menu Item Property Constants" (page 2042).

outMenu

On output, a new menu reference that contains the standard window menu items and commands.

Return Value

A result code. See "Window Manager Result Codes" (page 2051). An operating system status code.

Discussion

You can call this function to create a window menu for your application. To register a window to be tracked by this menu, you either create your window with `CreateNewWindow` (page 1815), passing the `kWindowInWindowMenuAttribute`, or you can use `ChangeWindowAttributes` (page 1805) after the window

is created. The Toolbox takes care of acting on the standard items such as zoom and minimize, as well as bringing selected windows to the front. All you need to do is insert the menu in your menu bar (typically at the end of your menu list) and register your windows, and the Toolbox does the rest.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

CreateWindowFromCollection

Creates a window from collection data. (Deprecated in Mac OS X v10.5. Use `HIArchiveCopyDecodedCFType` (page 2310) to decode a window from an archive instead.)

```
OSStatus CreateWindowFromCollection (
   Collection collection,
   WindowRef *outWindow
);
```

Parameters

collection

A reference to the collection to be used in creating the window. You pass a reference to a previously created collection, such as that returned by the Collection Manager function `NewCollection`. The collection used to create the window must contain the required items for a resource of type `'wind'` or window creation fails.

outWindow

On input, a pointer to a value of type `WindowRef`. On return, the window pointer points to the newly created window.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
This function creates a window invisibly and places it at the front of the window's window group. After calling `CreateWindowFromCollection`, you should set any desired associated data—using Window Manager or Control Manager accessor functions—then call the function `TransitionWindow` (page 1965) or `ShowWindow` (page 1959) to display the window. The number of references to the collection (that is, its owner count) is incremented by a minimum of one for the duration of this call.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

CreateWindowFromResource

Creates a window from `'wind'` resource data. (Deprecated in Mac OS X v10.5. Use nib files and `CreateWindowFromNib` (page 1059) instead.)

```
OSStatus CreateWindowFromResource (
   SInt16 resID,
   WindowRef *outWindow
);
```

Parameters

resID

The resource ID of a resource of type `'wind'`. Pass in the ID of the `'wind'` resource to be used to create the window.

outWindow

On input, a pointer to a value of type `WindowRef`. On return, the window pointer points to the newly created window.

Return Value

A result code. See "Window Manager Result Codes" (page 2051).

Discussion

The `CreateWindowFromResource` function loads a window from a `'wind'` resource. The Window Manager creates the window invisibly and places it at the front of the window's window group. After calling `CreateWindowFromResource`, you should set any desired associated data—using Window Manager or Control Manager accessor functions—then call the function `TransitionWindow` (page 1965) or `ShowWindow` (page 1959) to display the window.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

CreateWindowGroup

Creates a window group.

```
OSStatus CreateWindowGroup (
   WindowGroupAttributes inAttributes,
   WindowGroupRef *outGroup
);
```

Parameters

inAttributes

Attributes for the new window group. See "Window Group Attributes" (page 2031) for a listing of possible attributes.

outGroup

For information on this data type, see WindowGroupRef (page 1986).

Return Value

A result code. See "Window Manager Result Codes" (page 2051).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

DebugPrintAllWindowGroups

Debugging utility function listing all window groups.

```
void DebugPrintAllWindowGroups (
   void
);
```

Carbon Porting Notes

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`HIToolboxDebugging.h`

DebugPrintWindowGroup

Debugging utility functions for use with window groups.

```
void DebugPrintWindowGroup (
   WindowGroupRef inGroup
);
```

Parameters

inGroup
The window group. For information on this data type,

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`HIToolboxDebugging.h`

DetachSheetWindow

Detaches a sheet window from its parent window.

```
OSStatus DetachSheetWindow (
   WindowRef inSheet
);
```

Parameters

inSheet

The window sheet that is to be detached from its parent window.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
This function detaches a sheet window from its parent window without affecting the visibility or position of the sheet or its parent. This function is useful for hiding a sheet window without an animation effect. To do so, call `DetachSheetWindow` to detach the sheet from the parent, and then call `HideWindow` (page 1872) to hide the sheet. Call `DisposeWindow` (page 1820) to destroy the sheet.

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

DisableScreenUpdates

Disables updates for changes to the current application's windows.

```
OSStatus DisableScreenUpdates (
   void
);
```

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

DisposeWindow

Removes a window.

```
void DisposeWindow (
   WindowRef window
);
```

Parameters

window

On input, the window to be closed.

Discussion
The `DisposeWindow` function reduces a window's reference count by one. If the resulting reference count is zero, then `DisposeWindow` removes the window from the screen and deletes it from the window list, then releases the memory occupied by all structures associated with the window, including the window structure.

Note that `DisposeWindow` assumes that any picture pointed to by the window structure field `windowPic` is data, not a resource, and it calls the QuickDraw function `KillPicture` to delete it. If your application uses a picture stored as a resource, you must release the resource or release the memory it occupies with the `ReleaseResource` function and set the `windowPic` field to `NULL` before closing the window.

Any pending update events for the window are discarded. If the window being removed is the frontmost window, the window behind it, if any, becomes the active window.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
HID Calibrator
HID Config Save
HID Explorer
QTCarbonShell
QTMetaData

Declared In
`MacWindows.h`

DisposeWindowDefUPP

Disposes of the UPP for your window definition. (Deprecated in Mac OS X v10.5. The WDEF interface is deprecated; use a custom HIView to draw your custom window frame instead.)

```
void DisposeWindowDefUPP (
   WindowDefUPP userUPP
);
```

Parameters
`userUPP`
The UPP that is to be disposed of.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

DisposeWindowPaintUPP

Disposes of the UPP to your region painting callback function. (Deprecated in Mac OS X v10.5. The window content painting interface is deprecated; use a `kEventControlDraw` Carbon event handler on a compositing window's content view instead.)

```
void DisposeWindowPaintUPP (
   WindowPaintUPP userUPP
);
```

Parameters

userUPP

The UPP that is to be disposed of.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`MacWindows.h`

DragGrayRgn

Moves a gray outline of a region on the screen, following the movements of the cursor, until the mouse button is released. (Deprecated in Mac OS X v10.5. Use an overlay window or other custom drawing instead.)

```
long DragGrayRgn (
   RgnHandle theRgn,
   Point startPt,
   const Rect *limitRect,
   const Rect *slopRect,
   short axis,
   DragGrayRgnUPP actionProc
);
```

Parameters

theRgn

On input, a handle to the region to be dragged.

startPt

On input, the location, in the local coordinates of the current graphics port, of the cursor when the mouse button was pressed.

limitRect

On input, a pointer to a rectangle, given in the local coordinates of the current graphics port, that limits where the region can be dragged. This parameter works with the `slopRect` parameter.

slopRect

On input, a pointer to a rectangle, given in the local coordinates of the current graphics port, that gives the user some leeway in moving the mouse without violating the limits of the `limitRect` parameter. The `slopRect` rectangle should be larger than the `limitRect` rectangle.

axis

On input, a constant that constrains the region's motion. The `axis` parameter can have one of three values: `noConstraint` (0), `hAxisOnly` (1), or `vAxisOnly` (2).

If an axis constraint is in effect, the outline follows the cursor's movements along only the specified axis, ignoring motion along the other axis. With or without an axis constraint, the outline appears only when the mouse is inside the `slopRect` rectangle.

actionProc

On input, a pointer to a function that defines an action to be performed repeatedly as long as the user holds down the mouse button. The function can have no parameters. If the value of `actionProc` is `null`, `DragGrayRgn` simply retains control until the mouse button is released.

Return Value
A long integer that specifies the difference between the point where the mouse button was pressed and the offset point.

Discussion
The `DragGrayRgn` function is called by `DragWindow` to move an outline of a window around the screen as the user drags a window. It returns the difference between the point where the mouse button was pressed and the offset point (the point in the region whose horizontal and vertical offsets from the upper-left corner of the region's enclosing rectangle are the same as the offsets of the starting point when the user pressed the mouse button. `DragGrayRgn` stores the vertical difference between the starting point and the offset point in the high-order word of the return value and the horizontal difference in the low-order word.

It limits the movement of the region according to constraints set by the `limitRect` and `slopRect` parameters:

- As long as the cursor is inside the `limitRect` rectangle, the region's outline follows it normally. If the mouse button is released while the cursor is within this rectangle, the return value reflects the simple distance that the cursor moved in each dimension.
- When the cursor moves outside the `limitRect` rectangle, the offset point stops at the edge of the `limitRect` rectangle. If the mouse button is released while the cursor is outside the `limitRect` rectangle but inside the `slopRect` rectangle, the return value reflects only the difference between the starting point and the offset point, regardless of how far outside of the `limitRect` rectangle the cursor may have moved. (Note that part of the region can fall outside the `limitRect` rectangle, but not the offset point.)
- When the cursor moves outside the `slopRect` rectangle, the region's outline disappears from the screen. The `DragGrayRgn` function continues to track the cursor, however, and if the cursor moves back into the `slopRect` rectangle, the outline reappears. If the mouse button is released while the cursor is outside the `slopRect` rectangle, both words of the return value are set to `0x8000`. In this case, the Window Manager does not move the window from its original location.
- To accommodate systems with multiple monitors, QuickDraw recognizes a port rectangle of `screenBits.bounds` as a special case and allows drawing on all parts of the desktop.

The region stops moving when the offset point reaches the edge of the `limitRect` rectangle. The cursor continues to move, but the region does not.

If the mouse button is released while the cursor is anywhere inside the `slopRect` rectangle, the Window Manager redraws the window in its new location, which is calculated from the value returned by `DragGrayRgn`.

Carbon Porting Notes

Can't be used for live dragging. If you are implementing your own window dragging, use `DragWindow` instead. If you need to override window positioning during a drag, register a Carbon event handler for `kEventWindowBoundsChanging`. Okay to use if you're dragging objects within a window.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

DragTheRgn

Tracks the mouse as the user drags the outline of a region. (Deprecated in Mac OS X v10.5. Use an overlay window or other custom drawing instead.)

```
long DragTheRgn (
   RgnHandle theRgn,
   Point startPt,
   const Rect *limitRect,
   const Rect *slopRect,
   short axis,
   DragGrayRgnUPP actionProc
);
```

Carbon Porting Notes

Can't be used for live dragging. If you are implementing your own window dragging, use `DragWindow` instead. If you need to override window positioning during a drag, register a Carbon event handler for `kEventWindowBoundsChanging`. Okay to use if you're dragging objects within a window.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

DragWindow

Moves a window on the screen when the user drags it by its drag region.

```
void DragWindow (
   WindowRef window,
   Point startPt,
   const Rect *boundsRect
);
```

Parameters

window
The window that is to be dragged.

startPt
On input, the location, in global coordinates, of the cursor at the time the user pressed the mouse button. Your application retrieves this point from the `where` field of the event structure.

boundsRect

On input, a pointer to a rectangle, given in global coordinates, that limits the region to which a window can be dragged. If the mouse button is released when the cursor is outside the limits of `boundsRect`, `DragWindow` returns without moving the window (or, if it was inactive, without making it the active window).

In CarbonLib and Mac OS X, this parameter can be `NULL` to indicate that there are no restrictions on window movement. This parameter is ignored by CarbonLib and Mac OS X v10.0 through v10.2; it is obeyed in Mac OX v10.3 and later.

Discussion
The `DragWindow` function moves the window around the screen, following the movement of the cursor until the user releases the mouse button. If the Command key was not pressed when the mouse button was pressed, `DragWindow` calls `SelectWindow` to make the window active before it drags the window. If the Command key was pressed when the mouse button was pressed, `DragWindow` moves the window without making it active.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
HideMenuBar

Declared In
`MacWindows.h`

DrawGrowIcon

Draws a grow icon in the window frame. (Deprecated in Mac OS X v10.5. There is no replacement function.)

```
void DrawGrowIcon (
    WindowRef window
);
```

Special Considerations

This function is not needed in Mac OS X. Theme-savvy windows include the grow box in the window frame.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

EnableScreenUpdates

Enables screen updates for changes to the current application's windows.

```
OSStatus EnableScreenUpdates (
   void
);
```

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

EndUpdate

Finishes updating a window.

```
void EndUpdate (
   WindowRef window
);
```

Parameters
window
The window for which updating is to be finished.

Discussion
The `EndUpdate` function restores the normal visible region of a window's graphics port. When you receive an update event for a window, you call `BeginUpdate` (page 1801), redraw the update region, and then call `EndUpdate`. Each call to `BeginUpdate` must be balanced by a subsequent call to `EndUpdate`.

Special Considerations

This function should not be used on composited windows. Modifying a composited window's update region does not affect the area of the window to be drawn. A composited window does not use its window update region to control drawing. Instead, a composited window determines what to draw by looking at the invalid regions of the views contained in the window.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
HideMenuBar
Simple DrawSprocket

Declared In
`MacWindows.h`

EndWindowProxyDrag

Disposes of the drag reference when the user completes the drag of a proxy icon.

```
OSStatus EndWindowProxyDrag (
   WindowRef window,
   DragRef theDrag
);
```

Parameters

window

The window whose proxy icon is being dragged.

theDrag

A value that refers to the current drag process. Pass in the value produced in the `outNewDrag` parameter of `BeginWindowProxyDrag`.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
Typically, if the proxy icon represents a type of object (currently, file system entities such as files, folders, and volumes) supported by the Window Manager, the Window Manager can handle all aspects of the drag process itself, and your application should call the function `TrackWindowProxyDrag` (page 1963). However, if the proxy icon represents a type of data that the Window Manager does not support, or if you want to implement custom dragging behavior, your application should call the function `TrackWindowProxyFromExistingDrag` (page 1964).

The `TrackWindowProxyFromExistingDrag` (page 1964) function accepts an existing drag reference and adds file data if the window contains a file proxy. If your application uses `TrackWindowProxyFromExistingDrag`, you then have the choice of using this function in conjunction with the functions `BeginWindowProxyDrag` (page 1802) and `EndWindowProxyDrag` or simply calling `TrackWindowProxyFromExistingDrag` and handling all aspects of creating and disposing of the drag yourself.

Specifically, your application can call `BeginWindowProxyDrag` to set up the drag image and drag reference. Your application must then track the drag, using `TrackWindowProxyFromExistingDrag`, and do any required moving of data and, finally, call `EndWindowProxyDrag` to dispose of the drag reference and its associated image data. The `EndWindowProxyDrag` function does not dispose of the region created for use by `BeginWindowProxyDrag`, however, so this remains the application's responsibility to dispose. The `EndWindowProxyDrag` function should not be used for types handled by the Window Manager unless you want to implement custom dragging behavior for those types.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

FindWindow

Maps the location of the cursor to a part of the screen or a region of a window when your application receives a mouse-down event.

```
WindowPartCode FindWindow (
    Point thePoint,
    WindowRef *window
);
```

Parameters

thePoint

The point, in global coordinates, where the mouse-down event occurred. Your application retrieves this information from the `where` field of the event structure.

window

A pointer to the window in which the mouse-down event occurred. `FindWindow` produces `NULL` if the mouse-down event occurred outside a window.

Return Value

The location of the cursor when the user pressed the mouse button; see "Window Part Code Constants" (page 2013).

Discussion

You typically call the function `FindWindow` whenever you receive a mouse-down event. The `FindWindow` function helps you dispatch the event by reporting whether the cursor was in the menu bar or in a window when the mouse button was pressed. If the cursor was in a window, the function will produce both a pointer to the window and a constant that identifies the region of the window in which the event occurred.

If you are using the Carbon event handlers to handle events, a faster way of getting the window and part that received a mouse-down event is to get the `kEventParamWindowRef` and `kEventParamWindowPartCode` parameters from the event.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

See Also

`HIWindowFindAtLocation` (page 1882)

Related Sample Code

HideMenuBar

Simple DrawSprocket

Declared In

`MacWindows.h`

FindWindowOfClass

Finds a window of a specific class at the specified point onscreen.

```
OSStatus FindWindowOfClass (
   const Point *where,
   WindowClass inWindowClass,
   WindowRef *outWindow,
   WindowPartCode *outWindowPart
);
```

Parameters

where

The point, in global coordinates, at which to search for a window.

inWindowClass

The class of window for which to search. Passing `kAllWindowClasses` returns any window found at `where`.

outWindow

On return, a pointer to the window, if it is of the specified class. If no window was found, this value is `NULL`. Note that you can pass `NULL` for this parameter.

outWindowPart

On return, the part code of the window part under the mouse. If no window was found, this value is `inDesk`. Note that you can pass `NULL` for this parameter.

Return Value

A result code. If no window of the specified class is found at the specified point, this function returns `errWindowNotFound`. For other possible return values, see "Window Manager Result Codes" (page 2051).

Discussion

This function is similar to `FindWindow` (page 1827), but lets you restrict the search to windows of a particular class.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

See Also

`HIWindowFindAtLocation` (page 1882)

Declared In

`MacWindows.h`

FrontNonFloatingWindow

Returns to the application the frontmost visible window that is not a floating window.

```
WindowRef FrontNonFloatingWindow (
   void
);
```

Return Value

The first visible window in the window list that is of a nonfloating class. See "Window Class Constants" (page 1988) for a description of window classes.

Discussion
Your application should call the `FrontNonFloatingWindow` function when you want to identify the frontmost visible window that is not a floating window. If you want to identify the frontmost visible window, whether floating or not, your application should call the function `FrontWindow`.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
CarbonSketch
QTCarbonShell

Declared In
`MacWindows.h`

FrontWindow

Identifies the frontmost visible window. (Deprecated in Mac OS X v10.5. Use `ActiveNonFloatingWindow` (page 1800), `FrontNonFloatingWindow` (page 1829), or `GetFrontWindowOfClass` (page 1834) instead.)

```
WindowRef FrontWindow (
   void
);
```

Return Value
The first visible window in the window list. If there are no visible windows, `FrontWindow` returns `NULL`.

Discussion
Most applications should call `ActiveNonFloatingWindow` (page 1800) or `FrontNonFloatingWindow` (page 1829) instead of `FrontWindow` because `ActiveNonFloatingWindow` and `FrontNonFloatingWindow` return the active and frontmost document window, respectively, skipping over other types of windows that may be in front of the active document, such as the menubar window, floating windows, help tags and toolbars.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Related Sample Code
HideMenuBar

Declared In
`MacWindows.h`

GetAvailableWindowAttributes

Returns the window attributes that are valid for a window class

```
WindowAttributes GetAvailableWindowAttributes (
   WindowClass inClass
);
```

Parameters

inClass

The window class to query.

Return Value

The window attributes that are valid for the window class specified by `inClass`. See "Window Attributes" (page 1998) for a list of possible attributes.

Discussion

Some window classes support different attributes on different platforms. For example, floating windows can have collapse boxes in Mac OS 9, but not in Mac OS X. The Window Manager returns an error if you attempt to create a window with attributes that aren't supported for the requested window class.

You can use this API to determine those attributes that are supported by the current platform and remove those attributes that are not supported by the current platform before calling `CreateNewWindow` (page 1815).

Availability

Available in Mac OS X v10.1 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

GetAvailableWindowPositioningBounds

Obtains the available window positioning bounds.

```
OSStatus GetAvailableWindowPositioningBounds (
   GDHandle inDevice,
   Rect *outAvailableRect
);
```

Parameters

inDevice

The screen for which the available window positioning bounds are to be obtained.

outAvailableRect

On return, a pointer to the available bounds for the device specified by `inDevice`.

Return Value

A result code. See "Window Manager Result Codes" (page 2051).

Discussion

The available window positioning bounds is that area on the screen inside which a window may be positioned without intersecting or overlapping the menu bar, Dock, or other UI provided by the operating system.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

GetAvailableWindowPositioningRegion

Obtains the available window positioning region.

```
OSStatus GetAvailableWindowPositioningRegion (
   GDHandle inDevice,
   RgnHandle ioRgn
);
```

Parameters

inDevice

The screen for which the available window positioning region is to be obtained.

ioRgn

On input, contains a preallocated `RgnHandle`. On return, the `RgnHandle` has been modified to contain the available region for the given screen.

Return Value

A result code. See "Window Manager Result Codes" (page 2051).

Discussion

The available window positioning region is that area on the screen inside which a window may be positioned without intersecting or overlapping the menu bar, Dock, or other UI provided by the operating system. This function differs from `GetAvailableWindowPositioningBounds` (page 1831) in that the bounds version removes the entire area that may theoretically be covered by the Dock, even if the Dock does not currently reach from edge to edge of the device on which it is positioned. The region version includes the area at the sides of the Dock that is not covered by the Dock in the available region.

Availability

Available in Mac OS X v10.2 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

GetDrawerCurrentEdge

Obtains the current window edge from which the drawer appears.

```
OptionBits GetDrawerCurrentEdge (
   WindowRef inDrawerWindow
);
```

Parameters

inDrawerWindow

The drawer window whose window edge is to be obtained.

Return Value

The current window edge. See "Window Edge Constants" (page 2041) for a list of possible return values.

Availability

Available in Mac OS X v10.2 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

GetDrawerOffsets

Obtains the positioning offsets of a drawer.

```
OSStatus GetDrawerOffsets (
   WindowRef inDrawerWindow,
   CGFloat *outLeadingOffset,
   CGFloat *outTrailingOffset
);
```

Parameters

inDrawerWindow

The drawer window whose positioning offsets are to be obtained.

outLeadingOffset

On exit, a pointer to the drawer's leading offset. Pass `NULL` if you don't need this information.

outTrailingOffset

On exit, a pointer to the drawer's trailing offset. Pass `NULL` if you don't need this information.

Return Value

A result code. See "Window Manager Result Codes" (page 2051).

Availability

Available in Mac OS X v10.2 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

GetDrawerParent

Obtains the parent window of a drawer.

```
WindowRef GetDrawerParent (
   WindowRef inDrawerWindow
);
```

Parameters

inDrawerWindow

The drawer window whose parent window is to be obtained.

Return Value

The window that is the parent of the drawer specified by `inDrawerWindow`.

Availability

Available in Mac OS X v10.2 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

GetDrawerPreferredEdge

Obtains the preferred opening edge for a drawer.

```
OptionBits GetDrawerPreferredEdge (
   WindowRef inDrawerWindow
);
```

Parameters

inDrawerWindow

The drawer window whose preferred opening edge is to be obtained.

Return Value

See "Window Edge Constants" (page 2041) for a list of possible values.

Discussion

Note that the preferred edge may not be the same as the current edge, due to window positioning. For example, the right edge may be the preferred edge, but if the window is placed such that the right edge is offscreen, the drawer will appear on the left edge instead.

Availability

Available in Mac OS X v10.2 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

GetDrawerState

Determines the current state of the drawer.

```
WindowDrawerState GetDrawerState (
   WindowRef inDrawerWindow
);
```

Parameters

inDrawerWindow

The drawer window whose state is to be determined.

Return Value

See "Drawer State Constants" (page 2040) for a list of possible values.

Availability

Available in Mac OS X v10.2 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

GetFrontWindowOfClass

Obtains the frontmost window of a given class.

```
WindowRef GetFrontWindowOfClass (
   WindowClass inWindowClass,
   Boolean mustBeVisible
);
```

Parameters

inWindowClass

The class of the window you want to obtain. If you pass `kAllWindowClasses`, the function returns the frontmost window in the window list.

mustBeVisible

If set to `true`, the function returns the frontmost visible window. If set to `false`, the function returns the frontmost window of the specified class, regardless of whether the window is visible.

Return Value
A reference to the frontmost window of the class specified by `inWindowClass`.

Carbon Porting Notes

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
HID Config Save
QTCarbonShell
QTMetaData

Declared In
`MacWindows.h`

GetGrayRgn

Returns a region that covers the desktop area of all active displays. (Deprecated in Mac OS X v10.5. To determine the area in which a window may be positioned, use `HIWindowGetAvailablePositioningBounds` (page 1885) or `HIWindowCopyAvailablePositioningShape` (page 1878).)

```
RgnHandle GetGrayRgn (
    void
);
```

Return Value
A handle to the current desktop region from the global variable `GrayRgn`.

Discussion
When your application calls `DragWindow` to let the user drag a window, it can use `GetGrayRgn` to set the limiting rectangle to the entire desktop area. The desktop region represents all available screen space, that is, the desktop area displayed by all monitors attached to the computer.

Special Considerations

Your application should not modify the desktop region.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Related Sample Code
HideMenuBar

Declared In
`MacWindows.h`

GetIndexedWindow

Obtains the window at the given index in the window group.

```
OSStatus GetIndexedWindow (
   WindowGroupRef inGroup,
   ItemCount inIndex,
   WindowGroupContentOptions inOptions,
   WindowRef *outWindow
);
```

Parameters

inGroup
The window group. For information on this data type,

inIndex
The index of the window. This parameter may range from `1` to the value returned by `CountWindowGroupContents`.

inOptions
Options for determining the number of windows. See "Window Group Content Options" (page 2032) for possible values.

outWindow
The window at the index specified by `inIndex` in the group specified by `inGroup`.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

GetNewCWindow

Creates a color window from a window resource. (Deprecated in Mac OS X v10.5. Use nib files and `CreateWindowFromNib` (page 1059) instead.)

```
WindowRef GetNewCWindow (
   short windowID,
   void *wStorage,
   WindowRef behind
);
```

Parameters

windowID

On input, the resource ID of the 'WIND' resource that defines the properties of the window.

wStorage

On input, a pointer to memory space for the window structure. If you specify a value of null for wStorage, the GetNewCWindow function allocates the window structure as a nonrelocatable object in the heap. You can reduce the chances of heap fragmentation by allocating the memory your application needs for window structures early in your initialization code. Whenever you need to create a window, you can allocate memory from your own block and pass a pointer to it in the wStorage parameter.

behind

On input, a pointer to the window that appears immediately in front of the new window on the desktop. To place a new window in front of all other windows on the desktop, specify a value of (WindowRef)-1L. When you place a window in front of all others, GetNewCWindow removes the highlighting from the previously active window, highlights the newly created window, and generates the appropriate activate events. Note that if you create an invisible window in front of all others on the desktop, the user sees no active window until you make the new window visible (or make another window active).

To place a new window behind all other windows, specify a value of null.

Return Value

A pointer to the newly created window structure.

Discussion

The GetNewCWindow function creates a new color window from the specified window resource and returns a pointer to the newly created window structure. You can use the returned window pointer to refer to this window in most Window Manager functions. If GetNewCWindow is unable to read the window or window definition function from the resource file, it returns null.

The GetNewCWindow function looks for a 'wctb' resource with the same resource ID as that of the 'WIND ' resource. If it finds one, it uses the window color information in the 'wctb' resource for coloring the window content area.

If the window's definition function (specified in the window resource) is not already in memory, GetNewCWindow reads it into memory and stores a handle to it in the window structure.

To create the window, GetNewCWindow retrieves the window characteristics from the window resource and then calls the NewCWindow function, passing the characteristics as parameters.

The GetNewCWindow function creates a window in a color graphics port. Your application typically sets up its own global variables reflecting the system setup during initialization by calling the Gestalt function.

Special Considerations

If you must get your window definition from a resource, use CreateWindowFromResource. Otherwise, use CreateWindowFromNib or CreateNewWindow.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

GetNewWindow

Creates a window from a window resource. (Deprecated in Mac OS X v10.5. Use nib files and `CreateWindowFromNib` (page 1059) instead.)

```
WindowRef GetNewWindow (
   short windowID,
   void *wStorage,
   WindowRef behind
);
```

Parameters

windowID

On input, the resource ID of the '`WIND`' resource that defines the properties of the window.

wStorage

On input, a pointer to memory space for the window structure. If you specify a value of `null` for `wStorage`, the `GetNewWindow` function allocates the window structure as a nonrelocatable object in the heap. You can reduce the chances of heap fragmentation by allocating the memory your application needs for window structures early in your initialization code. Whenever you need to create a window, you can allocate memory from your own block and pass a pointer to it in the `wStorage` parameter.

behind

On input, a pointer to the window that appears immediately in front of the new window on the desktop. To place a new window in front of all other windows on the desktop, specify a value of (`WindowRef`)-1l. When you place a window in front of all others, `GetNewWindow` removes the highlighting from the previously active window, highlights the newly created window, and generates the appropriate activate events. Note that if you create an invisible window in front of all others on the desktop, the user sees no active window until you make the new window visible (or make another window active). To place a new window behind all other windows, specify a value of `null`.

Return Value
A pointer to the newly created color window structure.

Discussion
The `GetNewWindow` function takes the same parameters as `GetNewCWindow` (page 1836) and returns a value of type `WindowRef`. The only difference is that it creates a monochrome graphics port, not a color graphics port, regardless of the presence of a corresponding '`wctb `' resource (it loads the resource but doesn't use it). The window structure and graphics port structure that describe monochrome and color graphics ports are the same size and can be used interchangeably in most Window Manager functions.

The `GetNewWindow` function creates a new window from the specified window resource and returns a pointer to the newly created window structure. You can use the returned window pointer to refer to this window in most Window Manager functions. If `GetNewWindow` is unable to read the window or window definition function from the resource file, it returns `null`.

If the window's definition function (specified in the window resource) is not already in memory, `GetNewWindow` reads it into memory and stores a handle to it in the window structure. It allocates space in the application heap for the structure and content regions of the window.

To create the window, `GetNewWindow` retrieves the window characteristics from the window resource and then calls the function `NewWindow`, passing the characteristics as parameters.

Special Considerations

If you must get your window definition from a resource, use `CreateWindowFromResource`. Otherwise, use `CreateWindowFromNib` or `CreateNewWindow`.

Version Notes

The `GetNewWindow` function was originally implemented prior to Color QuickDraw. In Mac OS 8, you should call the Color QuickDraw function `GetNewCWindow` instead of `GetNewWindow` to programmatically create a window, because Color QuickDraw is always available in Mac OS 8. Use of this function is not recommended with Mac OS 8 and later. `GetNewWindow` is described here only for completeness.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

GetNextWindow

Returns the next window in a window list.

```
WindowRef GetNextWindow (
    WindowRef window
);
```

Parameters

window

The window to start from.

Return Value

The next window in a window list.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

GetNextWindowOfClass

Obtains the next window in a given window group.

```
WindowRef GetNextWindowOfClass (
   WindowRef inWindow,
   WindowClass inWindowClass,
   Boolean mustBeVisible
);
```

Parameters

inWindow

The window at which to start.

inWindowClass

The class of window to obtain. If you pass `kAllWindowClasses`, the function returns the window directly behind the input window. If no windows exist behind the front window, the function returns `NULL`.

mustBeVisible

If set to `true`, this function returns the next visible window of the specified window class. If set to `false`, this function returns the next window of the specified window class, regardless of whether it is visible.

Return Value
A reference for the next window of the specified class after the window specified by `inWindow`.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
QTCarbonShell

Declared In
`MacWindows.h`

GetPreviousWindow

Returns the window above the specified window in the window list.

```
WindowRef GetPreviousWindow (
   WindowRef inWindow
);
```

Parameters

inWindow

The window at which to start.

Return Value
A reference for the previous window of the specified class.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

GetSheetWindowParent

Obtains the parent window of a sheet.

```
OSStatus GetSheetWindowParent (
   WindowRef inSheet,
   WindowRef *outParentWindow
);
```

Parameters

inSheet

The window sheet whose parent is to be obtained.

outParentWindow

A pointer to the reference for the parent of the window sheet specified by `inSheet`.

Return Value

A result code. See "Window Manager Result Codes" (page 2051).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

GetUserFocusWindow

Returns the current user focus window.

```
WindowRef GetUserFocusWindow (
    void
);
```

Return Value

The window receiving user focus.

Discussion

This function returns the window that receives menu commands and keyboard input as part of the standard event dispatching.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

GetWindowActivationScope

Obtains a window's activation scope.

```
OSStatus GetWindowActivationScope (
   WindowRef inWindow,
   WindowActivationScope *outScope
);
```

Parameters

inWindow

The window whose activation scope is to be obtained.

outScope

On return, a pointer to the window's activation scope.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
MacWindows.h

GetWindowAlpha

Returns the current alpha channel value for the window.

```
OSStatus GetWindowAlpha (
   WindowRef inWindow,
   CGFloat *outAlpha
);
```

Parameters

inWindow

The window for which the value of the alpha channel is to be obtained.

outAlpha

The current alpha value. This value can range from 0.0 (completely transparent) to 1.0 (opaque).

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
MacWindows.h

GetWindowAttributes

Obtains the attributes of a window.

```
OSStatus GetWindowAttributes (
   WindowRef window,
   WindowAttributes *outAttributes
);
```

Parameters

window

The window whose attributes you want to obtain.

outAttributes

On input, a pointer to an unsigned 32-bit value of type `WindowAttributes`. On return, the bits are set to the attributes of the specified window. See "Window Attributes" (page 1998) for a description of possible attributes.

Return Value

A result code. See "Window Manager Result Codes" (page 2051).

Discussion

Window attributes specify a window's features (such as whether the window has a close box) and logical attributes (such as whether the window receives update and activate events).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

GetWindowBounds

Obtains the size and position of the bounding rectangle of the specified window region.

```
OSStatus GetWindowBounds (
   WindowRef window,
   WindowRegionCode regionCode,
   Rect *globalBounds
);
```

Parameters

window

The window whose bounds you want to obtain.

regionCode

A constant identifying the window region whose bounds you want to obtain. See "Window Region Constants" (page 2021) for a list of possible values.

globalBounds

A pointer to a structure of type `Rect`. On return, the rectangle contains the dimensions and position, in global coordinates, of the window region specified in the `regionCode` parameter.

Return Value

A result code. See "Window Manager Result Codes" (page 2051).

Discussion

Use the `GetWindowBounds` function to obtain the bounding rectangle for the specified window region for the specified window.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

See Also
`HIWindowGetBounds` (page 1885)

Related Sample Code
QTCarbonShell

Declared In
`MacWindows.h`

GetWindowCancelButton

Returns the current Cancel button for a window.

```
OSStatus GetWindowCancelButton (
    WindowRef inWindow,
    ControlRef *outControl
);
```

Parameters

inWindow
The window whose Cancel button you want to obtain.

outControl
A pointer to a control. On output, the control is the Cancel button.

Return Value
A result code.

Discussion
You can use this function to determine which button or control is the specified Cancel button for a given window. This button would be considered to have been clicked if the user instead presses Command-period or the Escape key.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

GetWindowClass

Obtains the class of a window.

```
OSStatus GetWindowClass (
   WindowRef window,
   WindowClass *outClass
);
```

Parameters

window

The window whose class you want to obtain.

outClass

On input, a pointer to a value of type `WindowClass`. On return, this value identifies the class of the specified window. See "Window Class Constants" (page 1988) for a list of possible window classes. In Mac OS 8 and Mac OS 9, for windows not originally created by `CreateNewWindow` (page 1815), the class pointed to by the `outClass` parameter is always identified by the constant `kDocumentWindowClass`.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

GetWindowContentColor

Obtains the color to which a window's content region is redrawn.

```
OSStatus GetWindowContentColor (
   WindowRef window,
   RGBColor *color
);
```

Parameters

window

The window whose content color is being retrieved.

color

On input, a pointer to an `RGBColor` structure. On return, the structure contains the content color for the specified window.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
The `GetWindowContentColor` function obtains the color to which the window's content region is redrawn.

See also the function `SetWindowContentColor` (page 1938).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
MacWindows.h

GetWindowContentPattern

Obtains the pattern to which a window's content region is redrawn.

```
OSStatus GetWindowContentPattern (
   WindowRef window,
   PixPatHandle outPixPat
);
```

Parameters

window
The window whose content pattern is being retrieved.

outPixPat
On input, a handle to a structure of type PixPat. On return, the structure contains a copy of the content pattern data for the specified window, which your application is responsible for disposing.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
The GetWindowContentPattern function obtains the pattern to which the window's content region is redrawn.

See also the function SetWindowContentPattern (page 1939).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
MacWindows.h

GetWindowDefaultButton

Returns the current default button for a window.

```
OSStatus GetWindowDefaultButton (
    WindowRef inWindow,
    ControlRef *outControl
);
```

Parameters

inWindow
The window whose default button you want to obtain.

outControl
A pointer to a control. On output, the control is the default button.

Return Value
A result code.

Discussion
You can use this function to determine which button or control is the default for a given window. This button would be considered to have been clicked if the user instead presses the Return or Enter keys on the keyboard.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

GetWindowDockTileMenu

Returns the menu to be displayed by a window's dock tile.

```
MenuRef GetWindowDockTileMenu (
   WindowRef inWindow
);
```

Parameters

`inWindow`
The window whose menu is to be obtained.

Return Value
The menu reference for the window specified by `inWindow`. See the Menu Manager documentation for a description of the `MenuRef` data type.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

GetWindowFeatures

Obtains the features that a window supports.

```
OSStatus GetWindowFeatures (
   WindowRef window,
   UInt32 *outFeatures
);
```

Parameters

`window`
A pointer to the window to be examined.

`outFeatures`
On input, a pointer to an unsigned 32-bit value. On return, the bits of the value specify the features the window supports; see "Window Feature Bits" (page 2011).

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
The `GetWindowFeatures` function produces a window definition function's features in response to a `kWindowMsgGetFeatures` message.

Instead of calling this function, most applications should call `GetWindowAttributes` (page 1842) to check for specific attributes, such as `kWindowCollapseBoxAttribute` and `kWindowResizableAttribute`.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

GetWindowFromPort

Gets a window reference from a `CGrafPtr` data type.

```
WindowRef GetWindowFromPort (
   CGrafPtr port
);
```

Parameters

port
The port to query.

Return Value
The window reference obtained from the port specified by `port`, or `NULL` if the `port` parameter is not actually attached to a window.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

GetWindowGreatestAreaDevice

Returns the graphics device with the greatest area of intersection with a specified window region.

```
OSStatus GetWindowGreatestAreaDevice (
   WindowRef inWindow,
   WindowRegionCode inRegion,
   GDHandle *outGreatestDevice,
   Rect *outGreatestDeviceRect
);
```

Parameters

inWindow
The window to compare against.

inRegion

The window region to compare against. See "Window Region Constants" (page 2021) for a list of possible values.

outGreatestDevice

On return, the graphics device with the greatest intersection. May be `NULL`.

outGreatestDeviceRect

On return, the bounds of the graphics device with the greatest intersection. May be `NULL`. If the device with the greatest intersection also contains the menu bar, the device rect will exclude the menu bar area.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

GetWindowGroup

Obtains the window group associated with a window.

```
WindowGroupRef GetWindowGroup (
   WindowRef inWindow
);
```

Parameters

inWindow

The window whose window group is to be obtained.

Return Value
The window group reference for the window specified by `inWindow`. For information on this data type, see `WindowGroupRef` (page 1986).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

GetWindowGroupAttributes

Obtains the attributes of a window group.

```
OSStatus GetWindowGroupAttributes (
   WindowGroupRef inGroup,
   WindowGroupAttributes *outAttributes
);
```

Parameters

inGroup

The window group whose attributes are to be changed. For information on this data type,

outAttributes

On return, the attributes of the group. See "Window Group Attributes" (page 2031) for a list of possible values.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

GetWindowGroupContents

Obtains the contents of a window group.

```
OSStatus GetWindowGroupContents (
   WindowGroupRef inGroup,
   WindowGroupContentOptions inOptions,
   ItemCount inAllowedItems,
   ItemCount *outNumItems,
   void **outItems
);
```

Parameters

inGroup

The window group whose contents you want to obtain. For information on this data type, see `WindowGroupRef` (page 1986).

inOptions

Options for determining how to count the group members. See "Window Group Content Options" (page 2032) for a list of possible values.

inAllowedItems

The number of items that will fit in `outItems`.

outNumItems

On return, the number of items in the group.

outItems

On entry, this parameter must be a pointer to a pre-allocated buffer in which the window group contents (either window references or window group references) are to be placed. On return, the buffer pointed to by this parameter contains the requested window references or window group references.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

GetWindowGroupLevel

Obtains the level of the group in the window class hierarchy.

```
OSStatus GetWindowGroupLevel (
   WindowGroupRef inGroup,
   SInt32 *outLevel
);
```

Parameters

inGroup
The window group. For information on this data type,

outLevel
On exit, the window level of the windows in this group.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
The window group's level is only used to set the level of its windows if the window group is a child of the root group. If there is another group in the group hierarchy between this group and the root group, this group's level is ignored.

See the Core Graphics frameworks header `CGWindowLevel.h` for a listing of window levels.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

GetWindowGroupLevelOfType

Obtains the Core Graphics window level of a window group.

```
OSStatus GetWindowGroupLevelOfType (
   WindowGroupRef inGroup,
   UInt32 inLevelType,
   CGWindowLevel *outLevel
);
```

Parameters

inGroup
The window group whose Core Graphics window level is to be obtained.

inLevelType

The level type to obtain. Specify `kWindowGroupLevelActive`, `kWindowGroupLevelInactive`, or `kWindowGroupLevelPromoted`. For details, see "Window Group Level Constants" (page 2048).

outLevel

On output, the Core Graphics window level for the windows in this group.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
In Mac OS X v10.4 and later, multiple Core Graphics window levels may be associated with a window group: one level for when the application is active and another for when the application is inactive. The Window Manager automatically switches each group's Core Graphics window level as the application becomes active or inactive. Use `GetWindowGroupLevelOfType` to get each Core Graphics window level associated with a window group, including the promoted window level that is actually in use for windows in the group when the application is active.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

GetWindowGroupOfClass

Obtains the window group corresponding to a given window class.

```
WindowGroupRef GetWindowGroupOfClass (
   WindowClass windowClass
);
```

Parameters

windowClass

The window class to query.

Return Value
For information on this data type, see `WindowGroupRef` (page 1986).

Discussion
Each window class has an associated pre-defined window group. This function returns the window group reference for the window group that is associated with `windowClass`. Note that all windows in a group do not have to be of the same window class.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

GetWindowGroupOwner

Obtains the window that owns a window group. (if any)

```
WindowRef GetWindowGroupOwner (
   WindowGroupRef inGroup
);
```

Parameters

inGroup

The window group to query. For information on this data type,

Return Value

The window reference for the window that owns the group specified by `inGroup`.

Discussion

You call `SetWindowGroupOwner` (page 1944) to associate a window group with a particular window.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

GetWindowGroupParent

Obtains the parent group of a window group.

```
WindowGroupRef GetWindowGroupParent (
   WindowGroupRef inGroup
);
```

Parameters

inGroup

The window group whose parent is to be obtained.

Return Value

The parent of the window group specified by `inGroup`.

Discussion

You can nest window groups within each other.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

GetWindowGroupRetainCount

Determines the current reference count for a window group.

```
ItemCount GetWindowGroupRetainCount (
   WindowGroupRef inGroup
);
```

Parameters

inGroup

The window group for which the current reference count is to be obtained. For information on this data type, see `WindowGroupRef` (page 1986).

Availability
Available in Mac OS X v10.1 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

GetWindowGroupSibling

Obtains the next or previous group of a window group.

```
WindowGroupRef GetWindowGroupSibling (
   WindowGroupRef inGroup,
   Boolean inNextGroup
);
```

Parameters

inGroup

The window group for which the next or previous group is to be obtained. For information on this data type, see `WindowGroupRef` (page 1986).

inNextGroup

Pass `true` to obtain the next sibling; `false` to obtain the previous sibling.

Return Value
The next or previous group. For information on this data type, see `WindowGroupRef` (page 1986).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

GetWindowIdealUserState

Obtains the size and position of a window in its user state.

```
OSStatus GetWindowIdealUserState (
    WindowRef inWindow,
    Rect *outUserState
);
```

Parameters

inWindow

The window for which you want to obtain the user state.

outUserState

On input, a pointer to a structure of type `Rect`. On return, this rectangle specifies the current size and position of the window's user state, in global coordinates.

Return Value

A result code. See "Window Manager Result Codes" (page 2051).

Discussion

Because the window definition function relies upon the `WStateData` structure, it is unaware of the ideal standard state, and this causes the user state data that it stores in the `WStateData` structure to be unreliable. While the Window Manager is reliably aware of the window's zoom state, it cannot record the current user state in the `WStateData` structure, because the window definition function can overwrite that data. Therefore, the function `ZoomWindowIdeal` (page 1972) maintains the window's user state independently of the `WStateData` structure. The `GetWindowIdealUserState` function gives your application access to the user state data maintained by `ZoomWindowIdeal`. However, your application should not typically need to use this function; it is supplied for completeness.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

See Also

`HIWindowGetIdealUserState` (page 1887)

Declared In

`MacWindows.h`

GetWindowIndex

Obtains the index number of a specified window in a group.

```
OSStatus GetWindowIndex (
   WindowRef inWindow,
   WindowGroupRef inStartGroup,
   WindowGroupContentOptions inOptions,
   ItemCount *outIndex
);
```

Parameters

inWindow

The window whose window group index number is to be obtained.

inStartGroup

The window group to query.

inOptions

Specifies how to enumerate the specified window; `kWindowGroupContentsReturnWindows` is implied and does not need to be specified explicitly.

outIndex

A pointer to a variable that, on return, contains the window's z-order index. The frontmost window in a window group has an index of 1. Window indexes increase as the window gets lower in z-order (that is, visually further from the top of the window list and closer to the desktop.)

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

GetWindowKind

Returns a window's window kind.

```
short GetWindowKind (
   WindowRef window
);
```

Parameters

window

The window whose window kind is to be returned.

Return Value
An integer representing the window kind; see "Window Kinds" (page 2029).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

GetWindowList

Obtains the first window in a window list.

```
WindowRef GetWindowList (
   void
);
```

Return Value
A window reference for the first window in the list.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
MacWindows.h

GetWindowModality

Obtains the modality of a window.

```
OSStatus GetWindowModality (
   WindowRef inWindow,
   WindowModality *outModalKind,
   WindowRef *outUnavailableWindow
);
```

Parameters

inWindow
The window whose modality is to be obtained.

outModalKind
On return, contains the modality of the window.

outUnavailableWindow
On return, if the window is window-modal, contains the target window of the specified window's modality.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
MacWindows.h

GetWindowOwnerCount

Obtains the number of existing references to a window. (Deprecated in Mac OS X v10.5. Use `CFGetRetainCount` instead.)

```
OSStatus GetWindowOwnerCount (
   WindowRef window,
   ItemCount *outCount
);
```

Parameters

window
The window whose reference (owner) count is to be determined.

outCount
A pointer to a value that, on return, contains the current number of references to the window.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
All windows are created with a reference count (owner count) of one. The function `CloneWindow` (page 1808) increments the number of references to a window, and the earlier function `DisposeWindow` decrements the number of references. When the reference count reaches zero, `DisposeWindow` disposes of the window.

In Mac OS X v10.2 and later, you can also call `CFGetRetainCount` to get the number of existing references to a window.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

GetWindowPic

Returns a handle to a window's picture. (Deprecated in Mac OS X v10.5. Use an HIImageView object to draw a window's content and ask the view for its image instead.)

```
PicHandle GetWindowPic (
   WindowRef window
);
```

Parameters
window
The window whose picture handle is to be returned.

Return Value
A handle to the picture to be drawn in a specified window's content region. The handle must have been stored previously with the function `SetWindowPic` (page 1947).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

GetWindowPort

Gets the window's color graphics port.

```
CGrafPtr GetWindowPort (
   WindowRef window
);
```

Parameters
window
The window whose color graphics port is to be obtained.

Return Value
A pointer to the window's color graphics port. See the QuickDraw Manager documentation for a description of the `CGrafPtr` data type.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
CarbonSketch
QTCarbonShell

Declared In
`MacWindows.h`

GetWindowPortBounds

Obtains the bounds of the window port.

```
Rect * GetWindowPortBounds (
   WindowRef window,
   Rect *bounds
);
```

Parameters

window
The window whose port bounds you want.

bounds
A pointer to a `Rect` structure. On return, the `Rect` structure contains the bounds of the window port.

Return Value
The same value (pointer to a `Rect` structure) that was passed to `GetWindowPortBounds` in the `bounds` parameter.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
CarbonSketch
HID Explorer

Declared In
`MacWindows.h`

GetWindowProperty

Obtains a piece of data that is associated with a window.

```
OSStatus GetWindowProperty (
   WindowRef window,
   PropertyCreator propertyCreator,
   PropertyTag propertyTag,
   ByteCount bufferSize,
   ByteCount *actualSize,
   void *propertyBuffer
);
```

Parameters

window

The window to be examined for associated data.

propertyCreator

The creator code (typically, the application's signature) of the associated data to be obtained.

propertyTag

The application-defined code identifying the associated data to be obtained.

bufferSize

The size of the associated data to be obtained. If the size of the data is unknown, use the function `GetWindowPropertySize` (page 1861) to get the data's size. If the size specified does not match the actual size of the property, `GetWindowProperty` only retrieves data up to the size specified or up to the actual size of the property, whichever is smaller, and an error is returned.

actualSize

On input, a pointer to a value. On return, the value specifies the actual size of the obtained data. You may pass `NULL` for the `actualSize` parameter if you are not interested in this information.

propertyBuffer

On input, a pointer to a buffer. On return, this buffer contains a copy of the data that is associated with the specified window.

Return Value

A result code. See "Window Manager Result Codes" (page 2051).

Discussion

The data retrieved by the `GetWindowProperty` function must have been previously associated with the window with the function `SetWindowProperty` (page 1948).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

QTMetaData

Declared In

`MacWindows.h`

GetWindowPropertyAttributes

Obtains the attributes of a window property.

```
OSStatus GetWindowPropertyAttributes (
   WindowRef window,
   OSType propertyCreator,
   OSType propertyTag,
   OptionBits *attributes
);
```

Parameters

window

The window having a property whose attributes are to be obtained.

propertyCreator

The property creator.

propertyTag

The property tag.

attributes

On return, the property's attributes. Currently, the only valid property is `kWindowPropertyPersistent`. For a description of this property, see "Window Property Persistent Constant" (page 2024).

Return Value

A result code. See "Window Manager Result Codes" (page 2051).

Carbon Porting Notes

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

GetWindowPropertySize

Obtains the size of a piece of data that is associated with a window.

```
OSStatus GetWindowPropertySize (
   WindowRef window,
   PropertyCreator creator,
   PropertyTag tag,
   ByteCount *size
);
```

Parameters

window

The window to be examined for associated data.

creator

The creator code (typically, the application's signature) of the associated data whose size is to be obtained.

tag

The application-defined code identifying the associated data whose size is to be obtained.

size

A pointer to a value that, on return, specifies the size of the associated data.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
If you want to retrieve a piece of associated data with the `GetWindowProperty` (page 1859) function, you typically need to use the `GetWindowPropertySize` function to determine the size of the data beforehand.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

GetWindowProxyAlias

Obtains an alias for the file that is associated with a window.

```
OSStatus GetWindowProxyAlias (
   WindowRef window,
   AliasHandle *alias
);
```

Parameters

window
The window for which you want to determine the associated file.

alias
On input, a pointer to a value of type `AliasHandle`. On return, the `AliasRecord` structure referenced by the alias handle contains a copy of the alias data for the file associated with the specified window. Your application must dispose of this handle.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
Your application can call the `GetWindowProxyAlias` function to retrieve alias data for the file associated with a window.

See also the function `SetWindowProxyAlias` (page 1949).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

GetWindowProxyFSSpec

Obtains a file system specification structure for the file that is associated with a window. (Deprecated in Mac OS X v10.5. Use `HIWindowGetProxyFSRef` (page 1888) instead.)

```
OSStatus GetWindowProxyFSSpec (
   WindowRef window,
   FSSpec *outFile
);
```

Parameters

window

A pointer to the window for which you wish to determine the associated file.

outFile

On input, a pointer to an `FSSpec` structure. On return, this structure contains a copy of the file system specification data for the file associated with the specified window.

Return Value

A result code. See "Window Manager Result Codes" (page 2051).

Discussion

You can use the `GetWindowProxyFSSpec` function to obtain identifying information about a proxy file: its volume reference number, directory ID, and file name.

See also the function `SetWindowProxyFSSpec`.

Special Considerations

The use of file specification structures is no longer recommended.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Related Sample Code

QTCarbonShell

Declared In

`MacWindows.h`

GetWindowProxyIcon

Obtains a window's proxy icon.

```
OSStatus GetWindowProxyIcon (
   WindowRef window,
   IconRef *outIcon
);
```

Parameters

window

The window for which you want to obtain the proxy icon.

outIcon

A pointer to a variable of type `IconRef` that, on return, identifies the window's proxy icon. Your application must not dispose of this icon.

Return Value

A result code. If no proxy icon is found, this function returns `errWindowDoesNotHaveProxy`. For other possible return values, see "Window Manager Result Codes" (page 2051).

Discussion
There are several different ways to associate a proxy icon with a window:

- If you use the function `SetWindowProxyIcon` (page 1951), `GetWindowProxyIcon` returns the proxy icon you set.
- If you use the function `SetWindowProxyCreatorAndType` (page 1950), that function uses Icon Services to find and set the proxy icon corresponding to the creator and type. `GetWindowProxyIcon` returns that icon.
- If you use `SetWindowProxyAlias` (page 1949), `SetWindowProxyFSSpec` (page 1950), or `HIWindowSetProxyFSRef` (page 1894), then `GetWindowProxyIcon` attempts to resolve the alias (if available) and returns the icon associated with the specified file.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

GetWindowRegion

Obtains a handle to a specific window region. (Deprecated in Mac OS X v10.5.)

```
OSStatus GetWindowRegion (
   WindowRef window,
   WindowRegionCode inRegionCode,
   RgnHandle ioWinRgn
);
```

Parameters

window
The window for which a window region handle is to be obtained.

inRegionCode
A constant representing the window region whose handle you want to obtain; see "Window Region Constants" (page 2021) for a list of possible values.

ioWinRgn
On input, a handle to a region created by your application. On return, the handle is set to the specified window region.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
The `GetWindowRegion` function produces a handle to a window definition function's window region in response to a `kWindowMsgGetRegion` message. The visibility of the window is unimportant for `GetWindowRegion` to work correctly.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

See Also
HIWindowCopyShape (page 1879)

Declared In
MacWindows.h

GetWindowResizeLimits

Returns the minimum and maximum content sizes for a window.

```
OSStatus GetWindowResizeLimits (
   WindowRef inWindow,
   HISize *outMinLimits,
   HISize *outMaxLimits
);
```

Parameters

inWindow
The window whose minimum and maximum content sizes are to be obtained.

outMinLimits
On return, the window's minimum content size. Pass NULL if you don't want this information. For information on the HISize data type, see HIGeometry.h.

outMaxLimits
On return, the window's maximum content size. Pass NULL if you don't want this information. For information on the HISize data type, see HIGeometry.h.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
MacWindows.h

GetWindowRetainCount

Returns the retain count of a window. (Deprecated in Mac OS X v10.5. Use CFGetRetainCount instead.)

```
ItemCount GetWindowRetainCount (
   WindowRef inWindow
);
```

Parameters

inWindow
The window whose retain count to retrieve.

Discussion
This API is equivalent to GetWindowOwnerCount (page 1857). For consistency with Core Foundation and Carbon Events, it is preferred over GetWindowOwnerCount. Both APIs will continue to be supported.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

GetWindowStandardState

Obtains a window's standard zoom rectangle.

```
Rect * GetWindowStandardState (
   WindowRef window,
   Rect *rect
);
```

Parameters

window
The window whose standard zoom rectangle is to be obtained.

rect
On input, a pointer to a `Rect` structure. On return, the `Rect` structure contains the window's standard zoom rectangle, in global coordinates. A window's standard zoom rectangle is the window content bounds when the window is zoomed out to its greatest extent.

Return Value
The same value (pointer to a `Rect` structure) that was passed to `GetWindowStandardState` in the `rect` parameter.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

GetWindowStructurePort

Obtains a graphics port that is used when drawing a window's structure.

```
CGrafPtr GetWindowStructurePort (
   WindowRef inWindow
);
```

Parameters

inWindow
The window to query.

Return Value
The CGrafPtr that is used when drawing the window's structure (window frame).

Availability
Available in Mac OS X v10.1 and later.

Not available to 64-bit applications.

Declared In
MacWindows.h

GetWindowStructureWidths

Obtains the width of the structure region on each edge of a window.

```
OSStatus GetWindowStructureWidths (
   WindowRef inWindow,
   Rect *outRect
);
```

Parameters

inWindow
The window to query.

outRect
On return, the `Rect` structure is filled in with the widths of the structure.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
MacWindows.h

GetWindowToolbar

Obtains the toolbar associated with a window.

```
OSStatus GetWindowToolbar (
   WindowRef inWindow,
   HIToolbarRef *outToolbar
);
```

Parameters

inWindow
The window whose toolbar is to be obtained.

outToolbar
On return, the toolbar that is attached to the window, or `NULL` if the window has no toolbar.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
MacWindows.h

GetWindowUserState

Returns a window's user zoom rectangle.

```
Rect * GetWindowUserState (
   WindowRef window,
   Rect *rect
);
```

Parameters

window
The window whose user zoom rectangle is to be returned.

rect
On input, a pointer to a `Rect` structure. On return, the `Rect` structure contains the window's user zoom rectangle, in global coordinates. A window's user zoom rectangle is the window content bounds when the window is zoomed back in.

Return Value
The same value (pointer to a `Rect` structure) that was passed to `GetWindowUserState` in the `rect` parameter.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
MacWindows.h

GetWindowWidgetHilite

Obtains the window part code of the window widget that is currently highlighted.

```
OSStatus GetWindowWidgetHilite (
   WindowRef inWindow,
   WindowDefPartCode *outHilite
);
```

Parameters

inWindow
The window to query.

outHilite
The highlight.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
MacWindows.h

GetWRefCon

Returns the reference constant from a window.

```
SRefCon GetWRefCon (
   WindowRef window
);
```

Parameters

window
The window whose reference constant is to be returned.

Return Value
The long integer data stored in the `refCon` field of the window structure specified in the `window` parameter.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
CarbonSketch
QTCarbonShell

Declared In
MacWindows.h

GetWTitle

Retrieves the title of a window as a Pascal string. (Deprecated in Mac OS X v10.5. Use `CopyWindowTitleAsCFString` (page 1813) instead.)

```
void GetWTitle (
   WindowRef window,
   Str255 title
);
```

Parameters

window
On input, a pointer to the window structure.

title
A Pascal string. On output, the string contains the window title.

Discussion
The `GetWTitle` function returns the title of the window in the `title` parameter.

When you need to retrieve a window's title, you should always use `GetWTitle` instead of reading the title from a window structure.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
MacWindows.h

GetWVariant

Returns a window's variation code. (Deprecated in Mac OS X v10.5. Use `GetWindowAttributes` (page 1842) to determine aspects of a window's appearance or behavior.)

```
short GetWVariant (
    WindowRef window
);
```

Parameters

window
On input, a pointer to the window structure.

Return Value
A short integer that specifies the variation code of the specified window. Depending on the window definition function, the result of `GetWVariant` can represent one of the standard variation codes or a variation code defined by your own window definition function.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
MacWindows.h

GrowWindow

Allows the user to change the size of a window. (Deprecated in Mac OS X v10.5. Use `ResizeWindow` (page 1925) instead.)

```
long GrowWindow (
   WindowRef window,
   Point startPt,
   const Rect *bBox
);
```

Parameters

window
On input, a pointer to the window structure of the window to drag.

startPt
On input, the location of the cursor at the time the mouse button was first pressed, in global coordinates. Your application retrieves this point from the `where` field of the event structure.

bBox

On input, a pointer to a rectangle structure that specifies the limits on the vertical and horizontal measurements of the port rectangle, in pixels.

Although the `bBox` parameter gives the address of a structure which is in the form of the `Rect` data type, the four numbers in the structure represent lengths, not screen coordinates. The `top`, `left`, `bottom`, and `right` fields of the `bBox` parameter specify the minimum vertical measurement (`top`), the minimum horizontal measurement (`left`), the maximum vertical measurement (`bottom`), and the maximum horizontal measurement (`right`).

The minimum measurements must be large enough to allow a manageable rectangle 64 pixels on a side is typical. Because the user cannot ordinarily move the cursor off the screen, you can safely set the upper bounds to the largest possible length (65,535 pixels) when you're using `GrowWindow` to follow cursor movements.

Return Value
A long integer that specifies the new dimensions, in pixels, of the resulting window: the height in the high-order word of the returned long-integer value and the width in the low-order word. A return value of 0 means that the new size is the same as the size of the current port rectangle.

Discussion
The `GrowWindow` function displays an outline (grow image) of the window as the user moves the cursor to make the window larger or smaller; it handles all user interaction until the user releases the mouse button. After calling `GrowWindow`, you call the function `SizeWindow` to change the size of the window.

The `GrowWindow` function moves a dotted-line image of the window's right and lower edges around the screen, following the movements of the cursor until the mouse button is released. You can use the functions `HiWord` and `LoWord`, described in the *Mathematical and Logical Utilities Reference*, to retrieve only the high-order and low-order words, respectively.

Special Considerations

In non-Carbon implementations of `GrowWindow` on Mac OS 8 and 9, the maximum size that the specified window is allowed to grow to is actually one less than the values specified in the `bBox` parameter. For example, if you pass the values 500 in the `bBox.bottom` field and 600 in the `bBox.right` field, the maximum height and width of the window would actually be 499 and 599, respectively.

However, in Carbon, the maximum height and width allowed for the specified window is equal to the values passed in the `bBox.bottom` and `bBox.right` fields, respectively.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

HideFloatingWindows

Hides an application's floating windows.

```
OSStatus HideFloatingWindows (
   void
);
```

Return Value
A result code. For details, see "Window Manager Result Codes" (page 2051).

Discussion
When an application receives a suspend event, its floating windows are hidden automatically. When the application receives a resume event, the floating windows are made visible automatically. Call this function if you want to hide your floating windows manually.

See also the function `ShowFloatingWindows` (page 1956).

Special Considerations

The `HideFloatingWindows` function operates only upon windows created with the `kFloatingWindowClass` constant; see "Window Class Constants" (page 1988) for more details on this constant.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

HideSheetWindow

Hides a sheet window using appropriate visual effects.

```
OSStatus HideSheetWindow (
   WindowRef inSheet
);
```

Parameters

inSheet
The window sheet that is to be hidden.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

HideWindow

Makes a window invisible.

```
void HideWindow (
   WindowRef window
);
```

Parameters

window

The window that is to be made invisible.

Discussion

The `HideWindow` function make a visible window invisible. If you hide the frontmost window, `HideWindow` removes the highlighting, brings the window behind it to the front, highlights the new frontmost window, and generates the appropriate activate events.

To reverse the actions of `HideWindow`, you must call both `ShowWindow` (page 1959), to make the window visible, and `SelectWindow` (page 1929), to select it.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

HiliteWindow

Sets a window's highlighting status.

```
void HiliteWindow (
   WindowRef window,
   Boolean fHilite
);
```

Parameters

window

On input, a pointer to the window structure.

fHilite

On input, a Boolean value that specifies the highlighting status: `true` highlights a window; `false` removes highlighting.

Discussion

The `HiliteWindow` function sets a window's highlighting status to the specified state. If the value of the `fHilite` parameter is `true`, `HiliteWindow` highlights the specified window; if the specified window is already highlighted, the function has no effect. If the value of `fHilite` is `false`, `HiliteWindow` removes highlighting from the specified window; if the window is not already highlighted, the function has no effect.

Your application doesn't normally need to call `HiliteWindow`. To make a window active, you can call `SelectWindow` or `ActivateWindow`, which handle highlighting for you.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

HiliteWindowFrameForDrag

Sets the highlight state of the window's structure region to reflect the window's validity as a drag-and-drop destination.

```
OSStatus HiliteWindowFrameForDrag (
   WindowRef window,
   Boolean hilited
);
```

Parameters

window

The window for which you want to set the highlight state.

hilited

Pass `true` if the window's frame should be highlighted otherwise, pass `false`.

Return Value

A result code. See "Window Manager Result Codes" (page 2051).

Discussion

Applications typically call the Drag Manager functions `ShowDragHilite` and `HideDragHilite` to indicate that a window is a valid drag-and-drop destination. If your application does not do this—that is, if your application implements any type of custom drag highlighting, such as highlighting more than one area of a window at a time—it must call the `HiliteWindowFrameForDrag` function.

The `HiliteWindowFrameForDrag` function highlights a window's proxy icon when the user drags content inside the window that is a valid content type for that destination. The default behavior of system-defined windows is to highlight the proxy icon along with the window's content area when the window is a valid drag-and-drop destination. If you call the Drag Manager functions `ShowDragHilite` and `HideDragHilite`, you don't need to use `HiliteWindowFrameForDrag`.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

HIWindowChangeAttributes

Changes the attributes of a window.

```
OSStatus HIWindowChangeAttributes (
   WindowRef inWindow,
   const int *inAttrToSet,
   const int *inAttrToClear
);
```

Parameters

inWindow

The window to change.

inAttrToSet

A zero-terminated array of window attribute constants. Possible values are described in "Window Attribute Identifiers" (page 1992). Each array entry specifies an attribute of the window to set. You may pass NULL if you do not wish to set any attributes.

inAttrToClear

A zero-terminated array of window attribute constants. Possible values are described in "Window Attribute Identifiers" (page 1992). Each array entry specifies an attribute of the window to clear. You may pass NULL if you do not wish to clear any attributes.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
This function takes two arrays of window attribute constants, as described in "Window Attribute Identifiers" (page 1992). The first array specifies the attributes to set, and the second specifies the attributes to clear. For example, you might call this function as follows:

```
int setAttr[] = { kHIWindowBitCloseBox, kHIWindowBitZoomBox, 0 };
int clearAttr[] = { kHIWindowBitNoTitleBar, 0 };
HIWindowChangeAttributes (window, setAttr, clearAttr);
```

Special Considerations

In Mac OS X v10.4 or earlier, you can use the function ChangeWindowAttributes (page 1805) to achieve similar results.

Availability
Available in Mac OS X v10.5 and later.
Not available to 64-bit applications.

Declared In
MacWindows.h

HIWindowChangeAvailability

Changes the availability of a window during Exposé or in Spaces.

```
OSStatus HIWindowChangeAvailability (
   WindowRef inWindow,
   HIWindowAvailability inSetAvailability,
   HIWindowAvailability inClearAvailability
);
```

Parameters

inWindow

The window whose availability is to be changed.

inSetAvailability

The availability bits to set. For details, see "Window Availability Constants" (page 2046).

inClearAvailability

The availability bits to clear. For details, see "Window Availability Constants" (page 2046).

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
This function overrides the default behavior of the Window Manager in determining whether a window is visible during Exposé or in all Spaces workspaces. Most applications should not override the default behavior; these options should only be used in special cases. For example, accessibility assistance applications may need to create windows that are visible in all workspaces.

By default, newly created windows of class `kDocumentWindowClass` are given an availability of `0` (meaning that they are available during Exposé), and windows from all other window classes are given an availability of `kHIWindowExposeHidden`.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

HIWindowChangeClass

Changes the appearance and behavior of a window.

```
OSStatus HIWindowChangeClass (
   WindowRef inWindow,
   WindowClass inWindowClass
);
```

Parameters

window
The window whose class you want to change.

inClass
The new class that is to be applied to the window. See "Window Class Constants" (page 1988) for a list of possible window classes.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
This function changes the class of a window. Unlike `SetWindowClass` (page 1938), `HIWindowChangeClass` effectively changes both the appearance and behavior of the window.

This function can convert a window between `kDocumentWindowClass`, `kFloatingWindowClass`, `kUtilityWindowClass`, and `kMovableModalWindowClass` only. It cannot, for example, change a document window into a plain window.

The attributes of the window are adjusted to contain only those that are allowed for the new class. It is the caller's responsibility to adjust them further, as necessary, after `HIWindowChangeClass` returns.

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

HIWindowChangeFeatures

Changes a window's features.

```
OSStatus HIWindowChangeFeatures (
   WindowRef inWindow,
   UInt64 inSetThese,
   UInt64 inClearThese
);
```

Parameters

inWindow

The window whose features are to be changed.

inSetThese

The feature bits to set. For details, see "Window Feature Bits" (page 2011).

inClearThese

The feature bits to clear. For details, see "Window Feature Bits" (page 2011).

Return Value

A result code. See "Window Manager Result Codes" (page 2051).

Discussion

`HIWindowChangeFeatures` changes the features of a window on the fly. This function should only be used by custom window definitions or window frame views.

Availability

Available in Mac OS X v10.3 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

HIWindowConstrain

Moves and resizes a window to be within a specified bounding rectangle.

```
OSStatus HIWindowConstrain (
   WindowRef inWindowRef,
   WindowRegionCode inRegionCode,
   WindowConstrainOptions inOptions,
    HICoordinateSpace inSpace,
   const HIRect *inScreenBounds,
    const HISize *inMinimumSize,
   HIRect *ioBounds
);
```

Parameters

inWindowRef

The window to constrain.

inRegionCode

The window region to constrain. For a list of possible values, see "Window Region Constants" (page 2021).

inOptions

Flags controlling how the window is constrained. For a list of possible options, see "Window Constrain Options" (page 2028).

inSpace

The coordinate space in which the `inScreenBounds`, `inMinimumSize`, and `ioBounds` parameters are expressed. This parameter must be either `kHICoordSpaceScreenPixels` or `kHICoordSpace72DPIGlobal`.

inScreenBounds

A rectangle within which to constrain the window. You may pass `NULL` if you don't need to specify a screen bounds. If `NULL`, the window is constrained to the screen that has the greatest intersection with the specified window region.

inMinimumSize

A minimum size that should be kept within the specified screen bounds. This parameter is ignored if the `kWindowConstrainMoveMinimum` option is not set. Even if that option is set, you may still pass `NULL` if you don't need to customize the minimum dimensions.

ioBounds

If the `inOptions` parameter contains `kWindowConstrainUseSpecifiedBounds`, then this parameter should be a bounding rectangle of the specified window region. The bounding rectangle does not have to match the actual current bounds of the specified region; it may be a hypothetical bounds that you would like to constrain without actually moving the window to that location. On output, contains the new structure bounds of the window. You may pass `NULL` if you don't need the window bounds returned.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Availability
Available in Mac OS X v10.5 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

HIWindowCopyAvailablePositioningShape

Copies the available window positioning shape on a display.

```
OSStatus HIWindowCopyAvailablePositioningShape (
   CGDirectDisplayID inDisplay,
   HICoordinateSpace inSpace,
   HIShapeRef *outShape
);
```

Parameters

inDisplay

The display for which to find the available shape. May be `kCGNullDirectDisplay` to request the shape of the main display.

inSpace

The coordinate space in which the positioning shape should be returned. This parameter must be either `kHICoordSpaceScreenPixel` or `kHICoordSpace72DPIGlobal`.

outShape

A pointer to a shape (an HIShape object). On output, the shape describes the available bounds for the specified display. This shape is returned in the specified coordinate space. You should release the shape when you no longer need it.

Discussion

This function finds the area on the display in which a window may be positioned without intersecting or overlapping the menu bar, Dock, or other UI provided by the operating system. This function differs from `HIWindowGetAvailablePositioningBounds` (page 1885) in that the bounds version removes the entire area that may theoretically be covered by the Dock, even if the Dock does not currently reach from edge to edge of the display on which it is positioned. The shape version includes the area at the sides of the Dock that is not covered by the Dock.

Availability

Available in Mac OS X v10.5 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

HIWindowCopyDrawers

Obtains an array of the drawers that are attached to a window.

```
OSStatus HIWindowCopyDrawers (
    WindowRef inWindow,
    CFArrayRef *outDrawers
);
```

Parameters

inWindow

The parent window to access.

outDrawers

A pointer to a Core Foundation array. On output, each array entry is a drawer window attached to the parent window specified in the `inWindow` parameter. The array will be valid, but empty, if the parent window has no drawers. You should release the array when you no longer need it.

Return Value

A result code. See "Window Manager Result Codes" (page 2051).

Availability

Available in Mac OS X v10.5 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

HIWindowCopyShape

Retrieves a shape that describes a region of a window.

```
OSStatus HIWindowCopyShape (
    WindowRef inWindow,
    WindowRegionCode inRegion,
    HICoordinateSpace inSpace,
    HIShapeRef *outShape
);
```

Parameters

inWindow

The window to access.

inRegion

The window region whose shape you want to obtain. For a list of possible values, see "Window Region Constants" (page 2021).

inSpace

The coordinate space in which the shape should be returned. This parameter must be `kHICoordSpaceWindow`, `kHICoordSpaceScreenPixel`, or `kHICoordSpace72DPIGlobal`.

outShape

A pointer to a shape (an HIShape object). On output, the shape describes the specified window region. The shape is returned in the specified coordinate space. You should release the shape when you no longer need it. If the window does not support the specified window region, no shape is returned.

Return Value

A result code. If the window does not support the specified window region, the result returned is `errWindowRegionCodeInvalid`. For other possible values, see "Window Manager Result Codes" (page 2051).

Availability

Available in Mac OS X v10.5 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

HIWindowCreate

Creates a standard or custom window.

```
OSStatus HIWindowCreate (
    WindowClass inClass,
    const int *inAttributes,
    const WindowDefSpec *inDefSpec,
    HICoordinateSpace inSpace,
    const HIRect *inBounds,
    WindowRef *outWindow
);
```

Parameters

inClass

The class of window to be created. For a list of possible classes, see "Window Class Constants" (page 1988).

inAttributes

A zero-terminated array of window attribute constants. Each array entry specifies an attribute of the window to set. You may pass `NULL` if you don't need to set any attributes. For a list of possible attributes, see "Window Attribute Identifiers" (page 1992).

inDefSpec

A pointer to a custom window proc ID or root view for the window. You may pass `NULL` if you don't need to customize the window.

inSpace

The coordinate space in which the content bounds is expressed. This parameter must be either `kHICoordSpaceScreenPixels` or `kHICoordSpace72DPIGlobal`.

inBounds

A pointer to the bounds of the content area of the window in the coordinate space specified by the `inSpace` parameter. If you specify non-integral coordinates, they will be rounded to the nearest integral value in screen pixel space when the window is actually positioned or sized.

outWindow

A pointer to a window variable. On output, the variable contains the new window.

Return Value

A result code. See "Window Manager Result Codes" (page 2051).

Discussion

This function makes it possible to create windows with content bounds expressed in different coordinate spaces. In Mac OS X v10.5 and later, you can use this function in place of `CreateNewWindow` (page 1815) or `CreateCustomWindow` (page 1814) to create a window from a set of parameters.

Most developers will want to work primarily in the 72 DPI coordinate space. Doing so makes your code independent of the current user interface scale factor, and eases source compatibility with earlier versions of Mac OS X that do not support resolution independence. However, there are also certain cases where your application must express your window's bounds in pixel coordinates; primarily when you need to position your windows so they exactly align with each other or with some other fixed location, such as the edge of the display. For these cases, you should use the screen pixel coordinate space.

Availability

Available in Mac OS X v10.5 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

HIWindowCreateCollapsedDockTileContext

Creates a Quartz graphics context for drawing a collapsed window's Dock tile.

```
OSStatus HIWindowCreateCollapsedDockTileContext (
   WindowRef inWindow,
   CGContextRef *outContext,
   HISize *outContextSize
);
```

Parameters

inWindow

The collapsed window.

outContext

A pointer to a `CGContextRef` variable. On output, the variable contains the graphics context for drawing the window's Dock tile.

outContextSize

A pointer to a `HISize` structure. On output, the structure contains the width and height of the area in which to draw.

Return Value

A result code. If the window is not collapsed, the result code is `windowWrongStateErr`. For other possible values, see "Window Manager Result Codes" (page 2051).

Discussion

When you are finished drawing in the graphics context, you should:

1. Call `CGContextFlush` to ensure that your drawing appears onscreen.
2. Call `HIWindowReleaseCollapsedDockTileContext` (page 1892) to release the context. Do not call `CFRelease` or `CGContextRelease` to release the context, or you may leak system resources.

Availability

Available in Mac OS X v10.5 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

HIWindowFindAtLocation

Finds a window in the current process at a specified location.

```
OSStatus HIWindowFindAtLocation (
    const HIPoint *inLocation,
    HICoordinateSpace inSpace,
    WindowRef inStartWindow,
    OptionBits inOptions,
    WindowRef *outWindow,
    WindowPartCode *outWindowPart,
    HIPoint *outWindowLocation
);
```

Parameters

inLocation

The location, in global coordinates, at which to search for a window.

inSpace

The coordinate space in which the location is expressed. This parameter must be either `kHICoordSpaceScreenPixel` or `kHICoordSpace72DPIGlobal`.

inStartWindow

The window at which to start the search, inclusive. Pass `kFirstWindowOfClass` to start the search at the beginning of the window list. Passing `NULL` will cause the search to start at the end of the window list, and therefore no window will be found.

inOptions

Reserved. Pass zero.

outWindow

A pointer to a window variable. On output, the variable contains the window in the current process at the specified location, if any, or `NULL` if no window is found.

outWindowPart

A pointer to a window part code variable. On output, the variable contains the window part that was hit. You may pass `NULL` if you don't need this information.

outWindowLocation

A pointer to a point variable. On output, the variable contains the specified location transformed into window-relative coordinates, taking into account any window transform or magnification. You may pass `NULL` if you don't need this information.

Return Value
A result code. If no window is found that satisfies the search criteria, this function returns `errWindowNotFound`. For other possible return values, see "Window Manager Result Codes" (page 2051).

Discussion
This function searches the window list of the current process for a window that contains the specified location. If you need to determine whether the window is of a particular class, you can use the function `GetWindowClass` (page 1844) and compare the result to the desired class.

Availability
Available in Mac OS X v10.5 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

HIWindowFlush

Flushes any dirty areas a window might have.

```
OSStatus HIWindowFlush (
   WindowRef inWindow
);
```

Parameters

window

The window to flush.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
This function allows you to manually flush dirty areas of a window to the screen. This is the preferred way to flush window buffers in Mac OS X v10.3 and later. If called for a composited window, this function also renders any views in the window that are invalid.

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

HIWindowFromCGWindowID

Returns the window in the current process with a specified Quartz window ID.

```
WindowRef HIWindowFromCGWindowID (
    CGWindowID inWindowID
);
```

Parameters

inWindowID

The window ID, as returned by `HIWindowGetCGWindowID` (page 1886) or `CGWindowListCopyWindowInfo`.

Return Value

The window to which the window ID is assigned. This function returns `NULL` if the window ID is invalid or if it refers to a window in another process.

Discussion

This function returns the window in the current process to which the specified window ID is assigned by the window server when the window is created.

Availability

Available in Mac OS X v10.5 and later.

Not available to 64-bit applications.

See Also

`HIWindowGetCGWindowID` (page 1886)

Declared In

`MacWindows.h`

HIWindowGetAvailability

Obtains the availability of a window during Exposé or in Spaces.

```
OSStatus HIWindowGetAvailability (
   WindowRef inWindow,
   HIWindowAvailability *outAvailability
);
```

Parameters

inWindow

The window whose availability is to be obtained.

outAvailability

On exit, the current setting of the window's availability bits. For details, see "Window Availability Constants" (page 2046).

Return Value

A result code. See "Window Manager Result Codes" (page 2051).

Availability

Available in Mac OS X v10.4 and later.

Not available to 64-bit applications.

Declared In
MacWindows.h

HIWindowGetAvailablePositioningBounds

Gets the available window positioning bounds on a display.

```
OSStatus HIWindowGetAvailablePositioningBounds (
   CGDirectDisplayID inDisplay,
   HICoordinateSpace inSpace,
   HIRect *outAvailableRect
);
```

Parameters

inDisplay

The display for which to find the available bounds. May be `kCGNullDirectDisplay` to request the bounds of the main display.

inSpace

The coordinate space in which the positioning bounds should be returned. This must be either `kHICoordSpaceScreenPixel` or `kHICoordSpace72DPIGlobal`.

outAvailableRect

A pointer to a rectangle provided by the caller. On output, the rectangle contains the available bounds for the specified display. This rectangle is returned in the specified coordinate space.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
This function gets the bounds of the display not including the menu bar and Dock, if located on that display.

Availability
Available in Mac OS X v10.5 and later.
Not available to 64-bit applications.

Declared In
MacWindows.h

HIWindowGetBounds

Gets the bounds of a specified region of a window.

```
OSStatus HIWindowGetBounds (
    WindowRef inWindow,
    WindowRegionCode inRegion,
    HICoordinateSpace inSpace,
    HIRect *outBounds
);
```

Parameters

inWindow

The window to access.

inRegion

The window region. For a list of possible values, see "Window Region Constants" (page 2021).

inSpace

The coordinate space in which the bounds should be returned. This parameter must be `kHICoordSpaceWindow`, `kHICoordSpaceScreenPixel`, or `kHICoordSpace72DPIGlobal`.

outBounds

A pointer to an `HIRect` (page 2324) structure. On output, the structure contains the origin and size of the bounding rectangle of the specified window region. If the window does not support the region, the structure is not modified.

Return Value

A result code. If the window does not support the specified window region, the result returned is `errWindowRegionCodeInvalid`.

Availability

Available in Mac OS X v10.5 and later.

Not available to 64-bit applications.

See Also

`HIWindowSetBounds` (page 1893)

Declared In

`MacWindows.h`

HIWindowGetCGWindowID

Returns the Quartz window ID assigned to a window.

```
CGWindowID HIWindowGetCGWindowID (
   WindowRef inWindow
);
```

Parameters

inWindow

The window to access.

Return Value

The window ID of the specified window, or zero if the window is invalid.

Discussion

This function returns the window ID assigned by the window server when a window is created. The window ID is not generally useful with any other Carbon function, but may be used with other Mac OS X functions that require a window ID, such as functions in OpenGL.

Availability

Available in Mac OS X v10.5 and later.

Not available to 64-bit applications.

See Also

`HIWindowFromCGWindowID` (page 1884)

Declared In

`MacWindows.h`

HIWindowGetGreatestAreaDisplay

Finds the display with the greatest area of intersection with a window region.

```
OSStatus HIWindowGetGreatestAreaDisplay (
   WindowRef inWindow,
   WindowRegionCode inRegion,
   HICoordinateSpace inSpace,
   CGDirectDisplayID *outGreatestDisplay,
   HIRect *outGreatestDisplayRect
);
```

Parameters

inWindow

The window to compare against.

inRegion

The window region to compare against. See "Window Region Constants" (page 2021) for a list of possible values.

inSpace

The coordinate space in which the display bounds should be returned. This must be either `kHICoordSpaceScreenPixel` or `kHICoordSpace72DPIGlobal`.

outGreatestDisplay

A pointer to a display ID provided by the caller, or `NULL` if you don't need this information. On output, the display ID contains the display with the greatest intersection.

outGreatestDisplayRect

A pointer to a rectangle provided by the caller, or `NULL` if you don't need this information. On output, the rectangle contains the bounds of the display with the greatest intersection. If the display with the greatest intersection also contains the menu bar, the rectangle excludes the menu bar area. This rectangle is returned in the specified coordinate space.

Return Value

A result code. See "Window Manager Result Codes" (page 2051).

Availability

Available in Mac OS X v10.5 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

HIWindowGetIdealUserState

Gets the bounds of a window's content region in its user state.

```
OSStatus HIWindowGetIdealUserState (
   WindowRef inWindow,
   HICoordinateSpace inSpace,
   HIRect *outUserState
);
```

Parameters

inWindow

The window to access.

inSpace

The coordinate space in which the user state bounds should be returned. This parameter must be `kHICoordSpaceScreenPixel` or `kHICoordSpace72DPIGlobal`.

outUserState

A pointer to a structure of type `HIRect`. On return, this rectangle contains the global coordinates of the window's content region when zoomed in. If the window has not yet been zoomed, this rectangle contains the window's current content bounds.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
This function returns information about the window's user state most recently recorded by the function `ZoomWindowIdeal` (page 1972).

Availability
Available in Mac OS X v10.5 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

HIWindowGetProxyFSRef

Obtains the `FSRef` used to determine the proxy icon for a window.

```
OSStatus HIWindowGetProxyFSRef (
   WindowRef window,
   FSRef *outRef
);
```

Parameters

inWindow

The window whose proxy `FSRef` is to be obtained.

outRef

On exit, the `FSRef` for the window's proxy icon.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
If the specified window's proxy icon has been specified using `HIWindowSetProxyFSRef` (page 1894) or `SetWindowProxyAlias` (page 1949), `HIWindowGetProxyFSRef` returns `noErr` and a valid `FSRef` for the window's proxy icon. If the window has no proxy icon, or if the icon was specified by calling `SetWindowProxyCreatorAndType` or `SetWindowProxyIcon`, this function returns an error.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

HIWindowGetScaleMode

Obtains the window's scale mode and the application's display scale factor.

```
OSStatus HIWindowGetScaleMode (
   WindowRef inWindow,
   HIWindowScaleMode *outMode,
   CGFloat *outScaleFactor
);
```

Parameters

inWindow

The window whose scale mode is to be obtained.

outMode

On exit, an `HIWindowScaleMode` indicating the window's scale mode. For details, see "Window Scale Mode Constants" (page 2047).

outScaleFactor

On exit, a float indicating the display scale factor for the application. Pass `NULL` if you are not interested in acquiring the scale factor.

Return Value

A result code. See "Window Manager Result Codes" (page 2051).

Discussion

The window's scale mode is based on the application's display scale factor and any resolution-independent attributes specified at window creation time. Applications and the views within the window can use the scale mode and display scale factor to draw properly the content of a window for a particular scale mode.

Availability

Available in Mac OS X v10.4 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

HIWindowGetThemeBackground

Gets the theme background brush for a window.

```
OSStatus HIWindowGetThemeBackground (
    WindowRef inWindow,
    ThemeBrush *outThemeBrush
);
```

Parameters

inWindow

The window from which to get the brush.

outThemeBrush

A pointer to a theme brush. On output, the brush is the window's theme background brush.

Return Value

A result code. If no brush is found, `themeNoAppropriateBrushErr` is returned.

Discussion
This function gets the theme background brush previously set by calling the function `SetThemeWindowBackground` (page 1933).

Availability
Available in Mac OS X v10.5 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

HIWindowInvalidateShadow

Recalculates a window's shadow.

```
OSStatus HIWindowInvalidateShadow (
   WindowRef inWindow
);
```

Parameters
inWindow
The window whose shadow is to be recalculated.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
`HIWindowInvalidateShadow` is not typically used by applications. It is useful if your application has customized window frames that change shape dynamically. After you have drawn the new window shape, you should call `HIWindowInvalidateShadow` to recalculate the shadow so that it follows the new window shape.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

HIWindowIsAttributeAvailable

Returns a Boolean value indicating whether a window attribute is valid for a specified window class.

```
Boolean HIWindowIsAttributeAvailable (
   WindowClass inClass,
   int inAttr
);
```

Parameters
inClass
The window class to test.

inAttr

The window attribute to test. You must specify one of the window attributes described in "Window Attribute Identifiers" (page 1992).

Return Value
If `true`, the window class supports the specified attribute. Otherwise, `false`.

Availability
Available in Mac OS X v10.5 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

HIWindowIsDocumentModalTarget

Determines if a window is currently the target window of another document modal window, such as a sheet.

```
Boolean HIWindowIsDocumentModalTarget (
   WindowRef inWindow,
   WindowRef *outOwner
);
```

Parameters

inWindow

The window to query.

outOwner

If this function returns `true`, `inWindow` is the target of a document modal window and `outOwner` is set to the document modal window. If this function does not return `true`, `outOwner` is undefined. Pass `NULL` if you don't want the owner's window reference.

Return Value
A Boolean whose value is `true` if the window specified by `inWindow` is currently the target of a document modal window; otherwise, `false`.

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

HIWindowIsInStandardState

Returns a Boolean value indicating whether a window is zoomed out to its standard state.

```
Boolean HIWindowIsInStandardState (
   WindowRef inWindow,
   const HISize *inIdealSize,
    HICoordinateSpace inSpace,
   HIRect *outIdealStandardState
);
```

Parameters

inWindow

The window whose zoom state is to be determined.

inIdealSize

The ideal width and height of the window's content region, regardless of the actual screen device dimensions. If you set `inIdealSize` to `NULL`, this function examines the dimensions stored in the `stdState` field of the `WStateData` structure.

inSpace

The coordinate space in which the ideal size is expressed and in which the standard state bounds should be returned. This parameter must be either `kHICoordSpaceScreenPixel` or `kHICoordSpace72DPIGlobal`.

outIdealStandardState

A pointer to an `HIRect` variable. On return, the variable contains the bounds of the content region of the window in its standard state, based on the data supplied in the `inIdealSize` parameter. You may pass `NULL` if you do not need this information.

Return Value

If `true`, the window is currently in its standard state. If `false`, the window is currently in its user state.

Discussion

This function compares the window's current dimensions to those in the `inIdealSize` parameter to determine if the window is currently in its standard state. You can use this function to decide whether a user's click in the zoom box is a request to zoom to the user state or the standard state, as described in the function `ZoomWindowIdeal` (page 1972). You can also use this function to determine the size and position of the standard state that the Window Manager would calculate for a window, given a specified ideal size; this value is returned in the `outIdealStandardState` parameter.

Availability

Available in Mac OS X v10.5 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

HIWindowReleaseCollapsedDockTileContext

Releases a Quartz graphics context for drawing a collapsed window's Dock tile.

```
OSStatus HIWindowReleaseCollapsedDockTileContext (
   WindowRef inWindow,
   CGContextRef inContext
);
```

Parameters

inWindow

The collapsed window.

inContext

The graphics context to release. On return, the context is invalid and should no longer be used.

Return Value
A result code. If the window is not collapsed, the result code is `windowWrongStateErr`. For other possible values, see "Window Manager Result Codes" (page 2051).

Discussion
To ensure that your drawing appears onscreen, you should call `CGContextFlush` before calling this function to release the context. Do not call `CFRelease` or `CGContextRelease` to release the context, or you may leak system resources.

Availability
Available in Mac OS X v10.5 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

HIWindowSetBounds

Sets the bounds of a window based on either the structure or content region.

```
OSStatus HIWindowSetBounds (
    WindowRef inWindow,
    WindowRegionCode inRegion,
    HICoordinateSpace inSpace,
    HIRect *inBounds
);
```

Parameters

inWindow

The window to access.

inRegion

The window region on which to base the window's new bounds. This parameter must be either `kWindowStructureRgn` or `kWindowContentRgn`.

inSpace

The coordinate space in which the bounds are expressed. This parameter must be `kHICoordSpaceWindow`, `kHICoordSpaceScreenPixel`, or `kHICoordSpace72DPIGlobal`.

inBounds

A pointer to an `HIRect` (page 2324) structure that specifies the origin and size of the bounding rectangle of the specified window region. If the coordinate space is `kHICoordSpaceWindow`, then the origin of the bounds is a window-relative value. Therefore, you can use this coordinate space to resize a window without first getting its current bounds by setting the origin to (0,0), or you can offset a window from its current position by setting the origin to the offset amount and the size to the window's current size.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Availability
Available in Mac OS X v10.5 and later.
Not available to 64-bit applications.

See Also
HIWindowGetBounds (page 1885)

Declared In
MacWindows.h

HIWindowSetIdealUserState

Sets the bounds of a window's content region in its user state.

```
OSStatus HIWindowSetIdealUserState (
    WindowRef inWindow,
    HICoordinateSpace inSpace,
    const HIRect *inUserState
);
```

Parameters

inWindow
The window to access.

inSpace
The coordinate space in which the user state bounds are expressed. This parameter must be either kHICoordSpaceScreenPixel or kHICoordSpace72DPIGlobal.

inUserState
The new bounds (position and size) of the window's content region in its user state.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
A window's ideal user state is used by the function ZoomWindowIdeal (page 1972) when zooming in.

Availability
Available in Mac OS X v10.5 and later.
Not available to 64-bit applications.

Declared In
MacWindows.h

HIWindowSetProxyFSRef

Sets the proxy icon for a window using an FSRef to a file system object.

```
OSStatus HIWindowSetProxyFSRef (
   WindowRef window,
   const FSRef *inRef
);
```

Parameters

inWindow
The window whose proxy icon is to be set.

inRef
The file system object the window represents.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
This function determines the window's proxy icon by asking Icon Services for the icon for the object specified by `inRef`.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

HIWindowSetToolbarView

Sets a custom toolbar view for a window.

```
OSStatus HIWindowSetToolbarView (
    WindowRef inWindow,
    HIViewRef inView
);
```

Parameters

`inWindow`
The window whose toolbar view to set.

`inView`
The custom toolbar view for the window. You may pass `NULL` to remove the custom view from the window. Setting a custom view will also remove any HIToolbar that is associated with the window.

After a custom toolbar view has been set, the window owns the view and will release it automatically when the window is destroyed, or when a different custom view or standard HIToolbar is set for the window.

Return Value
A result code.

Discussion
This function is provided for use by applications that cannot use the HIToolbar API. For best compatibility with future versions of Mac OS X, you should use HIToolbar if possible. However, if HIToolbar is not sufficient for your needs, you can use this function to provide a custom toolbar view that will be placed at the standard location inside the window frame.

You are responsible for defining the appearance and behavior of the view. You cannot use this function to customize the view that is associated with an HIToolbar; a window with an HIToolbar uses a standard view that cannot be customized. When using a custom toolbar view, no function that takes an HIToolbar will work with that window. For more information about custom toolbar views, see `MacWindows.h`.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

HIWindowShowsFocus

Returns a Boolean value indicating whether a window's content should show focus indicators such as focus rings.

```
Boolean HIWindowShowsFocus (
    WindowRef inWindow
);
```

Parameters

inWindow
The window to access.

Return Value

If `true`, a window's content should show focus indicators; otherwise `false`.

Discussion

This function returns `true` if the window is either the modeless focus or the effective focus.

Availability

Available in Mac OS X v10.5 and later.
Not available to 64-bit applications.

Declared In

`MacWindows.h`

HIWindowTestAttribute

Returns a Boolean value indicating whether a window has a specified attribute.

```
Boolean HIWindowTestAttribute (
   WindowRef inWindow,
   int inAttr
);
```

Parameters

inWindow
The window to test.

inAttr
The window attribute to test. You must specify one of the window attributes described in "Window Attribute Identifiers" (page 1992).

Return Value

If `true`, the window has the specified attribute. Otherwise, `false`.

Availability

Available in Mac OS X v10.5 and later.
Not available to 64-bit applications.

Declared In

`MacWindows.h`

HIWindowTrackProxyDrag

Tracks the drag of a window proxy icon.

```
OSStatus HIWindowTrackProxyDrag (
    WindowRef inWindow,
    EventRef inEvent,
    DragRef inDrag
);
```

Parameters

inWindow

The window whose proxy icon to drag.

inEvent

The event that resulted in starting a drag. This will most commonly be `kEventControlTrack`, but any event with `kEventParamMouseLocation` and `kEventParamKeyModifiers` parameters is all that is required.

inDrag

The proxy icon drag reference. You may pass `NULL` if you want the Window Manager to create and populate the drag reference. The Window Manager will add its own drag flavors to the drag even if you pass a pre-created drag reference.

Discussion

You can use this function to manage the dragging of the proxy icon in your application's windows. If you use the standard window event handler and you do not need to customize the proxy icon drag process, you may rely on the standard handler to call this function.

If you want to allow the Window Manager to create the drag reference and populate it with drag flavors, you should pass `NULL` in the `inDrag` parameter. If you want to create the drag reference yourself and add your own drag flavors, you should call `BeginWindowProxyDrag` (page 1802) to create the drag reference, add your own flavors, call `HIWindowTrackProxyDrag` to track the proxy icon drag, and then call `EndWindowProxyDrag` (page 1826) to release the drag reference.

A proxy icon may only be dragged if the window represented by the proxy icon is not modified, as indicated by the `IsWindowModified` (page 1907) function. This restriction exists because a proxy icon is a representation of a physical file system object, and dragging the proxy icon may result in the Finder making a copy of the file system object. If the window is modified, then it contains user data that has not yet been saved to disk; making a copy of the file system object would result in a stale copy that did not contain the user's current data.

By default, all newly created windows are considered to be dirty. The application must pass false to `SetWindowModified` (page 1946) before the proxy icon will be draggable. In Mac OS X v10.3 and later, the proxy icon is also draggable in dirty windows if the proxy icon was provided using the `SetWindowProxyIcon` (page 1951) or `SetWindowProxyCreatorAndType` (page 1950) functions. Dragging is allowed in this case because the window does not represent an actual file system object, and therefore there is no risk of user data loss.

Availability

Available in Mac OS X v10.5 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

InstallWindowContentPaintProc

Installs a window content painting callback. (Deprecated in Mac OS X v10.5. Use a `kEventControlDraw` Carbon event handler on a window's content view instead.)

```
OSStatus InstallWindowContentPaintProc (
   WindowRef window,
   WindowPaintUPP paintProc,
   WindowPaintProcOptions options,
   void *refCon
);
```

Parameters

window

The window whose default content painting function you want to override.

paintProc

A UPP to your window painting callback function. See `WindowPaintProcPtr` (page 1978) for more information about the format of this function.

options

The options that are to be set. See "Window Paint Callback Options" (page 2043) for a list of possible values.

refCon

Application-defined data. This data is passed to your callback when it is called.

Return Value

A result code. See "Window Manager Result Codes" (page 2051).

Discussion

To remove a previously-installed paint proc (returning to the standard window manager erase-to-white content painting), pass `NULL` in the `paintProc` and `refCon` parameters.

Special Considerations

Instead of using this function, you should install a Carbon event handler for the `kEventControlDraw` event on a window's content view.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

InvalWindowRect

Adds a rectangle to a window's update region.

```
OSStatus InvalWindowRect (
   WindowRef window,
   const Rect *bounds
);
```

Parameters

window

The window containing the rectangle you want to be updated.

bounds

Set this structure to specify, in local coordinates, a rectangle to be added to the window's update region.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
The `InvalWindowRect` function informs the Window Manager that an area of a window should be redrawn.

See also the functions `ValidWindowRect` (page 1969) and `InvalWindowRgn` (page 1899).

Special Considerations

This function should not be used on composited windows. Modifying a composited window's update region does not affect the area of the window to be drawn. A composited window does not use its window update region to control drawing. Instead, a composited window determines what to draw by looking at the invalid regions of the views contained in the window.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
CarbonSketch
HID Explorer

Declared In
`MacWindows.h`

InvalWindowRgn

Adds a region to a window's update region.

```
OSStatus InvalWindowRgn (
   WindowRef window,
   RgnHandle region
);
```

Parameters

window

The window containing the region that you want to update.

region

Set this region to specify, in local coordinates, the area to be added to the window's update region.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
The `InvalWindowRgn` function informs the Window Manager that an area of a window should be redrawn.

See also the functions `InvalWindowRect` (page 1898) and `ValidWindowRgn` (page 1969).

Special Considerations

This function should not be used on composited windows. Modifying a composited window's update region does not affect the area of the window to be drawn. A composited window does not use its window update region to control drawing. Instead, a composited window determines what to draw by looking at the invalid regions of the views contained in the window.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

InvokeWindowDefUPP

Invokes the UPP for a window definition. (Deprecated in Mac OS X v10.5. The WDEF interface is deprecated; use a custom HIView to draw your custom window frame instead.)

```
long InvokeWindowDefUPP (
   short varCode,
   WindowRef window,
   short message,
   long param,
   WindowDefUPP userUPP
);
```

Parameters

varCode
The window's variation code.

window
The window whose UPP is to be invoked.

message
The message.

param
The parameter.

userUPP
The UPP to invoke.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

InvokeWindowPaintUPP

Invokes the UPP for the specified painting region. (Deprecated in Mac OS X v10.5. The window content painting interface is deprecated; use a `kEventControlDraw` Carbon event handler on a compositing window's content view instead.)

```
OSStatus InvokeWindowPaintUPP (
   GDHandle device,
   GrafPtr qdContext,
   WindowRef window,
   RgnHandle inClientPaintRgn,
   RgnHandle outSystemPaintRgn,
   void *refCon,
   WindowPaintUPP userUPP
);
```

Parameters

device

The graphics device on which the window background should be painted.

qdContext

The QuickDraw port in which the window background should be painted.

window

The window whose UPP is to be invoked.

inClientPaintRgn

The region of the window background that needs to be painted, in local coordinates.

outSystemPaintRgn

On return, the region of the window background that the paint proc requests the Window Manager to paint.

refCon

Application-defined data.

userUPP

The UPP to invoke. For more information on this data type, see `WindowPaintUPP` (page 1987).

Return Value

A result code. See "Window Manager Result Codes" (page 2051).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`MacWindows.h`

IsValidWindowClass

Determines whether a given window class is valid.

```
Boolean IsValidWindowClass (
   WindowClass inClass
);
```

Parameters

inClass

The window class to query.

Return Value

A Boolean whose value is `true` if the window class is valid; otherwise, `false`.

Availability

Available in Mac OS X v10.1 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

IsValidWindowPtr

Reports whether a pointer is a valid window pointer.

```
Boolean IsValidWindowPtr (
   WindowRef possibleWindow
);
```

Parameters

possibleWindow

The window to query.

Return Value

A Boolean whose value is `true` if the specified pointer is a valid window pointer; otherwise, `false`.

Discussion

This function is primarily intended for use with debugging your application.

Special Considerations

The `IsValidWindowPtr` function is a processor-intensive call.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

IsWindowActive

Indicates whether the specified window is active.

```
Boolean IsWindowActive (
   WindowRef inWindow
);
```

Parameters

inWindow

The window to query.

Return Value

Returns `true` if the window is active, `false` otherwise.

Discussion

Whether a window is considered active is determined by its activation scope, highlighting, and z-order. For windows that have an activation scope of `kWindowActivationScopeAll`, a window is active if it is the window returned by the `ActiveNonFloatingWindow` (page 1800) function or if it is in the same window group as the window returned by `ActiveNonFloatingWindow` and the window group has the `kWindowGroupAttrSharedActivation` attribute. For windows that have some other activation scope, the window is active if its window frame is highlighted and the window is the frontmost window in its window group.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

IsWindowCollapsable

Determines whether a window can be collapsed.

```
Boolean IsWindowCollapsable (
   WindowRef window
);
```

Parameters

window

The window to be examined.

Return Value

If `true`, the window can be collapsed; otherwise, `false`.

Discussion

You can call the `IsWindowCollapsable` function to determine if a given window can be collapsed by the `CollapseWindow` (page 1810) function. In Mac OS X, the presence or absence of the `kWindowCollapseBoxAttribute` is the primary way of determining whether a window can be collapsed. If that attribute is not present, the Window Manager checks for the `kWindowCanCollapse` feature bit.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

IsWindowCollapsed

Determines whether a window is currently collapsed.

```
Boolean IsWindowCollapsed (
   WindowRef window
);
```

Parameters

window
The window to be examined.

Return Value

If `true`, the window is collapsed. If `false`, the window is expanded.

Discussion

On Mac OS 9, only window definition functions that return the feature bit `kWindowCanCollapse` in response to a `kWindowGetFeatures` message support this function; for more information, see `GetWindowFeatures` (page 1847). Typically, a window's content region is empty in a collapsed state. In Mac OS X, the presence or absence of the `kWindowCollapseBoxAttribute` attribute determines whether a window can be collapsed.

Availability

Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In

`MacWindows.h`

IsWindowContainedInGroup

Determines if a window is a member of a window group or any of its subgroups.

```
Boolean IsWindowContainedInGroup (
   WindowRef inWindow,
   WindowGroupRef inGroup
);
```

Parameters

inWindow
The window to query.

inGroup
The window group to query.

Return Value

A Boolean whose value is `true` if `inWindow` is a member of the window group specified by `inGroup`, or if `inWindow` is a member of a window group that is a member of the window group specified by `inGroup`. Otherwise, this function returns `false`.

Discussion

This function returns `true` if group A contains window A. It also returns `true` if group A contains group B and group B contains window A.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
MacWindows.h

IsWindowHilited

Indicates whether the window frame is currently highlighted.

```
Boolean IsWindowHilited (
   WindowRef window
);
```

Parameters

window
The window to query.

Return Value
A Boolean value indicating whether or not the window frame is highlighted. If `true`, the window is visible. If `false`, the window frame is not highlighted.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
MacWindows.h

IsWindowInStandardState

Determines whether a window is currently zoomed in to the user state or zoomed out to the standard state.

```
Boolean IsWindowInStandardState (
    WindowRef inWindow,
    const Point *inIdealSize,
    Rect *outIdealStandardState
);
```

Parameters

inWindow
The window whose zoom state is to be determined.

inIdealSize
Set the `Point` structure to contain the ideal width and height of the window's content region, regardless of the actual screen device dimensions. If you set `inIdealSize` to `NULL`, `IsWindowInStandardState` examines the dimensions stored in the `stdState` field of the `WStateData` structure.

outIdealStandardState
On input, a pointer to a structure of type `Rect`. On return, the rectangle contains the global coordinates for the content region of the window in its standard state, based on the data supplied in the `inIdealSize` parameter. You may pass `NULL` if you do not want to receive this data.

Return Value
A Boolean whose value is `true` if the window is currently in its standard state; `false` if the window is currently in the user state.

Discussion
The `IsWindowInStandardState` function compares the window's current dimensions to those referred to by the `inIdealSize` parameter to determine if the window is currently in the standard state. Your application may use `IsWindowInStandardState` to decide whether a user's click of the zoom box is a request to zoom to the user state or the standard state, as described in the function `ZoomWindowIdeal` (page 1972). Your application may also use `IsWindowInStandardState` to determine the size and position of the standard state that the Window Manager would calculate for a window, given a specified ideal size; this value is produced in the `outIdealStandardState` parameter.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

See Also
`HIWindowIsInStandardState` (page 1891)

Declared In
`MacWindows.h`

IsWindowLatentVisible

Indicates whether a window is visible onscreen or is latently visible but not currently onscreen.

```
Boolean IsWindowLatentVisible (
   WindowRef inWindow,
   WindowLatentVisibility *outLatentVisible
);
```

Parameters

inWindow
The window to query.

outLatentVisible
If the window is onscreen, the latent visibility is zero. If the window is offscreen, this parameter returns the latent visibility flags of the window. If any of the flags are set, the window is latently visible.

Return Value
A Boolean whose value is `true` if the window is currently onscreen; otherwise, `false`.

Discussion
All windows are either onscreen or offscreen. A window that is offscreen may still be latently visible. This occurs, for example, when a floating window is hidden as an application is suspended. The floating window is not visible onscreen, but it is latently visible and is only hidden due to the suspended state of the application. When the application becomes active again, the floating window will be placed back onscreen.

Availability
Available in Mac OS X v10.1 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

IsWindowModified

Obtains the modification state of the specified window.

```
Boolean IsWindowModified (
   WindowRef window
);
```

Parameters

window

The window whose modification state is to be obtained.

Return Value

`true` if the content of the window has been modified; otherwise, `false`. Newly created windows start out with their modification state automatically set to `true`.

Discussion

Your application can use the functions `IsWindowModified` and `SetWindowModified` (page 1946) instead of maintaining its own separate record of the modification state of the content of a window.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

QTCarbonShell

Declared In

`MacWindows.h`

IsWindowPathSelectClick

Reports whether a mouse click should activate the window path pop-up menu. (Deprecated in Mac OS X v10.5. Use `IsWindowPathSelectEvent` (page 1908) instead.)

```
Boolean IsWindowPathSelectClick (
   WindowRef window,
   const EventRecord *event
);
```

Parameters

window

The window in which the mouse-down event occurred.

event

A pointer to the `EventRecord` structure containing the mouse-down event that `IsWindowPathSelectClick` is to examine.

Return Value

A Boolean whose value is `true` f the mouse click should activate the window path pop-up menu; otherwise `false`.

Discussion
The Window Manager provides system support for your application to display window path pop-up menus, such as those used in Finder windows. When the user presses the Command key and clicks on the window's title, the window displays a pop-up menu containing a standard file system path, informing the user of the location of the document displayed in the window and allowing the user to open windows for folders along the path.

Because the window title includes both the proxy icon region and part of the drag region of the window, your application must be prepared to respond to a click in either region by displaying a window path pop-up menu. Therefore, when the `FindWindow` (page 1827) function returns either the `inDrag` or the `inProxyIcon` result code—you should pass the event to the `IsWindowPathSelectClick` function to determine whether the mouse-down event should activate the window path pop-up menu. If `IsWindowPathSelectClick` returns a value of `true`, your application should then call the function `WindowPathSelect` (page 1970) to display the menu.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

IsWindowPathSelectEvent

Determines whether a Carbon event describing a click on a window's title should cause a path selection menu to be displayed.

```
Boolean IsWindowPathSelectEvent (
   WindowRef window,
   EventRef inEvent
);
```

Parameters

window
The window to query.

inEvent
The event. In CarbonLib and in Mac OS X v10.2 and earlier, the function only returns `true` for `kEventClassMouse`/`kEventMouseDown` events. In Mac OS X v10.3 and later, this function returns `true` for any event that has suitable `kEventParamMouseLocation` and `kEventParamModifiers` parameters.

Return Value
A Boolean whose value is `true` if the click should cause a path selection menu to be displayed; otherwise, `false`. If this function returns `true`, the application should call `WindowPathSelect` (page 1970).

Discussion
Windows that have a proxy icon provided using an `FSRef` or alias can support a path selection menu, which displays the file system path to the object, one menu item per directory. Making a selection from this item automatically opens the corresponding object in the Finder.

Availability
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
MacWindows.h

IsWindowToolbarVisible

Determines whether a window's toolbar is visible.

```
Boolean IsWindowToolbarVisible (
   WindowRef inWindow
);
```

Parameters

inWindow
The window to query.

Return Value
A Boolean whose value is `true` if the window's toolbar is visible; otherwise, `false`.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
MacWindows.h

IsWindowUpdatePending

Determines whether a window update is pending.

```
Boolean IsWindowUpdatePending (
   WindowRef window
);
```

Parameters

window
The non-composited window to query.

Return Value
A Boolean whose value is `true` if an update is pending; otherwise, `false`.

Special Considerations

Modifying a composited window's update region does not affect the area of the window to be drawn. A composited window does not use its window update region to control drawing. Instead, a composited window determines what to draw by looking at the invalid regions of the views contained in the window.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
MacWindows.h

IsWindowVisible

Indicates whether the window frame is currently visible.

```
Boolean IsWindowVisible (
    WindowRef window
);
```

Parameters

window

The window to query.

Return Value

A Boolean value indicating whether or not the window is visible. If `true`, the window is visible. If `false`, the window is invisible.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

MoveWindow

Moves a window on the desktop.

```
void MoveWindow (
    WindowRef window,
    short hGlobal,
    short vGlobal,
    Boolean front
);
```

Parameters

window

The window that is to be moved on the desktop.

hGlobal

On input, the new location, in global coordinates, of the left edge of the window's port rectangle.

vGlobal

On input, the new location, in global coordinates, of the top edge of the window's port rectangle.

front

On input, a Boolean value specifying whether the window is to become the frontmost, active window. If the value of the front parameter is `false`, `MoveWindow` does not change its plane or status. If the value of the front parameter is `true` and the window isn't active, `MoveWindow` makes it active by calling the `SelectWindow` (page 1929) function.

Discussion

The `MoveWindow` function moves the specified window to the location specified by the `hGlobal` and `vGlobal` parameters, without changing the window's size. The upper-left corner of the window's port rectangle is placed at the point (`vGlobal`, `hGlobal`). The local coordinates of the upper-left corner are unaffected.

Your application doesn't normally call `MoveWindow`. When the user drags a window by dragging its title bar, you can call `DragWindow` (page 1824) which in turn calls `MoveWindow` when the user releases the mouse button.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
HID Calibrator

Declared In
`MacWindows.h`

MoveWindowStructure

Positions a window relative to its structure region.

```
OSStatus MoveWindowStructure (
   WindowRef window,
   short hGlobal,
   short vGlobal
);
```

Parameters

window
The window that is to be moved.

hGlobal
A value specifying the horizontal position, in global coordinates, to which the left edge of the window's structure region is to be moved.

vGlobal
A value specifying the vertical position, in global coordinates, to which the top edge of the window's structure region is to be moved.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
The `MoveWindowStructure` function moves the specified window, but does not change the window's size. When your application calls `MoveWindowStructure`, the positioning of the specified window is determined by the positioning of its structure region. This is in contrast to the `MoveWindow` function, where the positioning of the window's content region determines the positioning of the window. After moving the window, `MoveWindowStructure` displays the window in its new position.

Note that your application should not call the `MoveWindowStructure` function to position a window when the user drags the window by its drag region. When the user drags the window, your application should call the pre–Mac OS 8.5 Window Manager function `DragWindow`.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
QTCarbonShell

Declared In
MacWindows.h

NewCWindow

Creates a window with a specified list of characteristics. (Deprecated in Mac OS X v10.5. Use CreateNewWindow (page 1815) instead.)

```
WindowRef NewCWindow (
   void *wStorage,
   const Rect *boundsRect,
   ConstStr255Param title,
   Boolean visible,
   short procID,
   WindowRef behind,
   Boolean goAwayFlag,
   SRefCon refCon
);
```

Parameters

wStorage

On input, a pointer to the window structure. If you specify null as the value of wStorage, NewCWindow allocates the window structure as a nonrelocatable object in the application heap. You can reduce the chances of heap fragmentation by allocating memory from a block of memory reserved for this purpose by your application and passing a pointer to it in the wStorage parameter.

boundsRect

On input, a pointer to a rectangle, given in global coordinates, that specifies the window's initial size and location. This rectangle becomes the port rectangle of the window's graphics port. For the standard window types, the boundsRect field defines the content region of the window. The NewCWindow function places the origin of the local coordinate system at the upper-left corner of the port rectangle. NewCWindow calls the QuickDraw function OpenCPort to create the graphics port. The bitmap, pen pattern, and other characteristics of the window's graphics port are the same as the default values set by OpenCPort, except for the character font, which is set to the application font instead of the system font.

title

On input, a pascal string that specifies the window's title. If the title is too long to fit in the title bar, the title is truncated. To suppress the title in a window with a title bar, pass an empty string, not null, in the title parameter. null is an invalid value and may cause runtime errors.

visible

On input, a Boolean value indicating visibility status: `true` means that the Window Manager displays the window; `false` means it does not. If the value of the `visible` parameter is `true`, the Window Manager draws a new window as soon as the window exists. The Window Manager first calls the window definition function to draw the window frame. If the value of the `goAwayFlag` parameter is also `true` and the window is frontmost (that is, if the value of the behind parameter is (`WindowRef`)–`1L`), the Window Manager instructs the window definition function to draw a close box in the window frame. After drawing the frame, the Window Manager generates an update event to trigger your application's drawing of the content region.

When you create a window, you typically specify `false` as the value of the `visible` parameter. When you're ready to display the window, call `ShowWindow`.

procID

On input, the window's definition ID, a value that specifies both the window definition function and the variation code within that definition function. For a list of possible values, see "Pre-Appearance Window Definition IDs" (page 2048).

behind

On input, a pointer to the window that appears immediately in front of the new window on the desktop. To place a new window in front of all other windows on the desktop, specify a value of (`WindowRef`)–`1L`. When you place a new window in front of all others, `NewCWindow` removes highlighting from the previously active window, highlights the newly created window, and generates activate events that trigger your application's updating of both windows. Note that if you create an invisible window in front of all others on the desktop, the user sees no active window until you make the new window visible (or make another window active).

To place a new window behind all other windows, specify a value of `null`.

goAwayFlag

On input, a Boolean value that determines whether the window has a close box. If the value of `goAwayFlag` is `true` and the window type supports a close box, the Window Manager draws a close box in the title bar and recognizes mouse clicks in the close region; if the value of `goAwayFlag` is `false` or the window type does not support a close box, it does not.

refCon

On input, a window's reference constant, set and used only by your application.

Return Value
A pointer to the newly created window structure.

Discussion
The `NewCWindow` function creates a window as specified by its parameters, adds it to the window list, and returns a pointer to the newly created window structure. You can use the returned window pointer to refer to this window in most Window Manager functions. If `NewCWindow` is unable to read the window definition function from the resource file, it returns `null`.

The `NewCWindow` function looks for a `'wctb '` resource with the same resource ID as the `'WIND '` resource. If it finds one, it uses the window color information in the `'wctb '` resource for coloring the window content region.

If the window's definition function is not already in memory, `NewCWindow` reads it into memory and stores a handle to it in the window structure. It allocates space for the structure and content regions of the window.

Storing the characteristics of your windows as resources, especially window titles and window items, makes your application easier to localize.

The `NewCWindow` function creates a window in a color graphics port. Creating color windows whenever possible ensures that your windows appear on color monitors with whatever color options the user has selected. Your application typically sets up its own set of global variables reflecting the system setup during initialization by calling the `Gestalt` function.

Special Considerations

If you let the Window Manager create the window structure in your application's heap, call `DisposeWindow` to close the window and dispose of its window structure. If you allocated the memory for the window structure yourself and passed a pointer to `NewCWindow`, use the function `CloseWindow` to close the window and the appropriate disposal function (determined by how you have allocated memory) to dispose of the window structure.

Carbon Porting Notes

In Carbon, you cannot pass your own storage in to the `wStorage` parameter.

Carbon does not support custom window definitions stored in 'WDEF' resources. If you want to specify a custom window definition for `NewCWindow`, you must compile your definition function directly in your application and then register the function by calling `RegisterWindowDefinition`. When `NewCWindow` gets a `procID` value that doesn't recognize, it checks a special mapping table to find the pointer that's registered for the resource ID embedded in the `procID` parameter. It then calls that function to implement your window.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

NewWindow

Creates a window from a parameter list. (Deprecated in Mac OS X v10.5. Use `CreateNewWindow` (page 1815) instead.)

```
WindowRef NewWindow (
   void *wStorage,
   const Rect *boundsRect,
   ConstStr255Param title,
   Boolean visible,
   short theProc,
   WindowRef behind,
   Boolean goAwayFlag,
   SRefCon refCon
);
```

Parameters

wStorage

On input, a pointer to the window structure. If you specify `null` as the value of `wStorage`, `NewWindow` allocates the window structure as a nonrelocatable object in the heap. You can reduce the chances of heap fragmentation by allocating the storage from a block of memory reserved for this purpose by your application and passing a pointer to it in the `wStorage` parameter.

boundsRect

On input, a pointer to a rectangle, given in global coordinates, which specifies the window's initial size and location. This rectangle becomes the port rectangle of the window's graphics port. For the standard window types, `boundsRect` defines the content region of the window. The `NewWindow` function places the origin of the local coordinate system at the upper-left corner of the port rectangle. `NewWindow` calls the QuickDraw function `OpenPort` to create the graphics port. The bitmap, pen pattern, and other characteristics of the window's graphics port are the same as the default values set by `OpenPort`, except for the character font, which is set to the application font instead of the system font. The coordinates of the graphics port's port boundaries and visible region are changed along with its port rectangle.

title

On input, a pascal string that specifies the window's title. If the title is too long to fit in the title bar, the title is truncated. To suppress the title in a window with a title bar, pass an empty string, not `null`. `null` is an invalid value and may cause runtime errors.

visible

On input, a Boolean value indicating visibility status: `true` means that the Window Manager displays the window; `false` means it does not.

If the value of the `visible` parameter is `true`, the Window Manager draws a new window as soon as the window exists. The Window Manager first calls the window definition function to draw the window frame. If the value of the `goAwayFlag` parameter (described below) is also `true` and the window is frontmost (that is, if the value of the behind parameter is (`WindowRef`(–`1L`), the Window Manager instructs the window definition function to draw a close box in the window frame. After drawing the frame, the Window Manager generates an update event to trigger your application's drawing of the content region.

When you create a window, you typically specify `false` as the value of the `visible` parameter. When you're ready to display the window, you call the function `ShowWindow`.

theProc

On input, the window's definition ID, which specifies both the window definition function and the variation code for that definition function. For a list of possible values, see "`Pre-Appearance Window Definition IDs`" (page 2048).

behind

On input, a pointer to the window that appears immediately in front of the new window on the desktop. To place a new window in front of all other windows on the desktop, specify a value of (`WindowRef`)–`1L`. When you place a new window in front of all others, `NewWindow` removes highlighting from the previously active window, highlights the newly created window, and generates activate events that trigger your application's updating of both windows. Note that if you create an invisible window in front of all others on the desktop, the user sees no active window until you make the new window visible (or make another window active).

To place a new window behind all other windows, specify a value of `null`.

goAwayFlag

On input, a Boolean value that determines whether or not the window has a close box. If the value of `goAwayFlag` is `true` and the window type supports a close box, the Window Manager draws a close box in the title bar and recognizes mouse clicks in the close region; if the value of `goAwayFlag` is `false` or the window type does not support a close box, it does not. The `goAwayFlag` parameter is ignored for movable modal or modal dialog boxes which do not support a close box.

refCon

On input, the window's reference constant, set and used only by your application.

Discussion

The `NewWindow` function takes the same parameters as `NewCWindow` and returns a `WindowRef` as its function result. The only difference is that `NewWindow` creates a window in a monochrome graphics port, not a color graphics port. The window structure and graphics port structure that describe monochrome and color graphics ports are the same size and can be used interchangeably in most Window Manager functions.

The `NewWindow` function creates a window as specified by its parameters, adds it to the window list, and returns a pointer to the newly created window structure. You can use the returned window pointer to refer to this window in most Window Manager functions. If `NewWindow` is unable to read the window definition function from the resource file, it returns `null`.

If the window's definition function is not already in memory, `NewWindow` reads it into memory and stores a handle to it in the window structure. It allocates space for the structure and content regions of the window.

Storing the characteristics of your windows as resources, especially window titles and window items, makes your application easier to localize.

Special Considerations

If you let the Window Manager create the window structure in your application's heap, call `DisposeWindow` to close the window and dispose of its window structure. If you allocated the memory for the window structure yourself and passed a pointer to `NewWindow`, use the function `CloseWindow` to close the window and the appropriate disposal function (determined by how you have allocated memory) to dispose of the window structure.

Version Notes

The `NewWindow` function was originally implemented prior to Color QuickDraw. In Mac OS 8, you should call the Color QuickDraw function `NewCWindow` instead of `NewWindow` to programmatically create a window, because Color QuickDraw is always available in Mac OS 8. Use of this function is not recommended with Mac OS 8 and later. `NewWindow` is described here only for completeness.

Carbon Porting Notes

In Carbon, you cannot pass your own storage in to the `wStorage` parameter.

In Carbon, `NewWindow` is functionally equivalent to the `NewCWindow`, in that `NewWindow` returns a color window instead of a monochrome window.

Carbon does not support custom window definitions stored in 'WDEF' resources. If you want to specify a custom window definition for `NewWindow`, you must compile your definition function directly in your application and then register the function by calling `RegisterWindowDefinition`. When `NewWindow` gets a `procID` value that doesn't recognize, it checks a special mapping table to find the pointer that's registered for the resource ID embedded in the `procID` parameter. It then calls that function to implement your window.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

NewWindowDefUPP

Creates a new UPP for a window definition. (Deprecated in Mac OS X v10.5. The WDEF interface is deprecated; use a custom HIView to draw your custom window frame instead.)

```
WindowDefUPP NewWindowDefUPP (
   WindowDefProcPtr userRoutine
);
```

Parameters

userRoutine

For information, see `WindowDefProcPtr` (page 1973).

Return Value

For a description of the `WindowDefUPP` data type, see `WindowDefUPP` (page 1986).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

NewWindowPaintUPP

Creates a new UPP for a painting region. (Deprecated in Mac OS X v10.5. The window content painting interface is deprecated; use a `kEventControlDraw` Carbon event handler on a compositing window's content view instead.)

```
WindowPaintUPP NewWindowPaintUPP (
   WindowPaintProcPtr userRoutine
);
```

Parameters

userRoutine

For information, see `WindowPaintProcPtr` (page 1978).

Return Value

A UPP to the window paint function.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`MacWindows.h`

OpenDrawer

Opens a drawer.

```
OSStatus OpenDrawer (
   WindowRef inDrawerWindow,
   OptionBits inEdge,
   Boolean inAsync
);
```

Parameters

inDrawerWindow

The drawer window to open.

inEdge

The parent window edge on which to open the drawer. Pass `kWindowEdgeDefault` to use the drawer's preferred edge. If there is not enough room on the preferred edge, `OpenDrawer` tries the opposite edge. If there is insufficient room on both edges, the drawer will open on the preferred edge but may extend offscreen, under the Dock, or under the menu bar.

inAsync

Indicates whether to open the drawer synchronously (the drawer is entirely opened before the function call returns) or asynchronously (the drawer opens using an event loop timer after the function call returns). Specify `true` for asynchronous and `false` for synchronous.

Return Value

A result code. See "Window Manager Result Codes" (page 2051).

Availability

Available in Mac OS X v10.2 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

PaintBehind

Redraws a series of windows in the window list. (Deprecated in Mac OS X v10.5. Use `InvalWindowRect` (page 1898), `InvalWindowRgn` (page 1899), or `HIViewSetNeedsDisplay` (page 2485) to invalidate a portion of a window.)

```
void PaintBehind (
   WindowRef startWindow,
   RgnHandle clobberedRgn
);
```

Parameters

startWindow

On input, a pointer to the window's complete window structure.

clobberedRgn

On input, a handle to the region that has become invalid.

Discussion

The Window Manager calls the `PaintBehind` function; your application does not normally need to. `PaintBehind` calls `PaintOne` for `startWindow` and all the windows behind `startWindow`, clipped to `clobberedRgn`.

Special Considerations

Mac OS X applications never need to call this function.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

PaintOne

Redraws the invalid, exposed portions of one window on the desktop. (Deprecated in Mac OS X v10.5. Use `InvalWindowRect` (page 1898), `InvalWindowRgn` (page 1899), or `HIViewSetNeedsDisplay` (page 2485) to invalidate a portion of a window.)

```
void PaintOne (
   WindowRef window,
   RgnHandle clobberedRgn
);
```

Parameters

window
On input, a pointer to the window structure.

clobberedRgn
On input, a handle to the region that has become invalid.

Discussion
The Window Manager calls the `PaintOne` function; your application does not normally need to. `PaintOne` "paints" the invalid portion of the specified window and all windows above it. `PaintOne` draws as much of the window frame as is in `clobberedRgn` and, if some content region is exposed, erases the exposed area (paints it with the content pattern rather than the background pattern using `SetWinColor` or `SetThemeWindowBackground`) and adds it to the window's update region.

If the value of the `window` parameter is `null`, the window is the desktop, and `PaintOne` paints it with the desktop pattern.

Special Considerations

Mac OS X applications never need to call this function.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

PinRect

Returns the point within the specified rectangle that is closest to the specified point.

```
long PinRect (
   const Rect *theRect,
   Point thePt
);
```

Parameters

theRect

On input, a pointer to a rectangle in which the point is to be contained.

thePt

On input, a pointer to the point to be contained.

Return Value

A long integer that specifies a point within the specified rectangle that is as close as possible to the specified point. (The high-order word of the returned long integer is the vertical coordinate; the low-order word is the horizontal coordinate.

Discussion

`DragGrayRgn` uses the `PinRect` function to contain a point within a specified rectangle. If the specified point is within the rectangle, `PinRect` returns the point itself. If not, then

- if the horizontal position is to the left of the rectangle, `PinRect` returns the left edge as the horizontal coordinate
- if the horizontal position is to the right of the rectangle, `PinRect` returns the right edge minus 1 as the horizontal coordinate
- if the vertical position is above the rectangle, `PinRect` returns the top edge as the vertical coordinate
- if the vertical position is below the rectangle, `PinRect` returns the bottom edge minus 1 as the vertical coordinate

The 1 is subtracted when the point is below or to the right of the rectangle so that a pixel drawn at that point lies within the rectangle.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

RegisterWindowDefinition

Registers a binding between a resource ID and a window definition function.

```
OSStatus RegisterWindowDefinition (
   SInt16 inResID,
   const WindowDefSpec *inDefSpec
);
```

Parameters

inResID

A WDEF proc ID, as used in a `'WIND'` resource.

inDefSpec

Specifies the `WindowDefUPP` that should be used for windows with the given WDEF proc ID. Pass `NULL` to unregister a given WDEF proc ID.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
In the Mac OS 8.x Window Manager, a `'WIND'` resource can contain an embedded WDEF procID that is used by the Window Manager as the resource ID of a `'WDEF'` resource to lay out and draw the window. The `'WDEF'` resource is loaded by the Window Manager when you load the window with `GetNewWindow`. Since WDEFs can no longer be packaged as code resources on Carbon, the procID can no longer refer directly to a WDEF resource. However, using `RegisterWindowDefinition` you can instead specify a UniversalProcPtr pointing to code in your application code fragment.

To unregister a window definition, pass `NULL` in the `inDefSpec` parameter for a given WDEF proc ID.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

ReleaseQDContextForCollapsedWindowDockTile

Releases a port and other state created by `CreateQDContextForCollapsedWindowDockTile`. (Deprecated in Mac OS X v10.5. Use `HIWindowReleaseCollapsedDockTileContext` (page 1892) instead.)

```
OSStatus ReleaseQDContextForCollapsedWindowDockTile (
   WindowRef inWindow,
   CGrafPtr inContext
);
```

Parameters

inWindow

The window whose port is to be released.

inContext

The port that is to be released.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
You must call this function instead of calling `DisposePort` directly, or you may leak system resources.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Related Sample Code
QTCarbonShell

Declared In
MacWindows.h

ReleaseWindow

Decrements the retain count of a window, and destroys the window if the retain count falls to zero. (Deprecated in Mac OS X v10.5. Use `CFRelease` instead.)

```
OSStatus ReleaseWindow (
   WindowRef inWindow
);
```

Parameters

inWindow
The window whose retain count is to be decremented.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
This API is equivalent to `DisposeWindow` (page 1820). For consistency with Core Foundation and Carbon Events, it is preferred over `DisposeWindow`. Both APIs will continue to be supported.

In Mac OS X v10.2 and later, you can also call `CFRelease` to decrement the retain count of a window.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
MacWindows.h

ReleaseWindowGroup

Decrements the reference count for a window group.

```
OSStatus ReleaseWindowGroup (
   WindowGroupRef inGroup
);
```

Parameters

inGroup
The window group whose reference count is to be queried.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
MacWindows.h

RemoveWindowProperty

Removes a piece of data that is associated with a window.

```
OSStatus RemoveWindowProperty (
   WindowRef window,
   PropertyCreator propertyCreator,
   PropertyTag propertyTag
);
```

Parameters

window

The window whose data is to be removed.

propertyCreator

The creator code (typically, the application's signature) of the associated data to be removed.

propertyTag

The application-defined code identifying the associated data to be removed.

Return Value

A result code. See "Window Manager Result Codes" (page 2051).

Discussion

The data removed by the `RemoveWindowProperty` function must have been previously associated with the window with the function `SetWindowProperty` (page 1948).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

RemoveWindowProxy

Dissociates a file from a window.

```
OSStatus RemoveWindowProxy (
   WindowRef window
);
```

Parameters

window

The window for which you want to remove the associated file.

Return Value

A result code. See "Window Manager Result Codes" (page 2051).

Discussion

The `RemoveWindowProxy` function redraws the window title bar after removing all data associated with a given file, including the proxy icon, path menu, and file data.

Special Considerations

On Mac OS 8.x and Mac OS 9.x, you must save and restore the current graphics port—by calling the QuickDraw functions `GetPort` and `SetPort`—around each call to the `RemoveWindowProxy` function.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
QTCarbonShell

Declared In
`MacWindows.h`

RepositionWindow

Positions a window relative to another window or a display screen.

```
OSStatus RepositionWindow (
   WindowRef window,
   WindowRef parentWindow,
   WindowPositionMethod method
);
```

Parameters

`window`
The window whose position you want to set.

`parentWindow`
A pointer to the "parent" window, as defined by your application. In cases where the window positioning method does not require a parent window, you should set the `parentWindow` parameter to `NULL`.

`method`
A constant specifying the window positioning method to be used; see "Window Position Constants" (page 2017) for descriptions of possible values.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
Your application may call the `RepositionWindow` function to position any window, relative to another window or to a display screen. After positioning the window, `RepositionWindow` displays the window in its new position.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
BSDLLCTest

Declared In
`MacWindows.h`

ReshapeCustomWindow

Notifies the Window Manager that a custom window's shape has changed.

```
OSStatus ReshapeCustomWindow (
   WindowRef window
);
```

Parameters

window

The window whose shape has changed.

Return Value

A result code. See "Window Manager Result Codes" (page 2051).

Discussion

If the shape of a custom window needs to change dynamically, outside of the context of normal Window Manager operations, you must use `ReshapeCustomWindow` to notify the Window Manager so that it can recalculate the window regions and update the screen. The Window Manager queries your custom window definition for the new structure and content regions and updates the screen with the new window shape.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

ResizeWindow

Handles all user interaction while a window is being resized.

```
Boolean ResizeWindow (
   WindowRef inWindow,
   Point inStartPoint,
   const Rect *inSizeConstraints,
   Rect *outNewContentRect
);
```

Parameters

window

The window that is to be resized.

inStartPoint

Set the `Point` structure to contain the location, specified in global coordinates, where the mouse-down event occurred. Your application may retrieve this value from the `where` field of the `EventRecord` structure.

inSizeConstraints

Set the rectangle to specify the limits on the vertical and horizontal measurements of the content rectangle, in pixels. Although this parameter gives the address of a structure that is in the form of the `Rect` data type, the four numbers in the structure represent limits, not screen coordinates. The `top`, `left`, `bottom`, and `right` fields of the structure specify the minimum vertical measurement (`top`), the minimum horizontal measurement (`left`), the maximum vertical measurement (`bottom`), and the maximum horizontal measurement (`right`). The minimum dimensions should be large enough to allow a manageable rectangle; 64 pixels on a side is typical. The maximum dimensions can be no greater than 32,767. You can pass `NULL` to allow the user to resize the window to any size that is contained onscreen.

outNewContentRect
On input, a pointer to a structure of type `Rect`. On return, the structure contains the new dimensions of the window's content region, in global coordinates.

Return Value
`true` if the window was successfully resized; otherwise, `false`.

Discussion
The `ResizeWindow` function moves either an outline of the window's edges (Mac OS 9.x and earlier) or the actual window (Mac OS X) around the screen, following the user's cursor movements, and handles all user interaction until the mouse button is released. Unlike with the function `GrowWindow`, there is no need to follow this call with a call to the function `SizeWindow`, because once the mouse button is released, `ResizeWindow` resizes the window if the user has changed the window size. Once the resizing is complete, `ResizeWindow` draws the window in the new size.

Your application should call `ResizeWindow` instead of the earlier Window Manager functions `SizeWindow` and `GrowWindow`. The `ResizeWindow` function informs your application of the new window bounds, so that your application can respond to any changes in the window's position.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

RetainWindow

Increments the retain count of a window. (Deprecated in Mac OS X v10.5. Use `CFRetain` instead.)

```
OSStatus RetainWindow (
   WindowRef inWindow
);
```

Parameters
inWindow
The window whose retain count is to be incremented.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
This API is equivalent to `CloneWindow` (page 1808). For consistency with Core Foundation and Carbon Events, it is preferred over `CloneWindow`. Both APIs will continue to be supported.

In Mac OS X v10.2 and later, you can also call `CFRetain` to increment the retain count of a window.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

RetainWindowGroup

Increments the reference count for a window group.

```
OSStatus RetainWindowGroup (
   WindowGroupRef inGroup
);
```

Parameters

inGroup

The window group whose reference count is to be incremented. For information on this data type, see `WindowGroupRef` (page 1986).

Return Value

A result code. See "Window Manager Result Codes" (page 2051).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

ScrollWindowRect

Scroll any area of a window.

```
OSStatus ScrollWindowRect (
   WindowRef inWindow,
   const Rect *inScrollRect,
   SInt16 inHPixels,
   SInt16 inVPixels,
   ScrollWindowOptions inOptions,
   RgnHandle outExposedRgn
);
```

Parameters

inWindow

The window to scroll in.

inScrollRect

The rectangle to scroll, in local coordinates.

inHPixels

The number of pixels to scroll horizontally.

inVPixels

The number of pixels to scroll vertically.

inOptions

Options for the scroll. See "Window Scrolling Options" (page 2044) for a list of possible options.

outExposedRgn

A valid region handle for the area newly revealed by the scroll (can be `NULL`). If `NULL`, the exposed region is added to the window's update region, regardless of the state of the `kScrollWindowInvalidate` option. This prevents updates from being lost in multiple monitor situations where the Window Manager can't copy the entire region due to differing color tables.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
Scrolls pixels that are inside the specified region of the input window. No other pixels or the bits they represent are affected. The pixels are shifted a distance of `inHPixels` horizontally and `inVPixels` vertically. The positive directions are to the right and down. The pixels that are shifted out of the specified window are not displayed, and the bits they represent are not saved. The exposed empty area created by the scrolling is returned in the update region parameter and optionally added to the window's update region.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

ScrollWindowRegion

Scrolls a window's region.

```
OSStatus ScrollWindowRegion (
   WindowRef inWindow,
   RgnHandle inScrollRgn,
   SInt16 inHPixels,
   SInt16 inVPixels,
   ScrollWindowOptions inOptions,
   RgnHandle outExposedRgn
);
```

Parameters

inWindow
The window to scroll in.

inScrollRgn
The region to scroll, in local coordinates.

inHPixels
The number of pixels to scroll horizontally.

inVPixels
The number of pixels to scroll vertically.

inOptions
Options for the scroll. See "Window Scrolling Options" (page 2044) for a list of possible options.

outExposedRgn
A valid region handle for the area newly revealed by the scroll (can be `NULL`). If `NULL`, the exposed region is added to the window's update region, regardless of the state of the `kScrollWindowInvalidate` option. This prevents updates from being lost in multiple monitor situations where the Window Manager can't copy the entire region due to differing color tables.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
Scrolls pixels that are inside the specified region of the input window. No other pixels or the bits they represent are affected. The pixels are shifted a distance of `inHPixels` horizontally and `inVPixels` vertically. The positive directions are to the right and down. The pixels that are shifted out of the specified window are not displayed, and the bits they represent are not saved. The exposed empty area created by the scrolling is returned in the update region parameter and optionally added to the window's update region

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

SelectWindow

Makes a window active.

```
void SelectWindow (
   WindowRef window
);
```

Parameters
window
The window that is to be made active.

Discussion
The `SelectWindow` function removes highlighting from the previously active window, brings the specified window to the front, highlights it, and generates the activate events to deactivate the previously active window and activate the specified window. If the specified window is already active, `SelectWindow` has no effect. Call `SelectWindow` when the user presses the mouse button while the cursor is in the content region of an inactive window.

Even if the specified window is invisible, `SelectWindow` brings the window to the front, activates the window, and deactivates the previously active window. Note that in this case, no active window is visible on the screen. If you do select an invisible window, be sure to call `ShowWindow` (page 1959) immediately to make the window visible (and accessible to the user).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
HideMenuBar
QTCarbonShell

Declared In
`MacWindows.h`

SendBehind

Moves one window behind another.

```
void SendBehind (
   WindowRef window,
   WindowRef behindWindow
);
```

Parameters

window

The window to be moved.

behindWindow

On input, a pointer to the window that is to be in front of the moved window.

Discussion

The `SendBehind` function moves the window pointed to by the parameter `window` behind the window pointed to by the parameter `behindWindow`. If the move exposes previously obscured windows or parts of windows, `SendBehind` redraws the frames as necessary and generates the appropriate update events to have any newly exposed content areas redrawn.

If the value of `behindWindow` is `NULL`, `SendBehind` sends the window to be moved behind all other windows on the desktop. If the window to be moved is the active window, `SendBehind` removes its highlighting, highlights the newly exposed frontmost window, and generates the appropriate activate events.

Do not use `SendBehind` to deactivate a window after you've made a new window active with the function `SelectWindow` (page 1929). The `SelectWindow` function automatically deactivates the previously active window.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

SendWindowGroupBehind

Orders one window group behind another.

```
OSStatus SendWindowGroupBehind (
   WindowGroupRef inGroup,
   WindowGroupRef behindGroup
);
```

Parameters

inGroup

The window group.

behindGroup

The "behind" window group.

Return Value

A result code. See "Window Manager Result Codes" (page 2051).

Discussion

A window group can contain multiple window groups. You can use this function to order nested groups.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

SetDrawerOffsets

Sets the positioning offsets for the drawer with respect to its parent window.

```
OSStatus SetDrawerOffsets (
   WindowRef inDrawerWindow,
   CGFloat inLeadingOffset,
   CGFloat inTrailingOffset
);
```

Parameters

inDrawerWindow
The drawer window whose positioning offsets are to be set.

inLeadingOffset
The new leading offset, in pixels. Pass `kWindowOffsetUnchanged` if you don't want to change the leading offset.

inTrailingOffset
The new trailing offset, in pixels. Pass `kWindowOffsetUnchanged` if you don't want to change the trailing offset.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

SetDrawerParent

Sets the parent window for a drawer.

```
OSStatus SetDrawerParent (
   WindowRef inDrawerWindow,
   WindowRef inParent
);
```

Parameters

inDrawerWindow
The drawer window whose parent window is to be set.

inParent
The window that is to be set as the parent of the window specified by `inDrawerWindow`.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

SetDrawerPreferredEdge

Set the preferred window edge from which the drawer should appear.

```
OSStatus SetDrawerPreferredEdge (
   WindowRef inDrawerWindow,
   OptionBits inEdge
);
```

Parameters

inDrawerWindow
The drawer window whose preferred window edge is to be set.

inEdge
The preferred edge. See "Window Edge Constants" (page 2041) for a list of possible values.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

SetPortWindowPort

Sets the current graphics port to the window's port.

```
void SetPortWindowPort (
   WindowRef window
);
```

Parameters

window
The window whose graphics port is to be set.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
HID Explorer

QTCarbonShell

Declared In
MacWindows.h

SetThemeTextColorForWindow

Sets a text color that contrasts with a theme brush.

```
OSStatus SetThemeTextColorForWindow (
   WindowRef inWindow,
   Boolean inActive,
   SInt16 inDepth,
   Boolean inColorDev
);
```

Parameters

inWindow
The window whose text color is to be set.

inActive
A Boolean whose value is `true` to indicate an active state or `false` to indicate an inactive state.

inDepth
The bit depth of the window's port. In Mac OS X, this parameter is ignored and should always be set to `32`.

inColorDev
A Boolean whose value is `true` to indicate that the window's port is color or `false` to indicate that the port is black and white. In Mac OS X, this parameter is ignored and should always be set to `true`.

Return Value
A result code. See "Window Manager Result Codes" (page 2051) for a list of possible values.

Discussion
`SetThemeTextColorForWindow` sets a text color in the specified window's port that contrasts with the brush specified by `SetThemeWindowBackground` (page 1933) and also matches the `inActive` parameter.

Only a subset of the theme brushes have theme text colors. As of Mac OS 9 and Mac OS X v10.4 and later, the Alert, Dialog, Modeless Dialog, and Notification brushes have corresponding text colors. For any other brush, `SetThemeTextColorForWindow` returns `themeNoAppropriateBrushErr` and does not modify the text color.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
MacWindows.h

SetThemeWindowBackground

Sets a window's background theme.

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

Parameters

inWindow

The window whose background theme is to be set.

inBrush

The theme brush that determines how the window background is painted. For information on theme brushes, see the Appearance Manager documentation.

inUpdate

A Boolean whose value is `true` if you want the window to be redrawn immediately using the new background brush; otherwise, `false`.

Return Value
A result code. See "Window Manager Result Codes" (page 2051) for a list of possible values.

Availability
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
`MacWindows.h`

SetUserFocusWindow

Designates a window to receive user focus.

```
OSStatus SetUserFocusWindow (
    WindowRef inWindow
);
```

Return Value
A result code.

Discussion
You can use this function to assign user focus to a specified window. This tells the Carbon Event Manager to route events that should go to the user focus (for example, commands and keyboard events) to the specified window. This can be used, for example, to route keyboard events to a floating palette, since floating palettes do not normally receive user focus.

Setting focus automatically defocuses whatever element formerly had user focus. If the focus changes to a new window, the `kEventWindowFocusAcquired` Carbon event will be sent to the newly focused window, and the `kEventWindowFocusRelinquish` Carbon event will be sent to the previously focused window.

If you pass `kUserFocusAuto` in the `inWindow` parameter, the system picks the best candidate for user focus (typically, this will be the active window). If you temporarily change the focus to a special window, you should use this option to restore the focus rather than setting the focus to an explicit window.

Availability
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
MacWindows.h

SetWindowActivationScope

Sets a window's activation scope.

```
OSStatus SetWindowActivationScope (
   WindowRef inWindow,
   WindowActivationScope inScope
);
```

Parameters

inWindow
The window whose activation scope is to be set.

inScope
The new activation scope.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
MacWindows.h

SetWindowAlpha

Sets the window's alpha channel value.

```
OSStatus SetWindowAlpha (
   WindowRef inWindow,
   CGFloat inAlpha
);
```

Parameters

inWindow
The window whose alpha channel value is to be set.

inAlpha
The alpha value to set. This value can range from 0.0 (completely transparent) to 1.0 (opaque).

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
MacWindows.h

SetWindowAlternateTitle

Sets an alternate window title.

```
OSStatus SetWindowAlternateTitle (
   WindowRef inWindow,
   CFStringRef inTitle
);
```

Parameters

inWindow

The window for which to set the alternate title.

inTitle

The alternate title for the window. Passing `NULL` for this parameter will remove any alternate title that might be present.

Return Value

A result code. See "Window Manager Result Codes" (page 2051). An operating system status code.

Discussion

This API sets an alternate title for a window. The alternate title overrides what is displayed in the Window menu. If you do not set an alternate title, the normal window title is used. You would normally use this if the window title was not expressive enough to be used in the Window menu (or similar text-only situation).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

SetWindowBounds

Sets a window's size and position from the bounding rectangle of the specified window region.

```
OSStatus SetWindowBounds (
   WindowRef window,
   WindowRegionCode regionCode,
   const Rect *globalBounds
);
```

Parameters

window

The window whose bounds are to be set.

regionCode

A constant specifying the region to be used in determining the window's size and position. The only region codes allowed for this parameter are `kWindowStructureRgn` and `kWindowContentRgn`.

globalBounds

Set the rectangle to specify the dimensions and position, in global coordinates, of the window region specified in the `regionCode` parameter.

Return Value

A result code. See "Window Manager Result Codes" (page 2051).

Discussion
The `SetWindowBounds` function sets a window's size and position to that specified by the rectangle that your application passes in the `globalBounds` parameter. After doing so, `SetWindowBounds` redraws the window, if the window is visible.

When you call the `SetWindowBounds` function, your application specifies whether the window's content region or its structure region is more important in determining the window's ultimate size and position. This distinction can be important with versions of the Mac OS running the Appearance Manager, since the total dimensions of a window—and, therefore, its spatial relationship to the rest of the screen—may vary from appearance to appearance. In general, you should specify `kWindowStructureRgn` for the `regionCode` parameter if how the window as a whole relates to a given monitor is more important than the exact positioning of its content on the screen. On the other hand, if you specify `kWindowContentRgn` for the `regionCode` parameter because the positioning of your application's content is of greatest concern, then it is important to note that with some appearances some part of the window's structure region or "frame" may extend past the edge of a monitor and not be displayed.

See also the function `GetWindowBounds` (page 1843).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

See Also
`HIWindowSetBounds` (page 1893)

Declared In
`MacWindows.h`

SetWindowCancelButton

Specifies a Cancel button for a window.

```
OSStatus SetWindowCancelButton (
    WindowRef inWindow,
    ControlRef inControl
);
```

Parameters

inWindow
The window whose Cancel button you want to set.

inControl
The control to designate as the Cancel button.

Return Value
A result code.

Discussion
You can use this function to specify a control (normally a button) to be the Cancel button for a given window. This button would be considered to have been clicked if the user instead presses Command-period or the Escape key.

The standard window event handler looks for keystrokes that correspond to the cancel button and generates events of type `kEventControlHit` when it detects the correct key being pressed. This is similar to the way the Dialog Manager responds to cancel buttons, except that instead of returning an item index for which button is pressed, the Carbon Event Manager generates a control hit event.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

SetWindowClass

Sets the class of a window. (Deprecated in Mac OS X v10.5. Use `HIWindowChangeClass` (page 1876), `SetWindowGroup` (page 1941), or `HIWindowChangeAttributes` (page 1874) instead.)

```
OSStatus SetWindowClass (
   WindowRef inWindow,
   WindowClass inWindowClass
);
```

Parameters

window
The window whose class you want to set.

inClass
The class that is to be set. See "Window Class Constants" (page 1988) for a list of possible window classes.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
This function changes the class of a window. It also changes the window's z-order so that it is grouped with other windows of the same class. It does not change the visual appearance of the window.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

SetWindowContentColor

Sets the color to which a window's content region is redrawn.

```
OSStatus SetWindowContentColor (
   WindowRef window,
   const RGBColor *color
);
```

Parameters

window

The window whose content color is to be set.

color

Set this structure to specify the content color to be used.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
If your application uses the `SetWindowContentColor` function, the window's content region is redrawn to the color you specify, without affecting the value specified in the window's `CGrafPort` structure for the current background color.

See also the function `GetWindowContentColor` (page 1845).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

SetWindowContentPattern

Sets the pattern to which a window's content region is redrawn.

```
OSStatus SetWindowContentPattern (
   WindowRef window,
   PixPatHandle pixPat
);
```

Parameters

window

A pointer to the window whose content pattern is being set.

pixPat

Set this structure to specify the content pattern to be used. This handle is copied by the Window Manager, and your application continues to own the original. Therefore there may be higher RAM requirements for applications with numerous identically patterned windows.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
If your application uses the `SetWindowContentPattern` function, the window's content region is redrawn to the pattern you specify, without affecting the value specified in the window's `CGrafPort` structure for the current background pattern.

See also the function `GetWindowContentPattern` (page 1846).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

SetWindowDefaultButton

Specifies a default button for a window.

```
OSStatus SetWindowDefaultButton (
    WindowRef inWindow,
    ControlRef inControl
);
```

Parameters

inWindow
The window whose default button you want to set.

inControl
The control to designate as the default.

Return Value
A result code.

Discussion
You can use this function to specify a control (normally a button) to be the default for a given window. This button would be considered to have been clicked if the user instead presses the Return or Enter keys on the keyboard.

The standard window event handler looks for keystrokes that correspond to the default button and generates events of type `kEventControlHit` when it detects the correct key being pressed. This is similar to the way the Dialog Manager responds to default buttons, except that instead of returning an item index for which button is pressed, the Carbon Event Manager generates a control hit event.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
BSDLLCTest
CarbonCocoa_PictureCursor

Declared In
`MacWindows.h`

SetWindowDockTileMenu

Associates a pop-up menu with a window.

```
OSStatus SetWindowDockTileMenu (
   WindowRef inWindow,
   MenuRef inMenu
);
```

Parameters

inWindow

The window with which a pop-up menu is to be associated.

inMenu

The pop-up menu that is to be associated with the window specified by `inWindow`

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
You specify a dock tile menu if you want to be able to present special selections when the user activates the pop-up menu associated with the window's minimized dock tile.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Related Sample Code
QTCarbonShell

Declared In
`MacWindows.h`

SetWindowGroup

Assigns a window to a window group.

```
OSStatus SetWindowGroup (
   WindowRef inWindow,
   WindowGroupRef inNewGroup
);
```

Parameters

inWindow

The window that is to be assigned to a window group.

inNewGroup

The window group. For information on this data type, see `WindowGroupRef` (page 1986).

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

SetWindowGroupLevel

Sets the level of group in the window class hierarchy.

```
OSStatus SetWindowGroupLevel (
   WindowGroupRef inGroup,
   SInt32 inLevel
);
```

Parameters

inGroup

The window group.

inLevel

The new level for the windows in this group.

Return Value

A result code. See "Window Manager Result Codes" (page 2051).

Discussion

The window group's level is only used to set the level of its windows if the window group is a child of the root group. If there is another group in the group hierarchy between this group and the root group, this group's level is ignored.

In Mac OS X v10.4 and later, `SetWindowGroupLevel` sets all three window levels associated with a window group: active, inactive, and promoted. It then immediately determines if the active level needs to be promoted to a larger value, and if so, sets the promoted level to that value.

See the Core Graphics frameworks header `CGWindowLevel.h` for a listing of window levels.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

SetWindowGroupLevelOfType

Sets the window level of a window group.

```
OSStatus SetWindowGroupLevelOfType (
   WindowGroupRef inGroup,
   UInt32 inLevelType,
   CGWindowLevel inLevel
);
```

Parameters

inGroup

The window group whose Core Graphics window level is to be set.

inLevelType

The level type to set. Specify `kWindowGroupLevelActive` or `kWindowGroupLevelInactive`. For details, see "Window Group Level Constants" (page 2048).

inLevel

The new level that is to be set for the windows in this group.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
In Mac OS X v10.4 and later, multiple window levels may be associated with a window group: one level for when the application is active and another for when the application is inactive. The Window Manager automatically switches each group's level as the application becomes active or inactive. Call `SetWindowGroupLevelOfType` to set the active and inactive window level for a window group. The window group's level is only used to set the level of its windows if the window group is a child of the root group. If there is another group in the group hierarchy between this group and the root group, this group's level is ignored.

You can also use `SetWindowGroupLevelOfType` to set the promoted window level that is actually used for windows in the group. Doing so is not recommended, however, because the promoted window level is reset by the Window Manager whenever the window group hierarchy structure changes. Any changes that you make to the promoted level may, therefore, be overwritten. In general, you should only use `SetWindowGroupLevelOfType` to set the active and inactive window levels. When setting the active level of a group with the fixed-level window group attribute, this function also automatically sets the promoted level to the same value and updates the promoted level of any non-fixed-level groups above the group being modified.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

SetWindowGroupName

Assigns a name to a window group.

```
OSStatus SetWindowGroupName (
   WindowGroupRef inGroup,
   CFStringRef inName
);
```

Parameters

inGroup
The window group. For information on this data type, see `WindowGroupRef` (page 1986).

inName

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Carbon Porting Notes

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

SetWindowGroupOwner

Sets a window as the owner of a window group.

```
OSStatus SetWindowGroupOwner (
   WindowGroupRef inGroup,
   WindowRef inWindow
);
```

Parameters

inGroup

The window group that is to be set as the owner of the window group specified by `inWindow`.

inWindow

The window group whose owner is to be set.

Return Value

A result code. See "Window Manager Result Codes" (page 2051).

Discussion

This function is rarely needed and is known to be problematic, so calling this function is not recommended.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

SetWindowGroupParent

Sets a window group to be the parent of another window group.

```
OSStatus SetWindowGroupParent (
   WindowGroupRef inGroup,
   WindowGroupRef inNewGroup
);
```

Parameters

inGroup

The window group whose parent window group is to be set. The specified window group cannot contain any windows at the time of this call.

inNewGroup

The window group that is to be the parent of `inGroup`. For information on this data type, see `WindowGroupRef` (page 1986).

Return Value

A result code. See "Window Manager Result Codes" (page 2051).

Discussion

You can nest groups within each other using this function.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
MacWindows.h

SetWindowIdealUserState

Sets the size and position of a window in its user state.

```
OSStatus SetWindowIdealUserState (
    WindowRef inWindow,
    const Rect *inUserState
);
```

Parameters

inWindow
The window whose size and position in its user state is to be set.

inUserState
Set this rectangle to specify the new size and position of the window's user state, in global coordinates.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
Because the window definition function relies upon the `WStateData` structure, it is unaware of the ideal standard state, and this causes the user state data that it stores in the `WStateData` structure to be unreliable. While the Window Manager is reliably aware of the window's zoom state, it cannot record the current user state in the `WStateData` structure, because the window definition function can overwrite that data. Therefore, the function `SetWindowIdealUserState` maintains the window's user state independently of the `WStateData` structure. The `SetWindowIdealUserState` function gives your application access to the user state data maintained by `ZoomWindowIdeal` (page 1972). However, your application does not typically need to use this function; it is supplied for completeness.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

See Also
`HIWindowSetIdealUserState` (page 1894)

Declared In
MacWindows.h

SetWindowKind

Sets a window's window kind.

```
void SetWindowKind (
   WindowRef window,
   short kind
);
```

Parameters

window
The window whose window kind is to be set.

kind

An integer representing the window kind.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

SetWindowModality

Sets the modality of a window.

```
OSStatus SetWindowModality (
   WindowRef inWindow,
   WindowModality inModalKind,
   WindowRef inUnavailableWindow
);
```

Parameters

inWindow

The window whose modality to set.

inModalKind

The new modality for the window. See "Window Modality Options" (page 2016) for a list of possible options.

inUnavailableWindow

If the window is becoming document-modal, this parameter specifies the window to which the `inWindow` parameter is modal. The window specified by this parameter will not be available while `inWindow` is in window-modal state.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
The modality of a window is used by the Carbon event manager to automatically determine appropriate event handling.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

SetWindowModified

Sets the modification state of the specified window.

```
OSStatus SetWindowModified (
   WindowRef window,
   Boolean modified
);
```

Parameters

window

The window whose modification state is to be set.

modified

Pass `true` if the content of the window has been modified; otherwise, pass `false`.

Return Value

A result code. See "Window Manager Result Codes" (page 2051).

Discussion

Your application can use the functions `SetWindowModified` and `IsWindowModified` (page 1907) instead of maintaining its own separate record of the modification state of the content of a window. The modification state of a window is visually represented by a dot in the window's close box. If the dot is present, the window is modified; if the dot is absent, the window is not modified.

Your application should distinguish between the modification state of the window and the modification state of the window's contents, typically a document. The modification state of the window contents are what should affect `SetWindowModified`. For example, in the case of a word processing document, you call `SetWindowModified` (passing `true` in the `modified` parameter) whenever the user types new characters into the document. However, you do not call `SetWindowModified` when the user moves the window, because that change does not affect the document contents. If you need to track whether the window position has changed, you need to do this with your own flag.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

QTCarbonShell

Declared In

`MacWindows.h`

SetWindowPic

Sets a picture for the Window Manager to draw in a window's content region. (Deprecated in Mac OS X v10.5. Use an HIImageView object to draw a window's content instead.)

```
void SetWindowPic (
   WindowRef window,
   PicHandle pic
);
```

Parameters

window

The window whose picture is to be set.

pic

On input, a handle to the picture to be drawn in the window.

Discussion
The `SetWindowPic` function stores in a window structure a handle to a picture to be drawn in the window. When the window's content region must be updated, the Window Manager then draws the picture or part of the picture, as necessary, instead of generating an update event.

The `DisposeWindow` (page 1820) function assumes that any picture pointed to by the window structure field `windowPic` is stored as data, not as a resource. If your application uses a picture stored as a resource, you must release the memory it occupies by calling the Resource Manager's `ReleaseResource` function and set the `WindowPic` field to `NULL` before you close the window.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

SetWindowProperty

Associates an arbitrary piece of data with a window.

```
OSStatus SetWindowProperty (
   WindowRef window,
   PropertyCreator propertyCreator,
   PropertyTag propertyTag,
   ByteCount propertySize,
   const void *propertyBuffer
);
```

Parameters

window
The window with which data is to be associated.

propertyCreator
The creator code (typically, the application's signature) of the data to be associated.

propertyTag
A value identifying the data to be associated. You define the tag your application uses to identify the data; this code is not to be confused with the file type for the data.

propertySize
The size of the data to be associated.

propertyBuffer
A pointer to the data to be associated.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
Data set with the `SetWindowProperty` function may be obtained with the function `GetWindowProperty` (page 1859) and removed with the function `RemoveWindowProperty` (page 1923).

Availability
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code
HID Calibrator
QTMetaData

Declared In
`MacWindows.h`

SetWindowProxyAlias

Associates a file with a window.

```
OSStatus SetWindowProxyAlias (
   WindowRef inWindow,
   AliasHandle inAlias
);
```

Parameters

inWindow
The window with which the specified file is to be associated.

inAlias
A handle to a structure of type `AliasRecord` for the file to associate with the specified window. You can obtain an alias handle by calling the function `GetWindowProxyAlias` (page 1862). The Window Manager copies the alias data, so you can dispose of the alias after `SetWindowProxyAlias` returns.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
Your application should call the `SetWindowProxyAlias` function to establish a proxy icon for a given window. The creator code and file type of the file associated with a window determine the proxy icon that is displayed for the window.

Because the `SetWindowProxyAlias` function won't work without a saved file, you must establish the initial proxy icon for a new, untitled window with the function `SetWindowProxyCreatorAndType` (page 1950), which requires that you know the file type and creator code for the file, but does not require that the file have been saved.

Special Considerations

On Mac OS 8.x and Mac OS 9.x, you must save and restore the current graphics port—by calling the QuickDraw functions `GetPort` and `SetPort`—around each call to the `SetWindowProxyAlias` function.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

SetWindowProxyCreatorAndType

Sets the proxy icon for a window that lacks an associated file.

```
OSStatus SetWindowProxyCreatorAndType (
   WindowRef window,
   OSType fileCreator,
   OSType fileType,
   SInt16 vRefNum
);
```

Parameters

window

The window for which you want to set the proxy icon.

fileCreator

A code that is to be used, together with the `fileType` parameter, to determine the proxy icon. This typically is the creator code of the file that would be created, were the user to save the contents of the window.

fileType

A code that is to be used, together with the `fileCreator` parameter, to determine the proxy icon. This typically is the file type of the file that would be created, were the user to save the contents of the window.

vRefNum

A value identifying the volume containing the default desktop database to search for the icon associated with the file type and creator code specified in the `fileCreator` and `fileType` parameters. Pass `kOnSystemDisk` if the volume is unknown.

Return Value

A result code. See "Window Manager Result Codes" (page 2051).

Special Considerations

On Mac OS 8.x and Mac OS 9.x, you must save and restore the current graphics port—by calling the QuickDraw functions `GetPort` and `SetPort`—around each call to the `SetWindowProxyCreatorAndType` function.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

SetWindowProxyFSSpec

Associates a file with a window. (Deprecated in Mac OS X v10.5. Use `HIWindowSetProxyFSRef` (page 1894) instead.)

```
OSStatus SetWindowProxyFSSpec (
   WindowRef window,
   const FSSpec *inFile
);
```

Parameters

window

A pointer to the window with which the specified file is to be associated.

inFile

Set the file system specification structure to contain the data for the file to associate with the specified window. You can obtain an `FSSpec` structure by calling the function `GetWindowProxyFSSpec` (page 1862).

Return Value

A result code. See "Window Manager Result Codes" (page 2051).

Discussion

Your application should call the `SetWindowProxyFSSpec` function to establish a proxy icon for a given window. The creator code and file type of the file associated with a window determine the proxy icon that is displayed for the window.

Because the `SetWindowProxyFSSpec` function won't work without a saved file, you must establish the initial proxy icon for a new, untitled window with the function `SetWindowProxyCreatorAndType`, which requires that you know the file type and creator code for the file, but does not require that the file have been saved.

You must save and restore the current graphics port—by calling the QuickDraw functions `GetPort` and `SetPort`—around each call to the `SetWindowProxyFSSpec` function.

See also the function `SetWindowProxyAlias`.

Special Considerations

The use of file specifications is no longer recommended.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Related Sample Code

QTCarbonShell

Declared In

`MacWindows.h`

SetWindowProxyIcon

Overrides the default proxy icon for a window.

```
OSStatus SetWindowProxyIcon (
   WindowRef window,
   IconRef icon
);
```

Parameters

window

The window for which you want to set the proxy icon.

icon

An icon reference identifying the icon to be used for the window's proxy icon. If there is already a proxy icon in use of the type desired, an `IconRef` value may be obtained for that icon by calling the function `GetWindowProxyIcon` (page 1863). Otherwise, your application must call the Icon Services function `GetIconRefFromFile` to get a value of type `IconRef`. The Window Manager retains the `IconRef`, so you can release `icon` after `SetWindowProxyIcon` returns. See the Icon Services and Utilities documentation for a description of the `IconRef` data type.

Return Value

A result code. See "Window Manager Result Codes" (page 2051).

Discussion

If you want to override the proxy icon that the Window Manager displays by default for a given file, your application should call the `SetWindowProxyIcon` function.

More typically, when you do not want to override a window's default proxy icon, your application would call one of the following functions: `HIWindowSetProxyFSRef` (page 1894), `SetWindowProxyAlias` (page 1949), or `SetWindowProxyCreatorAndType` (page 1950).

On Mac OS 8.x and Mac OS 9.x, you must save and restore the current graphics port—by calling the QuickDraw functions `GetPort` and `SetPort`—around each call to the `SetWindowProxyIcon` function.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

SetWindowResizeLimits

Sets the maximum and minimum resize limits for windows.

```
OSStatus SetWindowResizeLimits (
   WindowRef inWindow,
   const HISize *inMinLimits,
   const HISize *inMaxLimits
);
```

Parameters

inWindow

The window whose maximum and minimum resize limits are to be set.

inMinLimits

The minimum limits. Pass `NULL` if you don't want to set this limit. For information on the `HISize` data type, see HIGeometry.h.

inMaxLimits

The maximum limits. Pass `NULL` if you don't want to set this limit. For information on the `HISize` data type, see HIGeometry.h.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Related Sample Code
HID Calibrator

Declared In
`MacWindows.h`

SetWindowStandardState

Sets a window's standard zoom rectangle.

```
void SetWindowStandardState (
   WindowRef window,
   const Rect *rect
);
```

Parameters

window

The window whose standard zoom rectangle is to be set.

rect

On input, a rectangle (in global coordinates) representing the window's standard zoom rectangle. A window's standard zoom rectangle is the window content bounds when the window is zoomed out to its largest extent.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

SetWindowTitleWithCFString

Sets the window title to the contents of a Core Foundation string.

```
OSStatus SetWindowTitleWithCFString (
   WindowRef inWindow,
   CFStringRef inString
);
```

Parameters

inWindow

The window whose title is to be set.

inString
The title to set.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
HID Calibrator
QTCarbonShell
QTMetaData

Declared In
`MacWindows.h`

SetWindowToolbar

Associates a toolbar with a window.

```
OSStatus SetWindowToolbar (
   WindowRef inWindow,
   HIToolbarRef inToolbar
);
```

Parameters
inWindow
The window with which the toolbar specified by `inToolbar` is to be associated.
inToolbar
The toolbar that is to be associated with the window specified by `inWindow`.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

SetWindowUserState

Sets a window's user zoom rectangle.

```
void SetWindowUserState (
   WindowRef window,
   const Rect *rect
);
```

Parameters

window

The window whose user zoom rectangle is to be set.

rect

On input, a pointer to a rectangle (in global coordinates) representing the user zoom rectangle that is to be set. The window's user zoom rectangle is the window content bounds when the window is zoomed back in.

Availability
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
`MacWindows.h`

SetWRefCon

Sets the `refCon` field of a window.

```
void SetWRefCon (
   WindowRef window,
   SRefCon data
);
```

Parameters

window

The window whose `refCon` field is to be set.

data

On input, the data to be placed in the `refCon` field.

Discussion
The `SetWRefCon` function places the specified data in the `refCon` field of the specified window structure. The `refCon` field is available to your application for any window-related data it needs to store.

Availability
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code
CarbonSketch

QTCarbonShell

Declared In
`MacWindows.h`

SetWTitle

Specifies a window's title. (Deprecated in Mac OS X v10.5. Use `SetWindowTitleWithCFString` (page 1953) instead.)

```
void SetWTitle (
   WindowRef window,
   ConstStr255Param title
);
```

Parameters

window

On input, a pointer to the window structure.

title

On input, a Pascal string containing the window title. To suppress the title in a window with a title bar, pass an empty string, not `null`.

Discussion

The `SetWTitle` function changes a window's title to the specified string, both in the window structure and on the screen, and redraws the window's frame as necessary. Always use `SetWTitle` instead of changing the title in a window structure.

When the user opens a previously saved document, you typically create a new (invisible) window with the title "untitled" and then call `SetWTitle` to give the window the document's name before displaying it. You also call `SetWTitle` when the user saves a document under a new name.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

ShowFloatingWindows

Shows an application's floating windows.

```
OSStatus ShowFloatingWindows (
   void
);
```

Return Value

A result code. For details, see "Window Manager Result Codes" (page 2051).

Discussion

When an application receives a suspend event, its floating windows are hidden automatically. When the application receives a resume event, the floating windows are made visible automatically. Call this function if you want to make your floating windows visible manually.

See also the function `HideFloatingWindows` (page 1871).

Special Considerations

The `ShowFloatingWindows` function operates only upon windows created with the `kFloatingWindowClass` constant; see "Window Class Constants" (page 1988) for more details on this constant.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

ShowHide

Sets a window's visibility.

```
void ShowHide (
   WindowRef window,
   Boolean showFlag
);
```

Parameters

window
On input, a pointer to the window structure.

showFlag
On input, a Boolean value that specifies its visibility: `true` makes a window visible; `false` makes it invisible.

Discussion
The `ShowHide` function sets a window's visibility to the status specified by the `showFlag` parameter. If the value of `showFlag` is `true`, `ShowHide` makes the window visible if it's not already visible and has no effect if it's already visible. If the value of `showFlag` is `false`, `ShowHide` makes the window invisible if it's not already invisible and has no effect if it's already invisible.

The `ShowHide` function never changes the highlighting or front-to-back ordering of windows and generates no activate events.

Use `ShowHide` only where you need to manually control window activation. Otherwise, use `ShowWindow` or `HideWindow` instead.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

ShowHideWindowToolbar

Shows or hides the toolbar.

```
OSStatus ShowHideWindowToolbar (
   WindowRef inWindow,
   Boolean inShow,
   Boolean inAnimate
);
```

Parameters

inWindow

The window whose toolbar is to be shown or hidden.

inShow

Pass `true` to show the toolbar, `false` otherwise.

inAnimate

Pass `true` to animate the transition, pass `false` for no animation.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

ShowSheetWindow

Shows a sheet window using appropriate visual effects.

```
OSStatus ShowSheetWindow (
   WindowRef inSheet,
   WindowRef inParentWindow
);
```

Parameters

inSheet

The window sheet that is to be shown.

inParentWindow

The parent of the window specified by `inSheet`.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
QTCarbonShell

Declared In
`MacWindows.h`

ShowWindow

Makes an invisible window visible.

```
void ShowWindow (
    WindowRef window
);
```

Parameters

window

The window that is to be made visible.

Discussion

The `ShowWindow` function makes an invisible window visible. If the specified window is already visible, `ShowWindow` has no effect. Your application typically creates a new window in an invisible state, performs any necessary setup of the content region, and then calls `ShowWindow` to make the window visible.

When you call `ShowWindow` to display a window that is invisible, the Window Manager draws the window frame and sends an event to request the application to draw the content region before the window becomes visible. For compositing windows, the Window Manager sends a `kEventControlDraw` event to each HIView in the window. For non-compositing windows, the Window Manager sends a `kEventWindowDrawContent` event. If a non-compositing window does not handle the `kEventWindowDrawContent`, the Window Manager shows the window and generates an update event to request your application to draw the content region.

If the newly visible window is the frontmost window, `ShowWindow` highlights it if it's not already highlighted and generates an activate event to make it active. The `ShowWindow` function does not activate a window that is not frontmost on the desktop.

Because `ShowWindow` does not change the front-to-back ordering of windows, it is not the inverse of `HideWindow` (page 1872). If you make the frontmost window invisible with `HideWindow`, and`HideWindow` has activated another window, you must call both `ShowWindow` and `SelectWindow` (page 1929) to bring the original window back to the front.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

BSDLLCTest

CarbonSketch

HID Config Save

HID Explorer

QTCarbonShell

Declared In

`MacWindows.h`

SizeWindow

Sets the size of a window.

```
void SizeWindow (
   WindowRef window,
   short w,
   short h,
   Boolean fUpdate
);
```

Parameters

window

The window whose size is to be set.

w

On input, the new window width, in pixels.

h

On input, the new window height, in pixels.

fUpdate

On input, a Boolean value that specifies whether any newly created area of the content region is to be accumulated into the update region (`true`) or not (`false`). You ordinarily pass a value of `true` to ensure that the area is updated. If you pass `false`, you're responsible for maintaining the update region yourself. For a composited window, this parameter is ignored, and any views that intersect the newly exposed area of the window are automatically invalidated. For more information on adding rectangles to and removing rectangles from the update region, see `InvalWindowRect` (page 1898) and `ValidWindowRect` (page 1969).

Discussion

The `SizeWindow` function changes the size of the window's graphics port rectangle to the dimensions specified by the w and h parameters, or does nothing if the values of `w` and `h` are both `0`. The Window Manager redraws the window in the new size, recentering the title and truncating it if necessary.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

QTCarbonShell

Declared In

`MacWindows.h`

StoreWindowIntoCollection

Stores data describing a window into a collection. (Deprecated in Mac OS X v10.5. Use `HIArchiveEncodeCFType` (page 2314) to encode a window to an archive instead.)

```
OSStatus StoreWindowIntoCollection (
   WindowRef window,
   Collection collection
);
```

Parameters

window

The window to be stored.

collection

A reference to the collection into which the window is to be stored. You pass a reference to a previously created collection, such as that returned by the Collection Manager function `NewCollection`.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
The `StoreWindowIntoCollection` function stores any window—including those not created by the Window Manager calls—into the specified collection. The Window Manager does not empty the collection beforehand, so any existing items in the collection remain.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

ToggleDrawer

Toggles the drawer state.

```
OSStatus ToggleDrawer (
   WindowRef inDrawerWindow
);
```

Parameters

inDrawerWindow

The drawer window whose drawer state is to be toggled.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
If the drawer is currently open or opening, this function closes the drawer. If the drawer is currently closed or closing, this function opens the drawer.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

TrackBox

Tracks clicks in the collapse, close, size, and zoom boxes, and clicks of the toolbar button.

```
Boolean TrackBox (
   WindowRef window,
   Point thePt,
   WindowPartCode partCode
);
```

Parameters

window

The window in which the mouse button was pressed.

thePt

On input, the location of the cursor when the mouse button was pressed. Your application receives this point from the `where` field in the event structure.

partCode

On input, the part code (`inZoomIn`, `inZoomOut`, `inGoAway`, `inGrow`, `inCollapseBox`, or `inToolbarButton`) returned by `FindWindow` (page 1827); see "Part Identifier Constants" (page 2043).

Return Value

A Boolean whose value is `true` if the specified part was clicked; otherwise, `false`. If `TrackBox` returns `true`, it also removes highlighting from the specified part.

Discussion

The `TrackBox` function tracks the cursor when the user presses the mouse button while the cursor is in the specified part, retaining control until the mouse button is released. While the button is down, `TrackBox` highlights the part while the cursor is in the part's region.

When the mouse button is released, `TrackBox` removes the highlighting from the part and returns `true` if the cursor is within the part's region and `false` if it is not.

If `TrackBox` returns `true` after tracking the close box, your application should close the window. If `TrackBox` returns `true` after tracking the grow box, your application should call `ResizeWindow` (page 1925). If `TrackBox` returns `true` after tracking the collapse box, your application should call `CollapseWindow` (page 1810). When tracking the toolbar button, your application should call `ShowHideWindowToolbar` (page 1957).

Your application calls the `TrackBox` function when it receives a result code of `inZoomIn` or `inZoomOut` from the function `FindWindow` (page 1827). If `TrackBox` returns `true`, your application calculates the standard state, if necessary, and calls the function `ZoomWindow` (page 1971) to zoom the window. If `TrackBox` returns `false`, your application does nothing.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

TrackGoAway

Tracks the cursor when the user presses the mouse button while the cursor is in the close box.

```
Boolean TrackGoAway (
   WindowRef window,
   Point thePt
);
```

Parameters

window

On input, the window in which the mouse-down event occurred.

thePt

On input, the location of the cursor at the time the mouse button was pressed. Your application receives this point from the `where` field of the event structure.

Return Value

When the mouse button is released, `TrackGoAway` removes the highlighting from the close box and returns `true` if the cursor is within the close region and `false` if it is not.

Discussion

The `TrackGoAway` function tracks cursor activity when the user presses the mouse button while the cursor is in the close box, retaining control until the user releases the mouse button. While the button is down, `TrackGoAway` highlights the close box as long as the cursor is in the close region.

Your application calls the `TrackGoAway` function when it receives a result code of `inGoAway` from `FindWindow` (page 1827). If `TrackGoAway` returns `true`, your application calls its own function for closing a window, which can call `DisposeWindow` (page 1820) to remove the window from the screen. If `TrackGoAway` returns `false`, your application does nothing.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

HideMenuBar

Declared In

`MacWindows.h`

TrackWindowProxyDrag

Handles all aspects of the drag process when the user drags a proxy icon.

```
OSStatus TrackWindowProxyDrag (
   WindowRef window,
   Point startPt
);
```

Parameters

window

The window whose proxy icon is being dragged.

startPt

Set the `Point` structure to contain the point, specified in global coordinates, where the mouse-down event that began the drag occurred. Your application may retrieve this value from the `where` field of the `event` structure.

Return Value
A result code. See "Window Manager Result Codes" (page 2051). If you receive the error `errUserWantsToDragWindow` (–5607), your application should respond by calling the Window Manager function `DragWindow`. Errors are also returned from the Drag Manager, including `userCanceledErr` (–128).

Discussion
If your application uses proxy icons to represent a type of object (currently, file system entities such as files, folders, and volumes) supported by the Window Manager, your application should call the `TrackWindowProxyDrag` function, and the Window Manager can handle all aspects of the drag process for you. If your application calls the `TrackWindowProxyDrag` function, it does not have to call the Drag Manager function `WaitMouseMoved` before starting to track the drag, as the Window Manager handles this automatically. However, if a proxy icon represents a type of data that the Window Manager does not support, or if you want to implement custom dragging behavior, your application should call the function `TrackWindowProxyFromExistingDrag` (page 1964).

Your application detects that a user is dragging one of its proxy icons when the function `FindWindow` (page 1827) returns the `inProxyIcon` result code; see "Window Part Code Constants" (page 2013) for more details.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

See Also
`HIWindowTrackProxyDrag` (page 1897)

Declared In
`MacWindows.h`

TrackWindowProxyFromExistingDrag

Allows custom handling of the drag process when the user drags a proxy icon.

```
OSStatus TrackWindowProxyFromExistingDrag (
   WindowRef window,
   Point startPt,
   DragRef drag,
   RgnHandle inDragOutlineRgn
);
```

Parameters

window
The window whose proxy icon is being dragged.

startPt
Set the `Point` structure to contain the point, specified in global coordinates, where the mouse-down event that began the drag occurred. Your application may retrieve this value from the `where` field of the `event` structure.

drag
A value that refers to the current drag process. Pass in the value produced in the `outNewDrag` parameter of the function `BeginWindowProxyDrag` (page 1802). If you are not using `BeginWindowProxyDrag` in conjunction with `TrackWindowProxyFromExistingDrag`, you must create the drag reference yourself with the Drag Manager function `NewDrag`.

inDragOutlineRgn

A region handle representing an outline of the icon being dragged. You may obtain a handle to this region from the `outDragOutlineRgn` parameter of `BeginWindowProxyDrag`. If you are not using `BeginWindowProxyDrag` in conjunction with `TrackWindowProxyFromExistingDrag`, you must create the region yourself.

Return Value
A result code. See "Window Manager Result Codes" (page 2051). Errors are also returned from the Drag Manager, including `userCanceledErr` (-128).

Discussion
Typically, if the proxy icon represents a type of object (currently, file system entities such as files, folders, and volumes) supported by the Window Manager, the Window Manager can handle all aspects of the drag process itself, and your application should call the function `TrackWindowProxyDrag` (page 1963). However, if the proxy icon represents a type of data that the Window Manager does not support, or if you want to implement custom dragging behavior, your application should call the `TrackWindowProxyFromExistingDrag` function.

The `TrackWindowProxyFromExistingDrag` function accepts an existing drag reference and adds file data if the window contains a file proxy. If your application uses `TrackWindowProxyFromExistingDrag`, you then have the choice of using this function in conjunction with the functions `BeginWindowProxyDrag` (page 1802) and `EndWindowProxyDrag` (page 1826) or simply calling `TrackWindowProxyFromExistingDrag` and handling all aspects of creating and disposing of the drag yourself.

Your application detects a drag when the function `FindWindow` (page 1827) returns the `inProxyIcon` result code; see "Window Part Code Constants" (page 2013) for more details.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

See Also
`HIWindowTrackProxyDrag` (page 1897)

Declared In
`MacWindows.h`

TransitionWindow

Shows, hides, moves, or resizes a window with appropriate animation and sound.

```
OSStatus TransitionWindow (
   WindowRef inWindow,
   WindowTransitionEffect inEffect,
   WindowTransitionAction inAction,
   const Rect *inRect
);
```

Parameters
inWindow

The window on which to act.

inEffect

The type of visual effect to use. `TransitionWindow` supports the Zoom, Slide, Fade, and Genie transition effects. The Slide effect is supported in Mac OS X and in CarbonLib 1.5 and later. The Fade and Genie effects are supported in Mac OS X v10.3 and later. See "Window Transition Effect Constants" (page 2027) for constants and descriptions of these effects.

inAction

The action to take. `TransitionWindow` supports the Show, Hide, Move, and Resize actions. The Move and Resize actions are supported in Mac OS X and in CarbonLib 1.5 and later. See "Window Transition Action Constants" (page 2026) for possible values.

inRect

A screen rect in global coordinates, or `NULL` for some transition actions. The interpretation of the rect is dependent on the transition action. For details, see the documentation for each action.

If you pass `kWindowShowTransitionAction` in the `action` parameter then, before calling `TransitionWindow`, set the rectangle to specify the dimensions and position, in global coordinates, of the area from which the zoom is to start. If you pass `NULL`, `TransitionWindow` uses the center of the display screen as the source rectangle.

If you pass `kWindowHideTransitionAction` in the `action` parameter then, before calling `TransitionWindow`, set the rectangle to specify the dimensions and position, in global coordinates, of the area at which the zoom is to end.

If you pass `NULL`, `TransitionWindow` uses the center of the display screen as the destination rectangle.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
The `TransitionWindow` function displays an animation of a window's transition between the open and closed states, such as that displayed by the Finder. `TransitionWindow` uses the rectangle specified in the `rect` parameter for one end of the animation (the source or the destination of the zoom, depending upon whether the window is being shown or hidden, respectively) and the window's current size and position for the other end of the animation. `TransitionWindow` also plays sounds appropriate to the current theme for the opening and closing actions.

Your application may use `TransitionWindow` instead of the functions `ShowWindow` and `HideWindow`. Like these pre–Mac OS 8.5 Window Manager functions, `TransitionWindow` generates the appropriate update and active events when it shows and hides windows.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
QTCarbonShell

Declared In
`MacWindows.h`

TransitionWindowAndParent

Shows or hides a window, potentially also moving a second window, with animation and sound.

```
OSStatus TransitionWindowAndParent (
   WindowRef inWindow,
   WindowRef inParentWindow,
   WindowTransitionEffect inEffect,
   WindowTransitionAction inAction,
   const Rect *inRect
);
```

Parameters

inWindow

The window that is to be shown or hidden.

inParentWindow

The window to which inWindow is related. For the Sheet effect, this parameter must be a valid window reference; for other effects, this parameter should be `NULL`.

inEffect

The type of visual effect to use. This function is most commonly used to perform the Sheet transition effect, but it also supports the Zoom, Slide, Fade, and Genie effects. See "Window Transition Effect Constants" (page 2027) for constants and descriptions of these effects.

inAction

The action to take on the window. The Show, Hide, Move, and Resize actions are supported. See "Window Transition Action Constants" (page 2026) for the appropriate constants.

inRect

A screen rect in global coordinates. The interpretation of the rect is dependent on the transition action; see the documentation for each action for details. May be `NULL` for some transition actions.

Return Value

A result code. See "Window Manager Result Codes" (page 2051).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

TransitionWindowWithOptions

Transitions a window from one state to another with appropriate animation and sound.

```
OSStatus TransitionWindowWithOptions (
   WindowRef inWindow,
   WindowTransitionEffect inEffect,
   WindowTransitionAction inAction,
   const HIRect *inBounds,
   Boolean inAsync,
   TransitionWindowOptions *inOptions
);
```

Parameters

inWindow

The window to transition.

inEffect

The type of visual effect to use. For possible values, see "Window Transition Effect Constants" (page 2027) for a description of this value.

inAction

The action to take. For possible values, see "Window Transition Action Constants" (page 2026).

inBounds

A screen rect in global coordinates. The interpretation of the rect is dependent on the transition action; see "Window Transition Action Constants" (page 2026) for the details of each action. This parameter may be `NULL` for the Show and Hide actions for the Zoom and Sheet effects. This parameter is ignored and must be `NULL` for the Show and Hide actions for the Fade effect.

inAsync

A Boolean whose value indicates whether the transition should run synchronously or asynchronously. If `inAsync` is `true`, this function returns immediately, and the transition runs using an event loop timer. You must run your event loop for the transition to occur. If `inAsync` is `false`, this function blocks until the transition completes. In either case, the `kEventWindowTransitionStarted` and `kEventWindowTransitionCompleted` Carbon events are sent to the transitioning window at the start and end of the transition.

inOptions

Extra information that are required for some transitions. This parameter may be `NULL` if the specified transition effect does not require extra information.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

UpdateCollapsedWindowDockTile

Updates the image of a window in the dock to the current contents of the window.

```
OSStatus UpdateCollapsedWindowDockTile (
   WindowRef inWindow
);
```

Parameters

inWindow

The window whose image is to be updated.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
Use this function for periodic updates, not for animation purposes. If you want animation, use `CreateQDContextForCollapsedWindowDockTile` (page 1816).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
MacWindows.h

ValidWindowRect

Removes a rectangle from a window's update region.

```
OSStatus ValidWindowRect (
   WindowRef window,
   const Rect *bounds
);
```

Parameters

window

The window containing the rectangle you want to remove from the update region.

bounds

Set this structure to specify, in local coordinates, a rectangle to be removed from the window's update region.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
The `ValidWindowRect` function informs the Window Manager that an area of a window no longer needs to be redrawn. The `ValidWindowRect` function is similar to the `ValidRect` function, but `ValidWindowRect` allows the window that it operates upon to be explicitly specified, instead of operating on the current graphics port, so `ValidWindowRect` does not require the graphics port to be set before its use.

See also the functions `InvalWindowRect` (page 1898) and `ValidWindowRgn` (page 1969).

Special Considerations

This function should not be used on composited windows. Modifying a composited window's update region does not affect the area of the window to be drawn. A composited window does not use its window update region to control drawing. Instead, a composited window determines what to draw by looking at the invalid regions of the views contained in the window.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
MacWindows.h

ValidWindowRgn

Removes a region from a window's update region.

```
OSStatus ValidWindowRgn (
   WindowRef window,
   RgnHandle region
);
```

Parameters

window

The window containing the region you want to remove from the update region.

region

Set this region to specify, in local coordinates, the area to be removed from the window's update region.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
The `ValidWindowRgn` function informs the Window Manager that an area of a window no longer needs to be redrawn. The `ValidWindowRgn` function is similar to the `ValidRgn` function, but `ValidWindowRgn` allows the window that it operates upon to be explicitly specified, instead of operating on the current graphics port, so `ValidWindowRgn` does not require the graphics port to be set before its use.

See also the functions `InvalWindowRgn` (page 1899) and `ValidWindowRect` (page 1969).

Special Considerations

This function should not be used on composited windows. Modifying a composited window's update region does not affect the area of the window to be drawn. A composited window does not use its window update region to control drawing. Instead, a composited window determines what to draw by looking at the invalid regions of the views contained in the window.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

WindowPathSelect

Displays a window path pop-up menu.

```
OSStatus WindowPathSelect (
   WindowRef window,
   MenuRef menu,
   SInt32 *outMenuResult
);
```

Parameters

window

The window for which a window path pop-up menu is to be displayed.

menu

The menu to be displayed for the specified window or `NULL`. If you pass `NULL` in this parameter, the Window Manager provides a default menu and sends a Reveal Object Apple event to the Finder if a menu item is selected. Note that in order to pass `NULL`, a file must currently be associated with the window [call `HIWindowSetProxyFSRef` (page 1894) to associate a file with the window]. If you pass a menu, this menu supersedes the default window path pop-up menu. There does not have to be a file currently associated with the window if you pass in your own menu.

outMenuResult

A pointer to a value that, on return, contains the menu and menu item the user chose. The high-order word of the value produced contains the menu ID, and the low-order word contains the item number of the menu item. If the user does not select a menu item, 0 is produced in the high-order word, and the low-order word is undefined. For file menus that have not been overridden, 0 is always produced in this parameter. Pass `NULL` in this parameter if you do not want this information.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
Your application should call the `WindowPathSelect` function when it detects a Command-click in the title of a window, that is, when the `IsWindowPathSelectClick` (page 1907) or `IsWindowPathSelectEvent` (page 1908) function returns a value of `true`. Calling `WindowPathSelect` causes the Window Manager to display a window path pop-up menu for your window.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

ZoomWindow

Zooms the window when the user has pressed and released the mouse button with the cursor in the zoom box.

```
void ZoomWindow (
   WindowRef window,
   WindowPartCode partCode,
   Boolean front
);
```

Parameters

window

The window that is to be zoomed.

partCode

On input, the part code (either `inZoomIn` or `inZoomOut`) returned by the `FindWindow` function; see "Part Identifier Constants" (page 2043).

front

On return, a Boolean value that determines whether the window is to be brought to the front. If the value of `front` is `true`, the window necessarily becomes the frontmost, active window. If the value of `front` is `false`, the window's position in the window list does not change. Note that if a window was active before it was zoomed, it remains active even if the value of `front` is `false`.

Discussion
The `ZoomWindow` function zooms a window in or out, depending on the value of the `partCode` parameter. Your application calls `ZoomWindow`, passing it the part code returned by `FindWindow` (page 1827), when it receives a result of `true` from `TrackBox`. The `ZoomWindow` function then changes the window's port rectangle to either the user state (if the part code is `inZoomIn`) or the standard state (if the part code is `inZoomOut`), as stored in the window state structure, described in the section `WStateData` (page 1987).

If the part code is `inZoomOut`, your application ordinarily calculates and sets the standard state before calling `ZoomWindow`.

For best results, call the QuickDraw function `EraseRect`, passing the window's graphics port as the port rectangle, before calling `ZoomWindow`.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

ZoomWindowIdeal

Zooms a window in accordance with human interface guidelines.

```
OSStatus ZoomWindowIdeal (
   WindowRef inWindow,
   WindowPartCode inPartCode,
   Point *ioIdealSize
);
```

Parameters

inWindow
The window that is to be zoomed.

inPartCode
A value specifying the direction of the zoom being requested. Your application passes in the relevant value (either the `inZoomIn` or the `inZoomOut` constant).

ioIdealSize
When you specify `inZoomIn` in the `partCode` parameter, you pass a pointer to the `Point` structure, but do not fill the structure with data. On return, the `Point` structure contains the new height and width of the window's content region, and `ZoomWindowIdeal` restores the previous user state.

When you specify `inZoomOut` in the `partCode` parameter, you pass the ideal height and width of the window's content region in the `Point` structure. On return, the `Point` structure contains the new height and width of the window's content region. `ZoomWindowIdeal` saves the user state of the window and zooms the window to its ideal size for the standard state.

Return Value
A result code. See "Window Manager Result Codes" (page 2051).

Discussion
Applications should use the `ZoomWindowIdeal` function instead of the older function `ZoomWindow`. When your application calls `ZoomWindowIdeal`, it automatically conforms to the human interface guidelines for determining a window's standard state.

The `ZoomWindowIdeal` function calculates a window's ideal standard state and updates a window's ideal user state independently of the `WStateData` structure. Previously, the window definition function was responsible for updating the user state, but because it relies upon the `WStateData` structure, the window definition function is unaware of the ideal standard state and can no longer track the window's zoom state reliably.

While the Window Manager is reliably aware of the window's zoom state, it cannot record the current user state in the `WStateData` structure, because the window definition function can overwrite that data. Therefore, if your application uses `ZoomWindowIdeal`, the `WStateData` structure is superseded, and the result of the `FindWindow` (page 1827) function should be ignored when determining whether a particular user click of the zoom box is a request to zoom in or out. When you adopt `ZoomWindowIdeal` and your application receives a result of either `inZoomIn` or `inZoomOut` from `FindWindow`, your application must use the function `IsWindowInStandardState` (page 1905) to determine the appropriate part code to pass in the `partCode` parameter.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
HID Calibrator

Declared In
`MacWindows.h`

Callbacks

WindowDefProcPtr

Defines a pointer to a window definition callback function. Your window definition callback function determines how a window looks and behaves.

```
typedef long (*WindowDefProcPtr) (
    short varCode,
    WindowRef window,
    short message,
    long param
);
```

If you name your function `MyWindowDefProc`, you would declare it like this:

```
long MyWindowDefProc (
    short varCode,
    WindowRef window,
    short message,
    long param
);
```

Parameters

varCode
The window's variation code.

window
A pointer to the window's window structure.

message
A value indicating the task to be performed. The `message` parameter contains one of the values defined in "Window Definition Message Constants" (page 2036). Other messages are reserved for internal use by the system. The list in the discussion section that follows explains each of these tasks in detail.

param
Data associated with the task specified by the `message` parameter. If the task requires no data, this parameter is ignored.

Return Value
Your window definition function should perform whatever task is specified by the `message` parameter and return a function result, if appropriate. If the task performed requires no result code, return 0.

Discussion
Various Window Manager functions call a window definition function whenever they need to perform a window-dependent action, such as drawing the window on the screen. If you want to define new, nonstandard windows for your application, you must write a window definition function, compile it in your application, and either use `RegisterWindowDefinition` (page 1920) to register it with the system or call `CreateCustomWindow` (page 1814) to create the custom window directly.

Note that Carbon does not allow you to store custom window definitions in a `'WDEF'` resource file as you could in pre-Carbon systems.

If you use `RegisterWindowDefinition` (page 1920), the Window Manager calls the Resource Manager to access your window definition function with the given resource ID; see "Pre-Appearance Window Definition IDs" in*Window Manager Legacy Reference* for a description of how window definition IDs are derived from resource IDs and variation codes.

The Resource Manager reads your window definition function into memory and returns a handle to it. The Window Manager stores this handle in the `windowDefProc` field of the window structure. Later, when it needs to perform an action on the window, the Window Manager calls the window definition function and passes it the variation code as a parameter.

Your window definition function is responsible for

- drawing the window frame
- reporting the region where mouse-down events occur
- calculating the window's structure region and content region
- drawing the size box
- resizing the window frame when the user drags the size box
- reporting the window's features or the location of a specific window region
- performing any customized initialization or disposal tasks

The Window Manager defines the data type `WindowDefUPP` to identify the universal procedure pointer for this application-defined function:

```
typedef UniversalProcPtr WindowDefUPP;
```

You typically use the `NewWindowDefProc` macro like this:

```
WindowDefUPP myWindowDefUPP;
myWindowDefUPP = NewWindowDefProc(MyWindow);
```

You typically use the `CallWindowDefProc` macro like this:

```
CallWindowDefProc (myWindowDefUPP, varCode, theWindow, message,  param);
```

The `message` parameter contains a value specifying the task to be performed by your window definition function. These tasks are:

- *Drawing the Window Frame*

  When the Window Manager passes `wDraw` in the `message` parameter, your window definition function should respond by drawing the window frame in the current graphics port (which is the Window Manager port). The window part code to be drawn will be passed in the `param` parameter of your window definition function. Your window definition function should perform the following steps:

  - Change the current port from the `WMgrPort` to the `WMgrCPort` to allow the system to draw in the full range of RGB colors.
  - Update the pen attributes, text attributes, and `bkPat` fields in the `WMgrCPort` to the values of the corresponding fields in the `WMgrPort`. The Window Manager automatically transfers the `vis` and `clip` regions.

  The parallelism of the `WMgrPort` and the `WMgrCPort` is maintained only by the window definition functions. All window definition functions that draw in the `WMgrPort` should follow the steps listed above even if the changed fields do not affect their operation.

  You must make certain checks to determine exactly how to draw the frame. If the value of the `visible` field in the window structure is `false`, you should do nothing; otherwise, you should examine the `param` parameter and the status flags in the window structure:

  - If the value of `param` is 0, draw the entire window frame (including the size box, if your window definition function incorporates the size box into the frame).
  - If the value of `param` is 0 and the `hilited` field in the window structure is `true`, highlight the frame to show that the window is active. If the value of the `goAwayFlag` field in the window structure is also `true`, draw a close box in the window frame. If the value of the `spareFlag` field in the window structure is also `true`, draw a zoom box in the window frame.
  - If the value of the `param` parameter is `wInGoAway`, redraw the window's close box, with or without highlighting as appropriate.
  - If the value of the `param` parameter is `wInZoom`, redraw the window's zoom box, with or without highlighting as appropriate.
  - If the value of the `param` parameter is `wInCollapseBox`, redraw the window's collapse box, with or without highlighting as appropriate.

  You can call `GetWindowWidgetHilite` (page 1868) to determine whether the close, zoom, or collapse box is currently highlighted. This function returns the part code of the currently highlighted part, or zero if no part is highlighted. You should draw the indicated part with highlighting, and draw other parts with no highlighting.

  The window frame typically, but not necessarily, includes the window's title, which should be displayed in the system font and system font size. The Window Manager port is already set to use the system font and system font size.

  Nothing drawn outside the window's structure region will be visible.

Your window definition function should return 0 as the function result for this message.

- *Reporting the Region of a Mouse-Down Event*

  When the Window Manager passes `wHit` in the `message` parameter, your window definition function should respond by reporting the region of the specified mouse-down event. The mouse location (in global coordinates) of the window frame will be passed into the `param` parameter of your window definition function. The vertical coordinate is in the high-order word of the parameter, and the horizontal coordinate is in the low-order word.

  In response to the `wHit` message, your window definition function should return one of the constants defined in "Window Definition Hit Test Result Code Constants" (page 2034).

  In Mac OS 9, return the constants `wInGrow`, `wInGoAway`, `wInZoomIn`, `wInZoomOut`, and `wInCollapseBox` only if the window is active—by convention, the size box, close box, zoom box, and collapse box aren't drawn if the window is inactive. In an inactive document window, for example, a mouse-down event in the part of the title bar that would contain the close box if the window were active is reported as `wInDrag`. In Mac OS X, your WDEF can return these part codes for inactive windows because these boxes are drawn even if the window is inactive.

- *Calculating Regions*

  When the Window Manager passes `wCalcRgns` in the `message` parameter, your window definition function should respond by calculating the window's structure and content regions based on the current graphics port's port rectangle. These regions, whose handles are in the `strucRgn` and `contRgn` fields of the window structure, are in global coordinates. The Window Manager requests this operation only if the window is visible. The mouse location (in global coordinates) of the window frame will be passed into the `param` parameter of your window definition function.

  Your window definition function should call `IsWindowCollapsed` (page 1904) to determine its collapse state. Then your window definition function can modify its structure and content regions as appropriate. Typically, a window's content region is empty in a collapsed state.

  When you calculate regions for your own type of window, do not alter the clip region or the visible region of the Window Manager port. The Window Manager and QuickDraw take care of this for you. Altering the Window Manager port's clip region or visible region may damage other windows.

  Your window definition function should return 0 as the function result for this message.

- *Performing Additional Window Initialization*

  When the Window Manager passes `wNew` in the `message` parameter, your window definition function should respond by performing any initialization that it may require. If the content region has an unusual shape, for example, you might allocate memory for the region and store the region handle in the `dataHandle` field of the window structure. The initialization function for a standard document window creates the `wStateData` structure for storing zooming data.

  Your window definition function should ignore the `param` parameter and return 0 as the function result for this message.

- *Performing Additional Window Disposal Actions*

  When the Window Manager passes `wDispose` in the `message` parameter, your window definition function should respond by performing any additional tasks necessary for disposing of a window. You might, for example, release memory that was allocated by the initialization function. The dispose function for a standard document window disposes of the `wStateData` structure.

  Your window definition function should ignore the `param` parameter and return 0 as the function result for this message.

- *Drawing the Window's Grow Image*

When the Window Manager passes `wGrow` in the `message` parameter, your window definition function should respond to being resized by drawing a dotted outline of the window in the current graphics port in the pen pattern and mode. (The pen pattern and mode are set up—as `gray` and `notPatXor`—to conform to Appearance-compliant human interface guidelines.)

A rectangle (in global coordinates) whose upper-left corner is aligned with the port rectangle of the window's graphics port is passed into the `param` parameter of your window definition function. Your grow image should be sized appropriately for the specified rectangle. As the user drags the mouse, the Window Manager sends repeated `wGrow` messages, so that you can change your grow image to match the changing mouse location.

Your window definition function should return 0 as the function result for this message.

- *Drawing the Size Box*

  When the Window Manager passes `wDrawGIcon` in the `message` parameter, your window definition function should respond by drawing the size box in the content region if the window is active. If the window is inactive, your window definition function should draw whatever is appropriate to show that the window cannot currently be sized. Your window definition function may also draw scroll bar delimiter lines. Your window definition function should ignore the `param` parameter.

  If the size box is located in the window frame, draw the size box in response to a `wDraw` message, not a `wDrawGIcon` message.

  Your window definition function should return 0 as the function result for this message.

- *Reporting Window Features*

  When the Window Manager passes `kWindowMsgGetFeatures` in the `message` parameter, your window definition function should respond by setting the `param` parameter to reflect the features that your window supports. The value passed back in the `param` parameter should be comprised of one or more of the values defined in "Window Feature Bits" (page 2011).

  Your window definition function should return 1 as the function result for this message.

- *Returning the Location of Window Regions*

  When the Window Manager passes `kWindowMsgGetRegion` in the `message` parameter, your window definition function should respond by returning the location (in global coordinates) of the specified window region. A pointer to a window region structure will be passed in the `param` parameter.

  The window region structure is a structure of type `GetWindowRegionRec` (page 1981). Your window definition function should return an operating system status (`OSStatus`) message as the function result for this message. The result code `errWindowRegionCodeInvalid` indicates that the window region passed in was not valid.

  Application-defined window definition functions are changed with Appearance Manager 1.0 to support collapse boxes and feature reporting.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

WindowPaintProcPtr

Defines a pointer to a custom content region painting function.

```
typedef OSStatus (*WindowPaintProcPtr) (
    GDHandle device,
    GrafPtr qdContext,
    WindowRef window,
    RgnHandle inClientPaintRgn,
    RgnHandle outSystemPaintRgn,
    void * refCon
);
```

If you name your function `MyWindowPaintProc`, you would declare it like this:

```
OSStatus MyWindowPaintProc (
    GDHandle device,
    GrafPtr qdContext,
    WindowRef window,
    RgnHandle inClientPaintRgn,
    RgnHandle outSystemPaintRgn,
    void * refCon
);
```

Parameters

device

The current graphics device (`GDevice`).

qdContext

The graphics port to draw into. Note that you should draw into this port, not the one associated with the window; the painting region `inClientPaintRgn` is defined relative to this port. The port may be an offscreen graphics world.

window

The window to paint in.

inClientPaintRgn

The region to be painted. Treat as a const. This region is clipped to the intersection of the current graphics device and the `clobberedRgn` parameter passed to `PaintBehind`.

outSystemPaintRgn

The region for the system to paint. Initially empty. If your paint procedure sets this region before returning, the Window Manager will erase this region using the system's window content paint function.

refCon

Application-defined data that you passed to `InstallWindowContentPaintProc` (page 1898).

Return Value

A result code. See "Window Manager Result Codes" (page 2051).

Discussion

Each window in the system contains a reference to a content paint proc. This proc is called to erase the window's content region during `PaintBehind` or `PaintOne` operations. The client application can override the system paint proc by calling `InstallWindowContentPaintProc` (page 1898). A window may only have one paint proc installed at any one time, and the paint proc cannot be retrieved by the client application.

If your content region painting callback returns any value other than noErr, `outSystemPaintRgn` is ignored and the entire area of `inClientPaintRgn` is painted using the system paint proc.

When a previously obscured portion of a window is exposed, the window manager will iterate over active displays and call the window's content paint proc once for each device intersecting the region.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacWindows.h`

Data Types

BasicWindowDescription

Describes basic window characteristics for use in a collection item.

```
struct BasicWindowDescription {
    UInt32 descriptionSize
    Rect windowContentRect
    Rect windowZoomRect
    UInt32 windowRefCon
    UInt32 windowStateFlags
    WindowPositionMethod windowPositionMethod
    UInt32 windowDefinitionVersion
    union {
        struct {
            SInt16 windowDefProc;
            Boolean windowHasCloseBox;
        } versionOne;
        struct {
            WindowClass windowClass;
            WindowAttributes windowAttributes;
        } versionTwo;
    } windowDefinition;
};
typedef struct BasicWindowDescription BasicWindowDescription;
```

Fields

`descriptionSize`
A value specifying the size of the entire `BasicWindowDescription` structure.

`windowContentRect`
A structure of type `Rect`, specifying the initial size and screen location of the window's content area.

`windowZoomRect`
Reserved.

`windowRefCon`
The window's reference value field, which is simply storage space available to your application for any purpose. The value contained in this field persists when the `'WIND'` resource is stored, so you should avoid saving pointers in this field, as they may become stale.

`windowStateFlags`

A 32-bit value whose bits you set to indicate the status of transient window states. See "Basic Window Description State Constant" (page 2010) for possible values.

`windowPositionMethod`

The specification last used in the function `RepositionWindow` (page 1924) to position this window, if any. See "Window Position Constants" (page 2017) for a description of possible values for this field.

`windowDefinitionVersion`

The version of the window definition used for the window. Set this field to a value of 1 if your application is creating a pre–Mac OS 8.5 window, that is, a window lacking class and attribute information. Set this field to a value of 2 if your application is creating a window using class and attribute information. See "Basic Window Description Version Constants" (page 2023) for descriptions of these values.

`windowDefinition`

A union of the `versionOne` and `versionTwo` structures. Your application must either specify the window's class and attributes, or it must supply a window definition ID and specify whether or not the window has a close box. See "Window Class Constants" (page 1988) and "Window Attributes" (page 1998) for descriptions of class and attribute values.

Discussion

The `BasicWindowDescription` structure is a default collection item for a resource of type `'wind'`. You use the `BasicWindowDescription` structure to describe the statically-sized base characteristics of a window.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MacWindows.h`

GetGrowImageRegionRec

Defines a region to be XOR'd during a window grow or resize operation.

```
struct GetGrowImageRegionRec {
    Rect growRect;
    RgnHandle growImageRegion;
};
typedef struct GetGrowImageRegionRec GetGrowImageRegionRec;
```

Fields

`growRect`

The window's new bounds in global coordinates.

`growImageRegion`

The grow image region.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MacWindows.h`

GetWindowRegionRec

Passed to window definitions in the `kWindowMsgGetRegion` message.

```
struct GetWindowRegionRec {
    RgnHandle winRgn;
    WindowRegionCode regionCode;
};
typedef struct GetWindowRegionRec GetWindowRegionRec;
typedef GetWindowRegionRec * GetWindowRegionPtr;
```

Fields

`winRgn`

A handle to a window region based on the value specified in the `regionCode` field. Modify this region.

`regionCode`

A value representing a given window region; see "Window Region Constants" (page 2021).

Special Considerations

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MacWindows.h`

HIWindowRef

Represents a window.

```
typedef WindowRef HIWindowRef;
```

Availability

Available in Mac OS X v10.3 and later.

Declared In

`MacWindows.h`

MeasureWindowTitleRec

Defines specifications of the window title.

```
struct MeasureWindowTitleRec {
    SInt16 fullTitleWidth;
    SInt16 titleTextWidth;
    Boolean isUnicodeTitle;
    Boolean unused;
};
typedef struct MeasureWindowTitleRec MeasureWindowTitleRec;
typedef MeasureWindowTitleRec * MeasureWindowTitleRecPtr;
```

Fields

`fullTitleWidth`

Your window definition function sets this field to a value specifying the total width in pixels of the window title text and any proxy icon that may be present, ignoring any compression or truncation that might be required when the title is actually drawn. That is, the specified width should be the ideal width that would be used if the window were sufficiently wide to draw the entire title along with a proxy icon. You should measure the title width using the current system font. If no proxy icon is present, this field should have the same value as the `titleTextWidth` field.

`titleTextWidth`

Your window definition function sets this field to a value specifying the width in pixels of the window title text, ignoring any compression or truncation that might be required when the title is actually drawn. That is, the specified width should be the ideal width that would be used if the window were sufficiently wide to draw the entire title. You should measure the title width using the current system font.

`isUnicodeTitle`

Your window definition function may ignore this field; it is reserved for future use.

`unused`

Your window definition function may ignore this field; it is reserved for future use.

Discussion

If you implement a custom window definition function, when the Window Manager passes the message `kWindowMsgMeasureTitle` in your window definition function's `message` parameter it also passes a pointer to a structure of type `MeasureWindowTitleRec` in the `param` parameter. Your window definition function is responsible for setting the contents of the `MeasureWindowTitleRec` structure to contain data describing the ideal title width.

See "Window Definition Message Constants" (page 2036) and "Window Feature Bits" (page 2011) for more details on the `kWindowMsgMeasureTitle` message and the corresponding `kWindowCanMeasureTitle` feature flag.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MacWindows.h`

PropertyCreator

Defines the creator of a window property.

```
typedef OSType PropertyCreator;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In
MacWindows.h

PropertyTag

Defines a window property tag.

```
typedef OSType PropertyTag;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
MacWindows.h

PicHandle

Defines a picture handle.

```
typedef PicPtr * PicPatHandle;
```

PixPatHandle

Pixel pattern handle.

```
typedef PixPatPtr * PixPatHandle;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
QuickdrawTypes.h

RGBColor

RGB color.

```
struct RGBColor {
    unsigned short red;
    unsigned short green;
    unsigned short blue;
};
typedef struct RGBColor;
typedef RGBColor * RGBColorPtr;
```

Fields
red
An unsigned short integer specifying the red value of the color.
green
An unsigned short integer specifying the green value of the color.

`blue`
An unsigned short integer specifying the red value of the color.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`IOMacOSTypes.h`

RgnHandle

Region handle.

```
typedef struct OpaqueRgnHandle * RgnHandle;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

SetupWindowProxyDragImageRec

Defines a window proxy drag image.

```
struct SetupWindowProxyDragImageRec {
    GWorldPtr imageGWorld;
    RgnHandle imageRgn;
    RgnHandle outlineRgn;
};
typedef struct SetupWindowProxyDragImageRec SetupWindowProxyDragImageRec;
```

Fields
`imageGWorld`
A pointer to the offscreen graphics world containing the drag image. The window definition function must allocate the offscreen graphics world, since the Window Manager has no way of knowing the appropriate size for the drag image. The Window Manager disposes of the offscreen graphics world.

`imageRgn`
A handle to a region containing the drag image. Only this portion of the offscreen graphics world referred to by the `imageGWorld` field is actually drawn. The Window Manager allocates and disposes of this region.

`outlineRgn`
A handle to a region containing an outline of the drag image, for use on monitors incapable of displaying the drag image itself. The Window Manager allocates and disposes of this region.

Discussion
If you implement a custom window definition function, when the function `TrackWindowProxyDrag` (page 1963) is called, the Window Manager passes the message `kWindowMsgSetupProxyDragImage` in your window definition function's `message` parameter and passes a pointer to a structure of type `SetupWindowProxyDragImageRec` in the `param` parameter. Your window definition function is responsible for setting the contents of the `SetupWindowProxyDragImageRec` structure to contain data describing the proxy icon's drag image.

See "Window Definition Message Constants" (page 2036)and "Window Feature Bits" (page 2011) for more details on the `kWindowMsgSetupProxyDragImage` message and the corresponding `kWindowCanSetupProxyDragImage` feature flag.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacWindows.h`

TransitionWindowOptions

Defines transition options used when calling `TransitionWindowWithOptions`.

```
struct TransitionWindowOptions {
    UInt32 version;
    EventTime duration;
    WindowRef window;
    void * userData;
};
typedef struct TransitionWindowOptions TransitionWindowOptions;
```

Fields

`version`
The structure version. You must put `0` in this field.

`duration`
The duration of the fade, in seconds. For use with the Sheet, Slide, Fade, and Genie transition effects; ignored for other effects. You may pass 0 to use the default duration. The effect is not guaranteed to last precisely this long, but should be a close approximation.

`window`
The parent window of the sheet; for use with `kWindowSheetTransitionEffect`.

`userData`
A value that is sent as the `kEventParamUserData` parameter for the `kEventWindowTransitionStarted` and `kEventWindowTransitionCompleted` events.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`MacWindows.h`

WindowDefSpec

Defines a window definition.

```
struct WindowDefSpec {
    WindowDefType defType
    union {
        WindowDefUPP defProc;
        void * classRef;
        short procID;
        void * rootView;
    } u;
};
typedef struct WindowDefSpec WindowDefSpec;
typedef WindowDefSpec * WindowDefSpecPtr;
```

Fields

`defType`

The window definition type. See "Window Definition Type Constants" (page 2033) for a list of possible values.

`u`

A pointer to the window definition, depending on the constant passed into the `defType` field.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacWindows.h`

WindowDefUPP

Defines a UPP to a specified window definition.

```
typedef WindowDefProcPtr WindowDefUPP;
```

Discussion
For more information, see `WindowDefProcPtr` (page 1973).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacWindows.h`

WindowGroupRef

Represents a window group.

```
typedef struct OpaqueWindowGroupRef * WindowGroupRef;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacWindows.h`

WindowPaintUPP

Defines a UPP to the specified region painting callback.

```
typedef WindowPaintProcPtr WindowPaintUPP;
```

Discussion
For more information, see `WindowPaintProcPtr` (page 1978).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacWindows.h`

WindowRef

An opaque type that represents a window.

```
typedef WindowPtr WindowRef;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`QuickdrawTypes.h`

WStateData

Stores the user state and the standard state of a window.

```
struct WStateData {
    Rect userState;
    Rect stdState;
};
typedef struct WStateData WStateData;
typedef WStateData * WStateDataPtr;
```

Fields

`userState`

A rectangle that describes the window size and location established by the user.

The Window Manager initializes the user state to the size and location of the window when it is first displayed, and then updates the `userState` field whenever the user resizes a window. Although the user state specifies both the size and location of the window, the Window Manager updates the window state data structure only when the user resizes a window—not when the user merely moves a window.

`stdState`

The rectangle describing the window size and location that your application considers the most convenient, considering the function and contents of the document, the screen space available, and the position of the window in its user state. If your application does not define a standard state, the Window Manager automatically sets the standard state to the entire gray region on the main screen, minus a three-pixel border on all sides. The user cannot change a window's standard state.

Discussion
When the Appearance Manager is available, you should not extend the window state data structure. Instead use the `refCon` field of the color window structure or extend the window record structure.

The zoom box allows the user to alternate quickly between two window positions and sizes: the user state and the standard state. The Window Manager stores the user state and your application stores the standard state in the window state data structure of type `WStateData`. The handle to this structure appears in the `dataHandle` field of the window structure.

The `ZoomWindow` (page 1971) function changes the size of a window according to the values in the window state data structure.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacWindows.h`

Constants

Window Class Constants

Constants that specify the standard window classes.

```
typedef UInt32 WindowClass;
enum {
    kAlertWindowClass = 1,
    kMovableAlertWindowClass = 2,
    kModalWindowClass = 3,
    kMovableModalWindowClass = 4,
    kFloatingWindowClass = 5,
    kDocumentWindowClass = 6,
    kUtilityWindowClass = 8,
    kHelpWindowClass = 10,
    kSheetWindowClass = 11,
    kToolbarWindowClass = 12,
    kPlainWindowClass = 13,
    kOverlayWindowClass = 14,
    kSheetAlertWindowClass = 15,
    kAltPlainWindowClass = 16,
    kDrawerWindowClass = 20,
    kAllWindowClasses = 0xFFFFFFFF
};
```

Constants

`kAlertWindowClass`

Identifies an alert box window. An alert window is used when the application needs the user's attention immediately. On Mac OS 9 and earlier, a visible alert window will prevent the user from switching to any other application. Use `kThemeBrushAlertBackgroundActive` to draw the background of alert windows. Alert windows are initially placed in the modal window group, given a modality of `kWindowModalityAppModal`, and given an activation scope of `kWindowActivationScopeAll`.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kMovableAlertWindowClass`

Identifies a movable alert box window. Generally, you should use this window class rather than `kAlertWindowClass`. Use `kThemeBrushAlertBackgroundActive` to draw the background of alert windows. Alert windows are initially placed in the modal window group, given a modality of `kWindowModalityAppModal`, and given an activation scope of `kWindowActivationScopeAll`.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kModalWindowClass`

Identifies a modal dialog box window. Use `kThemeBrushDialogBackgroundActive` to draw the background of modal dialog windows. Modal dialog windows are initially placed in the modal window group, given a modality of `kWindowModalityAppModal`, and given an activation scope of `kWindowActivationScopeAll`.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kMovableModalWindowClass`

Identifies a movable modal dialog box window. In Mac OS X and CarbonLib 1.3 and later, use `kThemeBrushMovableModalBackground` to draw the background of alert windows. Alert windows are initially placed in the modal window group, given a modality of `kWindowModalityAppModal`, and given an activation scope of `kWindowActivationScopeAll`.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kFloatingWindowClass`

Identifies a window that floats above all document windows. If your application assigns this constant to a window, the Window Manager ensures that the window has the proper floating behavior. Use `kThemeBrushUtilityWindowBackgroundActive` or `kThemeBrushDocumentWindowBackground` to draw the background of floating windows. Floating windows are initially placed in the floating window group, given a modality of `kWindowModalityNone`, and given an activation scope of `kWindowActivationScopeIndependent`.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kDocumentWindowClass`

Identifies a document window or modeless dialog box window. Use `kThemeBrushDocumentWindowBackground` or your own custom drawing to draw the background of a document window. Document windows are initially placed in the document window group, given a modality of `kWindowModalityNone`, and given an activation scope of `kWindowActivationScopeAll`.The Window Manager assigns this class to pre–Mac OS 8.5 Window Manager windows.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kUtilityWindowClass`

Identifies a utility window. A utility window is similar to a floating window, but it floats above the windows of all applications rather than just above the windows of the application that creates it. Use `kThemeBrushUtilityWindowBackgroundActive` or `kThemeBrushDocumentWindowBackground` to draw the background of utility windows. Utility windows are initially placed in the utility window group, given a modality of `kWindowModalityNone`, and given an activation scope of `kWindowActivationScopeIndependent`.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kHelpWindowClass`

Identifies a window used for help tags. It has no window frame. Typically you should use the Help Manager to display help tags, rather than creating a help tag window yourself. Help windows are initially placed in the help window group, given a modality of `kWindowModalityNone`, and given an activation scope of `kWindowActivationScopeNone`.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kSheetWindowClass`

Identifies a sheet. (Mac OS X only.) Use `kThemeBrushSheetBackgroundOpaque` to draw an opaque background for sheet windows, or `kThemeBrushSheetBackgroundTransparent` to draw a transparent background. Sheet windows are initially placed in the document window group, given a modality of `kWindowModalityNone`, and given an activation scope of `kWindowActivationScopeAll`.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kToolbarWindowClass`

Identifies a toolbar window, which is used to display a palette of controls. A toolbar window is similar to a floating window, and like a floating window, is layered above all application windows except for alert and modal windows, but is layered beneath floating windows. Toolbar windows are initially placed in the toolbar window group, given a modality of `kWindowModalityNone`, and given an activation scope of `kWindowActivationScopeNone`.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kPlainWindowClass`

Identifies a plain window, which has a single-pixel window frame. Plain windows are initially placed in the document window group, given a modality of `kWindowModalityNone`, and given an activation scope of `kWindowActivationScopeAll`.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kOverlayWindowClass`

Identifies an overlay window, which is a completely transparent window. Overlay windows are positioned by default above all other windows, but you can group an overlay window with any other window, at any z-order. Overlay windows are intended as a replacement for the pre-Carbon practice of drawing directly into the Window Manager port. By creating a full-screen overlay window and drawing into it, you can draw over any window in any application without disturbing the contents of the windows beneath your drawing. Overlay windows have a default handler for `kEventWindowPaint` that uses `CGContextClearRect` to clear the overlay window's alpha channel to zero. This ensures the initial transparency of the window. You can install your own `kEventWindowPaint` handler to do your own drawing; typically, you would call through to the default handler with `CallNextEventHandler` first, and then use `QDBeginCGContext` to create your own context for drawing. You can use either QuickDraw or Core Graphics to draw into an overlay window, but you must use Core Graphics to draw if you need any of your drawing to be non-opaque, since QuickDraw always sets the alpha channel of any pixels that it touches to 1.0. (QuickDraw is also deprecated in Mac OS X v10.4 and later.) You can also use the standard window event handler together with regular controls in an overlay window. When using the standard window event handler, you will probably want your `kEventWindowPaint` handler to return `eventNotHandledErr` (after calling the default handler with `CallNextEventHandler` first) so that after the Paint handler returns, the Window Manager will send a `kEventWindowDrawContent` event which the standard window event handler can respond to by drawing the controls in the window. Overlay windows are initially placed in the overlay window group, given a modality of `kWindowModalityNone`, and given an activation scope of `kWindowActivationScopeNone`.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kSheetAlertWindowClass`

Identifies an alert sheet. Use `kThemeBrushSheetBackgroundOpaque` to draw an opaque background for sheet alert windows, or `kThemeBrushSheetBackgroundTransparent` to draw a transparent background. Sheet alert windows are initially placed in the document window group, given a modality of `kWindowModalityNone`, and given an activation scope of `kWindowActivationScopeAll`.

Available in Mac OS X v10.1 and later.

Declared in `MacWindows.h`.

`kAltPlainWindowClass`
: Identifies an alternate plain window, which is similar to a plain window but has a solid black shadow on its right and bottom sides. It is rarely used in modern Mac OS applications. Alternate plain windows are initially placed in the document window group, given a modality of `kWindowModalityNone`, and given an activation scope of `kWindowActivationScopeAll`.

  Available in Mac OS X v10.1 and later.

  Declared in `MacWindows.h`.

`kDrawerWindowClass`
: Identifies a drawer. Use `kThemeBrushDrawerBackground` or `kThemeBrushDocumentWindowBackground` to draw the background of drawer windows. Drawer windows are initially placed in the document window group, given a modality of `kWindowModalityNone`, and given an activation scope of `kWindowActivationScopeAll`. Drawer windows should always be created using the compositing window attribute.

  Available in Mac OS X v10.2 and later.

  Declared in `MacWindows.h`.

`kAllWindowClasses`
: Specifier used to designate all window classes. Used with `GetFrontWindowOfClass`, `FindWindowOfClass`, and `GetNextWindowOfClass` to indicate that there should be no restriction on the class of the returned window. Also used with `GetWindowGroupOfClass` to get the root window group.

  Available in Mac OS X v10.0 and later.

  Declared in `MacWindows.h`.

Discussion
The `WindowClass` constants categorize windows into groups of like types. The grouping of windows facilitates the appropriate display (that is, both the look and the front-to-back ordering) and tracking of windows.

You can define a window's class using the function `CreateNewWindow` (page 1815) and obtain a window's class using the function `GetWindowClass` (page 1844). You can change the class of certain windows by calling `HIWindowChangeClass` (page 1876).

Window Attribute Identifiers

Constants that specify standard window attributes.

```
enum {
    kHIWindowBitCloseBox = 1,
    kHIWindowBitZoomBox = 2,
    kHIWindowBitCollapseBox = 4,
    kHIWindowBitResizable = 5,
    kHIWindowBitSideTitlebar = 6,
    kHIWindowBitToolbarButton = 7,
    kHIWindowBitUnifiedTitleAndToolbar = 8,
    kHIWindowBitTextured = 9,
    kHIWindowBitNoTitleBar = 10,
    kHIWindowBitTexturedSquareCorners = 11,
    kHIWindowBitNoTexturedContentSeparator = 12,
    kHIWindowBitDoesNotCycle = 16,
    kHIWindowBitNoUpdates = 17,
    kHIWindowBitNoActivates = 18,
    kHIWindowBitOpaqueForEvents = 19,
    kHIWindowBitCompositing = 20,
    kHIWindowBitFrameworkScaled = 21,
    kHIWindowBitNoShadow = 22,
    kHIWindowBitCanBeVisibleWithoutLogin = 23,
    kHIWindowBitAsyncDrag = 24,
    kHIWindowBitHideOnSuspend = 25,
    kHIWindowBitStandardHandler = 26,
    kHIWindowBitHideOnFullScreen = 27,
    kHIWindowBitInWindowMenu = 28,
    kHIWindowBitLiveResize = 29,
    kHIWindowBitIgnoreClicks = 30,
    kHIWindowBitNoConstrain = 32,
    kHIWindowBitDoesNotHide = 33,
    kHIWindowBitAutoViewDragTracking = 34,
    kHIWindowBitDoesNotShowBadgeInDock = 35
};
```

Constants

`kHIWindowBitCloseBox`

The window has a close box. This attribute is available for windows of class `kDocumentWindowClass` (page 1990), `kFloatingWindowClass` (page 1990), and `kUtilityWindowClass` (page 1990).

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitZoomBox`

The window has a zoom box. This attribute is available for windows of class `kDocumentWindowClass` (page 1990), `kFloatingWindowClass` (page 1990), and `kUtilityWindowClass` (page 1990). When this attribute is set on a window, both the `kWindowHorizontalZoomAttribute` and `kWindowVerticalZoomAttribute` bits are set automatically.

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitCollapseBox`
The window has a collapse box. This attribute is available for windows of class `kDocumentWindowClass` (page 1990), `kFloatingWindowClass` (page 1990), and `kUtilityWindowClass` (page 1990). For floating and utility window classes, this attribute must be added to the window after the window is created; it may not be added to the window at creation time.

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitResizable`
The window has a resize tab or box and is resizable. This attribute is available for windows of class `kDocumentWindowClass` (page 1990), `kMovableModalWindowClass` (page 1989), `kFloatingWindowClass` (page 1990), `kUtilityWindowClass` (page 1990), and `kSheetWindowClass` (page 1990).

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitSideTitlebar`
The window has a vertical title bar on the side of the window. This attribute is available for windows of the `kFloatingWindowClass` (page 1990) and `kUtilityWindowClass` (page 1990) class.

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitToolbarButton`
The window has a toolbar button. This oblong clear button shows and hides the toolbar. This attribute is available for windows of class `kDocumentWindowClass` (page 1990).

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitUnifiedTitleAndToolbar`
The window draws its window title and toolbar using a unified appearance that has no separator between the two areas. A window may not have both this attribute and the `kHIWindowBitTextured` attribute.

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitTextured`
The window uses the textured or brushed-metal appearance. Drawers can also be textured, but dynamically adjust their appearance based on their parent window's appearance; it is not necessary to specify this attribute for a textured drawer. This attribute is available for windows of class `kDocumentWindowClass` (page 1990) and `kFloatingWindowClass` (page 1990).

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitNoTitleBar`
The window's title bar can be hidden. This attribute is available for windows of class `kDocumentWindowClass` (page 1990), `kFloatingWindowClass` (page 1990), and `kUtilityWindowClass` (page 1990).

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitTexturedSquareCorners`
Indicates that a textured window should have square corners. By default, a textured window has round corners. When this attribute is set on a window, the window frame view automatically makes the grow box view opaque, and when this attribute is cleared, the window frame view automatically makes the grow box view transparent. You can change the grow box view transparency after modifying this attribute with the function `HIGrowBoxViewSetTransparent` (page 2416). Relevant only for textured windows; ignored in non-textured windows.

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitNoTexturedContentSeparator`
Indicates that no border is drawn between the toolbar and window content. This attribute is relevant only in textured windows; it is ignored in non-textured windows.

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitDoesNotCycle`
The window does not participate in window cycling invoked by Command-~ or keyboard shortcuts defined in the Keyboard & Mouse preference pane.

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitNoUpdates`
The window does not receive update events. This attribute is available for all windows.

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitNoActivates`
The window does not receive activate events. This attribute is available for all windows.

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitOpaqueForEvents`
The window receives mouse events even for areas of the window that are transparent (that is, have an alpha channel component of zero).

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitCompositing`
The window uses HIView-based compositing, which means that the entire window is comprised of HIViews, and can be treated thusly. This attribute must be specified at window creation; you may not add this attribute after the window has been created.

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitFrameworkScaled`
The window's content is scaled to match the display scale factor. This attribute can only be used when `kHIWindowBitCompositing` is also enabled. When this attribute is enabled, you may not draw with QuickDraw in the window. If this attribute is enabled and if the scale factor is something other than 1.0, the window's scale mode is `kHIWindowScaleModeFrameworkScaled`. If you specify this attribute and `kHIWindowBitApplicationScaled`, the `kHIWindowBitApplicationScaled` attribute is ignored. You may only specify this attribute at window creation time.

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitNoShadow`
The window has no shadow. This attribute is available for all windows, and is given automatically to windows of class `kOverlayWindowClass` (page 1991).

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitCanBeVisibleWithoutLogin`
The window can be made visible prior to user login. By default, in Mac OS X 10.5 and later no windows can be visible before a user logs into the system; this protects the user against certain types of malicious use of insecure applications. However, some software, such as input methods or other accessibility software, may need to deliberately make windows available prior to user login. Such software should add this window attribute to its windows.

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitAsyncDrag`
The window server drags the window automatically. Your application should not call `DragWindow` (page 1824) for this window because this function would fight with the Window Server for control. This attribute is ignored if the window is grouped with other windows in a window group that has the `kWindowGroupAttrMoveTogether` attribute.

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitHideOnSuspend`
The window is hidden automatically on suspend and shown on resume. This attribute available for all windows and is given automatically to windows of class `kFloatingWindowClass` (page 1990), `kHelpWindowClass` (page 1990), and `kToolbarWindowClass` (page 1991).

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitStandardHandler`
The window supports the standard window event handler. The standard event handler provides standard actions for common window events. See *Carbon Event Manager Programming Guide* for details. This attribute is available for all windows.

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitHideOnFullScreen`
The window is automatically hidden during full-screen mode (when the menubar is invisible) and shown afterwards. Available for all windows. This attribute is automatically given to windows of class `kUtilityWindowClass` (page 1990).

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitInWindowMenu`
The window title appears in the system-generated Window menu. This attribute is only available for windows of class `kDocumentWindowClass` (page 1990) and is automatically given to windows of that class.

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitLiveResize`
The window supports live resizing. This attribute is available for all windows.

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitIgnoreClicks`
The window never receives mouse events, even in areas that are opaque. Instead, clicks on the window are passed through to windows beneath it.

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitNoConstrain`
The window is not repositioned by the default `kEventWindowConstrain` handler in response to changes in monitor size, Dock position, and so on.

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitDoesNotHide`
The window does not hide when the application is hidden.

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitAutoViewDragTracking`
The window automatically installs Drag Manager callbacks to detect drag actions, and automatically sends HIView drag Carbon events. Setting this attribute is equivalent to calling the function `SetAutomaticControlDragTrackingEnabledForWindow` (page 643) (and calling that function will set this attribute).

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitDoesNotShowBadgeInDock`
Indicates that the Dock should not add a badge to this window's icon when the window is minimized to the Dock.

Discussion
In Mac OS X version 10.5 and later, you may use these constants to set or test the attributes of a window. For example, you may use them with the function `HIWindowCreate` (page 1880) to define the attributes of a new window, the function `HIWindowChangeAttributes` (page 1874) to change a window's attributes, and the function `HIWindowTestAttribute` (page 1896) to test whether a window has a specific attribute.

Window Attributes

Bit masks that specify standard window attributes. In Mac OS X v10.5 and later, you may use "Window Attribute Identifiers" (page 1992) instead.

```
typedef UInt32 WindowAttributes;
enum {
    kWindowNoAttributes = 0,
    kWindowCloseBoxAttribute = (1L << 0),
    kWindowHorizontalZoomAttribute = (1L << 1),
    kWindowVerticalZoomAttribute = (1L << 2),
    kWindowFullZoomAttribute = (kWindowVerticalZoomAttribute |
kWindowHorizontalZoomAttribute),
    kWindowCollapseBoxAttribute = (1L << 3),
    kWindowResizableAttribute = (1L << 4),
    kWindowSideTitlebarAttribute = (1L << 5),
    kWindowToolbarButtonAttribute = (1L << 6),
    kWindowUnifiedTitleAndToolbarAttribute = (1L << 7),
    kWindowMetalAttribute = (1L << 8),
    kWindowNoTitleBarAttribute = (1L << 9),
    kWindowTexturedSquareCornersAttribute = (1L << 10),
    kWindowMetalNoContentSeparatorAttribute = (1L << 11),
    kWindowDoesNotCycleAttribute = (1L << 15),
    kWindowNoUpdatesAttribute = (1L << 16),
    kWindowNoActivatesAttribute = (1L << 17),
    kWindowOpaqueForEventsAttribute = (1L << 18),
    kWindowCompositingAttribute = (1L << 19),
    kWindowFrameworkScaledAttribute = (1L << 20),
    kWindowNoShadowAttribute = (1L << 21),
    kWindowCanBeVisibleWithoutLoginAttribute = (1L << 22),
    kWindowAsyncDragAttribute = (1L << 23),
    kWindowHideOnSuspendAttribute = (1L << 24),
    kWindowStandardHandlerAttribute = (1L << 25),
    kWindowHideOnFullScreenAttribute = (1L << 26),
    kWindowInWindowMenuAttribute = (1L << 27),
    kWindowLiveResizeAttribute = (1L << 28),
    kWindowIgnoreClicksAttribute = (1L << 29),
    kWindowNoConstrainAttribute = (1L << 31),
    kWindowStandardDocumentAttributes = (kWindowCloseBoxAttribute |
kWindowFullZoomAttribute | kWindowCollapseBoxAttribute | kWindowResizableAttribute),
    kWindowStandardFloatingAttributes = (kWindowCloseBoxAttribute |
kWindowCollapseBoxAttribute)
};
```

Constants

`kWindowNoAttributes`

If no bits are set, the window has none of the standard attributes.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowCloseBoxAttribute`

If the bit specified by this mask is set, the window has a close box. See `kHIWindowBitCloseBox` (page 1993).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowHorizontalZoomAttribute`
If the bit specified by this mask is set, the window changes width when zooming. See `kHIWindowBitZoomBox` (page 1993).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowVerticalZoomAttribute`
If the bit specified by this mask is set, the window changes height when zooming. See `kHIWindowBitZoomBox` (page 1993).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowFullZoomAttribute`
If the bits specified by this mask are set, the window changes both width and height when zooming. See `kHIWindowBitZoomBox` (page 1993).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowCollapseBoxAttribute`
If the bit specified by this mask is set, the window has a collapse box. See `kHIWindowBitCollapseBox` (page 1994).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowResizableAttribute`
If the bit specified by this mask is set, the window has a resize tab or box and is resizable. See `kHIWindowBitResizable` (page 1994).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowSideTitlebarAttribute`
If the bit specified by this mask is set, the window has a side title bar. See `kHIWindowBitSideTitlebar` (page 1994).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowToolbarButtonAttribute`
If the bit specified by this mask is set, the window has a toolbar button. See `kHIWindowBitToolbarButton` (page 1994).

Available in Mac OS X v10.1 and later.

Declared in `MacWindows.h`.

`kWindowUnifiedTitleAndToolbarAttribute`
If the bit specified by this mask is set, the window draws its window title and toolbar using a unified appearance that has no separator between the two areas. A window may not have both this attribute and the `kWindowMetalAttribute` attribute.

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kWindowMetalAttribute`
If the bit specified by this mask is set, the window has a textured or brushed-metal appearance. See `kHIWindowBitTextured` (page 1994).

Available in Mac OS X v10.2 and later.

Declared in `MacWindows.h`.

`kWindowNoTitleBarAttribute`
If the bit specified by this mask is set, the window's title bar can be hidden. See `kHIWindowBitNoTitleBar` (page 1994).

Available in Mac OS X v10.4 and later.

Declared in `MacWindows.h`.

`kWindowTexturedSquareCornersAttribute`
See `kHIWindowBitTexturedSquareCorners` (page 1995).

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kWindowMetalNoContentSeparatorAttribute`
If the bit specified by this mask is set, no border is drawn between the toolbar and window content. See `kHIWindowBitNoTexturedContentSeparator` (page 1995).

Available in Mac OS X v10.4 and later.

Declared in `MacWindows.h`.

`kWindowDoesNotCycleAttribute`
If the bit specified by this mask is set, the window does not participate in window cycling. See `kHIWindowBitDoesNotCycle` (page 1995).

Available in Mac OS X v10.3 and later.

Declared in `MacWindows.h`.

`kWindowNoUpdatesAttribute`
If the bit specified by this mask is set, the window does not receive update events. See `kHIWindowBitNoUpdates` (page 1995).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowNoActivatesAttribute`
If the bit specified by this mask is set, the window does not receive activate events. See `kHIWindowBitNoActivates` (page 1995).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowOpaqueForEventsAttribute`
If the bit specified by this mask is set, the window receives mouse events even for areas of the window that are transparent. See `kHIWindowBitOpaqueForEvents` (page 1995).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowCompositingAttribute`
If the bit specified by this mask is set, the window uses HIView-based compositing. See `kHIWindowBitCompositing` (page 1995).

Available in Mac OS X v10.2 and later.

Declared in `MacWindows.h`.

`kWindowFrameworkScaledAttribute`
If the bit specified by this mask is set, this window's content is scaled to match the display scale factor. See `kHIWindowBitFrameworkScaled` (page 1996).

Available in Mac OS X v10.4 and later.

Declared in `MacWindows.h`.

`kWindowNoShadowAttribute`
If the bit specified by this mask is set, the window has no shadow. See `kHIWindowBitNoShadow` (page 1996).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowCanBeVisibleWithoutLoginAttribute`
If the bit specified by this mask is set, the window can be made visible prior to user login. See `kHIWindowBitCanBeVisibleWithoutLogin` (page 1996).

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kWindowAsyncDragAttribute`
If the bit specified by this mask is set, the window server drags the window automatically. See `kHIWindowBitAsyncDrag` (page 1996).

Available in Mac OS X v10.3 and later.

Declared in `MacWindows.h`.

`kWindowHideOnSuspendAttribute`
If the bit specified by this mask is set, the window is hidden automatically on suspend and shown on resume. See `kHIWindowBitHideOnSuspend` (page 1996).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowStandardHandlerAttribute`
If the bit specified by this mask is set, the window supports the standard window event handler. See `kHIWindowBitStandardHandler` (page 1996).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowHideOnFullScreenAttribute`
If the bit specified by this mask is set, the window is automatically hidden during fullscreen mode (when the menubar is invisible) and shown afterwards. See `kHIWindowBitHideOnFullScreen` (page 1997).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowInWindowMenuAttribute`
If the bit specified by this mask is set, the window title appears in the system-generated Window menu. See `kHIWindowBitInWindowMenu` (page 1997).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowLiveResizeAttribute`
If the bit specified by this mask is set, the window supports live resizing. See `kHIWindowBitLiveResize` (page 1997).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowIgnoreClicksAttribute`
If the bit specified by this mask is set, the window never receives mouse events, even in areas that are opaque. See `kHIWindowBitIgnoreClicks` (page 1997).

Available in Mac OS X v10.2 and later.

Declared in `MacWindows.h`.

`kWindowNoConstrainAttribute`
If the bit specified by this mask is set, the window is not repositioned by the default `kEventWindowConstrain` handler. See `kHIWindowBitNoConstrain` (page 1997).

Available in Mac OS X v10.1 and later.

Declared in `MacWindows.h`.

`kWindowStandardDocumentAttributes`
If the bits specified by this mask are set, the window has the attributes of a standard document window—that is, a close box, full zoom box, collapse box, and size box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowStandardFloatingAttributes`
If the bits specified by this mask are set, the window has the attributes of a standard floating window—that is, a close box and collapse box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

Discussion
The `WindowAttributes` enumeration defines masks your application can use to set or test the bits in a window attributes parameter. You can use these masks with the function `CreateNewWindow` (page 1815) to define a window's attributes, and with the function `ChangeWindowAttributes` (page 1805) to change a window's attributes. You can also use these masks to test the attributes parameter produced by the function `GetWindowAttributes` (page 1842), thereby obtaining a window's attributes.

User Focus Auto-Select Constant

Defines a constant that tells the system to pick the best user focus window.

```
#define kUserFocusAuto  ((WindowRef)(-1))
```

Constants
`kUserFocusAuto`
Pass this constant to the function `SetUserFocusWindow` (page 1934) to have the system choose the most appropriate window for user focus.

Available in Mac OS X v10.2 and later.

Declared in `MacWindows.h`.

Appearance-Compliant Window Resource IDs

Define window resources for Appearance-compliant applications.

```
enum {
    kWindowDocumentDefProcResID = 64,
    kWindowDialogDefProcResID = 65,
    kWindowUtilityDefProcResID = 66,
    kWindowUtilitySideTitleDefProcResID = 67,
    kWindowSheetDefProcResID = 68,
    kWindowSimpleDefProcResID = 69,
    kWindowSheetAlertDefProcResID = 70
};
```

Constants

`kWindowDocumentDefProcResID`

Defines Appearance-compliant standard document windows with a size box. Standard document windows created with this resource ID can use variation codes to create windows with vertical and horizontal zoom boxes.

Available with Appearance 1.0 and later.

Declared in `MacWindows.h`.

`kWindowDialogDefProcResID`

Defines Appearance-compliant dialog and alert boxes. Modal and movable modal dialog boxes created with this resource ID are displayed with no space between their content and structure region. Alert boxes created with this resource ID are displayed with a red-tinged border.

Declared in `MacWindows.h`.

Available with Appearance 1.0 and later.

`kWindowUtilityDefProcResID`

Defines Appearance-compliant utility (floating) windows with a top title bar and a size box.

Available with Appearance 1.0 and later.

Declared in `MacWindows.h`.

`kWindowUtilitySideTitleDefProcResID`

Defines Appearance-compliant utility (floating) windows with a side title bar and a size box.

Available with Appearance 1.0 and later.

Declared in `MacWindows.h`.

`kWindowSheetDefProcResID`

Defines a window sheet.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowSimpleDefProcResID`

Defines a simple window with no window frame.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowSheetAlertDefProcResID`

Defines a sheet window that is displayed as an alert (rather than a dialog) on Mac OS 9.

Available in Mac OS X v10.1 and later.

Declared in `MacWindows.h`.

Discussion
Window resource IDs are changed with Appearance Manager 1.0. The Window Manager now provides many new standard, Appearance-compliant window resource IDs for your application.

You can use a window resource ID constant to create a window definition ID; see "Pre-Appearance Window Definition IDs" in*Window Manager Legacy Reference* for more details.

Note that the standard Appearance-compliant resource ID constants `kWindowDocumentDefProcResID`, `kWindowUtilityDefProcResID`, and `kWindowUtilitySideTitleDefProcResID` specify windows with collapse boxes.

Resource IDs 0 through 127 are reserved for use by the system.

Appearance-Compliant Window Definition ID Constants

Define different window kinds.

```
enum {
    kWindowDocumentProc = 1024,
    kWindowGrowDocumentProc = 1025,
    kWindowVertZoomDocumentProc = 1026,
    kWindowVertZoomGrowDocumentProc = 1027,
    kWindowHorizZoomDocumentProc = 1028,
    kWindowHorizZoomGrowDocumentProc = 1029,
    kWindowFullZoomDocumentProc = 1030,
    kWindowFullZoomGrowDocumentProc = 1031
};
enum {
    kWindowPlainDialogProc = 1040,
    kWindowShadowDialogProc = 1041,
    kWindowModalDialogProc = 1042,
    kWindowMovableModalDialogProc = 1043,
    kWindowAlertProc = 1044,
    kWindowMovableAlertProc = 1045
};
enum {
    kWindowMovableModalGrowProc = 1046
};
enum {
    kWindowFloatProc = 1057,
    kWindowFloatGrowProc = 1059,
    kWindowFloatVertZoomProc = 1061,
    kWindowFloatVertZoomGrowProc = 1063,
    kWindowFloatHorizZoomProc = 1065,
    kWindowFloatHorizZoomGrowProc = 1067,
    kWindowFloatFullZoomProc = 1069,
    kWindowFloatFullZoomGrowProc = 1071
};
enum {
    kWindowFloatSideProc = 1073,
    kWindowFloatSideGrowProc = 1075,
    kWindowFloatSideVertZoomProc = 1077,
    kWindowFloatSideVertZoomGrowProc = 1079,
    kWindowFloatSideHorizZoomProc = 1081,
    kWindowFloatSideHorizZoomGrowProc = 1083,
    kWindowFloatSideFullZoomProc = 1085,
    kWindowFloatSideFullZoomGrowProc = 1087
};
enum {
    kWindowSheetProc = 1088,
    kWindowSheetAlertProc = 1120
};
enum {
    kWindowSimpleProc = 1104,
    kWindowSimpleFrameProc = 1105
};
```

Constants

`kWindowDocumentProc`

Appearance-compliant movable window with no size box or zoom box. Available with Appearance 1.0 and later. The corresponding pre-Appearance window definition ID is `noGrowDocProc`.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowGrowDocumentProc`
: Appearance-compliant standard document window (movable window with size box). Available with Appearance 1.0 and later. The corresponding pre-Appearance window definition ID is `documentProc`.

  Available in Mac OS X v10.0 and later.

  Declared in `MacWindows.h`.

`kWindowVertZoomDocumentProc`
: Appearance-compliant window with vertical zoom box and no size box. Available with Appearance 1.0 and later. There is no corresponding pre-Appearance window definition ID.

  Available in Mac OS X v10.0 and later.

  Declared in `MacWindows.h`.

`kWindowVertZoomGrowDocumentProc`
: Appearance-compliant window with vertical zoom box and size box. Available with Appearance 1.0 and later. There is no corresponding pre-Appearance window definition ID.

  Available in Mac OS X v10.0 and later.

  Declared in `MacWindows.h`.

`kWindowHorizZoomDocumentProc`
: Appearance-compliant window with horizontal zoom box and no size box. Available with Appearance 1.0 and later. There is no corresponding pre-Appearance window definition ID.

  Available in Mac OS X v10.0 and later.

  Declared in `MacWindows.h`.

`kWindowHorizZoomGrowDocumentProc`
: Appearance-compliant window with horizontal zoom box and size box. Available with Appearance 1.0 and later. There is no corresponding pre-Appearance window definition ID.

  Available in Mac OS X v10.0 and later.

  Declared in `MacWindows.h`.

`kWindowFullZoomDocumentProc`
: Appearance-compliant window with full zoom box and no size box. Available with Appearance 1.0 and later. The corresponding pre-Appearance window definition ID is `zoomNoGrow`.

  Available in Mac OS X v10.0 and later.

  Declared in `MacWindows.h`.

`kWindowFullZoomGrowDocumentProc`
: Appearance-compliant window with full zoom box and size box. Available with Appearance 1.0 and later. The corresponding pre-Appearance window definition ID is `zoomDocProc`.

  Available in Mac OS X v10.0 and later.

  Declared in `MacWindows.h`.

`kWindowPlainDialogProc`
: Appearance-compliant modeless dialog box. Available with Appearance 1.0 and later. The corresponding pre-Appearance window definition ID is `plainDBox`.

  Available in Mac OS X v10.0 and later.

  Declared in `MacWindows.h`.

`kWindowShadowDialogProc`
Appearance-compliant modeless dialog box with shadow. Available with Appearance 1.0 and later. The corresponding pre-Appearance window definition ID is `altDBoxProc`.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowModalDialogProc`
Appearance-compliant modal dialog box. Available with Appearance 1.0 and later. The corresponding pre-Appearance window definition ID is `dBoxProc`.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowMovableModalDialogProc`
Appearance-compliant movable modal dialog box. Available with Appearance 1.0 and later. The corresponding pre-Appearance window definition ID is `movableDBoxProc`.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowAlertProc`
Appearance-compliant alert box. Available with Appearance 1.0 and later. There is no corresponding pre-Appearance window definition ID.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowMovableAlertProc`
Appearance-compliant movable alert box. Available with Appearance 1.0 and later. There is no corresponding pre-Appearance window definition ID.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowMovableModalGrowProc`
Appearance-compliant movable modal dialog box with size box. Available with Appearance 1.0.1 and later. There is no corresponding pre-Appearance window definition ID.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowFloatProc`
Appearance-compliant utility (floating) window with no size box or zoom box. Available with Appearance 1.0 and later. The corresponding pre-Appearance window definition ID is `floatProc`.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowFloatGrowProc`
Appearance-compliant utility (floating) window with a size box. Available with Appearance 1.0 and later. The corresponding pre-Appearance window definition ID is `floatGrowProc`.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowFloatVertZoomProc`
: Appearance-compliant utility (floating) window with a vertical zoom box. Available with Appearance 1.0 and later. There is no corresponding pre-Appearance window definition ID.

  Available in Mac OS X v10.0 and later.

  Declared in `MacWindows.h`.

`kWindowFloatVertZoomGrowProc`
: Appearance-compliant utility (floating) window with a vertical zoom box and size box. Available with Appearance 1.0 and later. There is no corresponding pre-Appearance window definition ID.

  Available in Mac OS X v10.0 and later.

  Declared in `MacWindows.h`.

`kWindowFloatHorizZoomProc`
: Appearance-compliant utility (floating) window with a horizontal zoom box. Available with Appearance 1.0 and later. There is no corresponding pre-Appearance window definition ID.

  Available in Mac OS X v10.0 and later.

  Declared in `MacWindows.h`.

`kWindowFloatHorizZoomGrowProc`
: Appearance-compliant utility (floating) window with a horizontal zoom box and size box. Available with Appearance 1.0 and later. There is no corresponding pre-Appearance window definition ID.

  Available in Mac OS X v10.0 and later.

  Declared in `MacWindows.h`.

`kWindowFloatFullZoomProc`
: Appearance-compliant utility (floating) window with full zoom box. Available with Appearance 1.0 and later. The corresponding pre-Appearance window definition ID is `floatZoomProc`.

  Available in Mac OS X v10.0 and later.

  Declared in `MacWindows.h`.

`kWindowFloatFullZoomGrowProc`
: Appearance-compliant utility (floating) window with full zoom box and size box. Available with Appearance 1.0 and later. The corresponding pre-Appearance window definition ID is `floatZoomGrowProc`.

  Available in Mac OS X v10.0 and later.

  Declared in `MacWindows.h`.

`kWindowFloatSideProc`
: Appearance-compliant utility (floating) window with side title bar. Available with Appearance 1.0 and later. The corresponding pre-Appearance window definition ID is `floatSideProc`.

  Available in Mac OS X v10.0 and later.

  Declared in `MacWindows.h`.

`kWindowFloatSideGrowProc`
: Appearance-compliant utility (floating) window with side title bar and size box. Available with Appearance 1.0 and later. The corresponding pre-Appearance window definition ID is `floatSideGrowProc`.

  Available in Mac OS X v10.0 and later.

  Declared in `MacWindows.h`.

`kWindowFloatSideVertZoomProc`
: Appearance-compliant utility (floating) window with side title bar and vertical zoom box. Available with Appearance 1.0 and later. There is no corresponding pre-Appearance window definition ID.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowFloatSideVertZoomGrowProc`
: Appearance-compliant utility (floating) window with side title bar, vertical zoom box, and size box. Available with Appearance 1.0 and later. There is no corresponding pre-Appearance window definition ID.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowFloatSideHorizZoomProc`
: Appearance-compliant utility (floating) window with side title bar and horizontal zoom box. Available with Appearance 1.0 and later. There is no corresponding pre-Appearance window definition ID.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowFloatSideHorizZoomGrowProc`
: Appearance-compliant utility (floating) window with side title bar, horizontal zoom box, and size box. Available with Appearance 1.0 and later. There is no corresponding pre-Appearance window definition ID.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowFloatSideFullZoomProc`
: Appearance-compliant utility (floating) window with side title bar and full zoom box. Available with Appearance 1.0 and later. The corresponding pre-Appearance window definition ID is `floatSideZoomProc`.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowFloatSideFullZoomGrowProc`
: Appearance-compliant utility (floating) window with side title bar, full zoom box, and size box. Available with Appearance 1.0 and later. The corresponding pre-Appearance window definition ID is `floatSideZoomGrowProc`.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowSheetProc`
: A standard document sheet.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowSheetAlertProc`
: An alert sheet.

Available in Mac OS X v10.1 and later.

Declared in `MacWindows.h`.

`kWindowSimpleProc`
A window that has no structure region; the content covers the entire window.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowSimpleFrameProc`
A window that has a 1-pixel black frame as its structure.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

Basic Window Description State Constant

Define the window description state constant.

```
enum {
    kWindowIsCollapsedState = (1 << 0L)
};
```

Constants

`kWindowIsCollapsedState`
If the bit specified by this mask is set, the window is currently collapsed.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

Discussion

You can use this mask to set a bit in the `windowStateFlags` field of a structure of type `BasicWindowDescription` (page 1979), thereby specifying a transient window state.

Window Frame View Part Codes

HIView part codes used by window frame views.

```
enum {
    kHIWindowTitleBarPart = 2,
    kHIWindowDragPart = 3,
    kHIWindowTitleProxyIconPart = 2
};
```

Constants

`kHIWindowTitleBarPart`
Identifies the title bar part of a window frame view. This part code is used by the functions `GetWindowBounds` (page 1843) and `GetWindowRegion` (page 1864) when called with `kWindowTitleBarRgn`.

Available in Mac OS X v10.5 and later.

Declared in `HIWindowViews.h`.

`kHIWindowDragPart`
Identifies the draggable part of a window frame view. This part code is used by `GetWindowBounds` (page 1843) and `GetWindowRegion` (page 1864) when called with `kWindowDragRgn`.

Available in Mac OS X v10.5 and later.

Declared in `HIWindowViews.h`.

`kHIWindowTitleProxyIconPart`
Identifies the proxy icon part of a window frame title view. The title view is a subview of the window frame view and is identified by an HIViewID of `kHIViewWindowTitleID`. This part code is not used by the window frame view itself, but only by the title view. This part code is used by `GetWindowBounds` (page 1843) and `GetWindowRegion` (page 1864) when called with `kWindowTitleProxyIconRgn`.

Available in Mac OS X v10.5 and later.

Declared in `HIWindowViews.h`.

Discussion
These part codes are used by an HIView that implements the frame of a window. They may be used with the standard document windows provided by the Window Manager. A custom window frame view may optionally (but is not required to) implement these part codes in its event handlers for `kEventControlGetPartRegion` and `kEventControlGetPartBounds`.

Window Feature Bits

Specify features available in a window.

```
enum {
    kWindowCanGrow = (1 << 0),
    kWindowCanZoom = (1 << 1),
    kWindowCanCollapse = (1 << 2),
    kWindowIsModal = (1 << 3),
    kWindowCanGetWindowRegion = (1 << 4),
    kWindowIsAlert = (1 << 5),
    kWindowHasTitleBar = (1 << 6),
    kWindowSupportsDragHilite = (1 << 7),
    kWindowSupportsModifiedBit = (1 << 8),
    kWindowCanDrawInCurrentPort = (1 << 9),
    kWindowCanSetupProxyDragImage = (1 << 10),
    kWindowCanMeasureTitle = (1 << 11),
    kWindowWantsDisposeAtProcessDeath = (1 << 12),
    kWindowSupportsGetGrowImageRegion = (1 << 13),
    kWindowIsOpaque = (1 << 14),
    kWindowDefSupportsColorGrafPort = 0x40000002
};
```

Constants
`kWindowCanGrow`
If this bit (bit 0) is set, the window has a grow box (may not be visible).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowCanZoom`
If this bit (bit 1) is set, the window has a zoom box (may not be visible).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowCanCollapse`
If this bit (bit 2) is set, the window has a collapse box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowIsModal`
If this bit (bit 3) is set, the window should behave as modal.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowCanGetWindowRegion`
If this bit (bit 4) is set, the window supports a call to `GetWindowRegion` (page 1864).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowIsAlert`
If this bit (bit 5) is set, the window is an alert box (may be movable or not). When this constant is added to `kWindowIsModal`, the user should be able to switch out of the application and move the alert box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowHasTitleBar`
If this bit (bit 6) is set, the window has a title bar.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowSupportsDragHilite`
If the bit specified by this mask is set, the window supports the `kWindowMsgDragHilite` message. (Mac OS 8.5 and later.)

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowSupportsModifiedBit`
If the bit specified by this mask is set, the window supports the `kWindowMsgModified` message. (Mac OS 8.5 and later.)

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowCanDrawInCurrentPort`
If the bit specified by this mask is set, the window supports the `kWindowMsgDrawInCurrentPort` message. (Mac OS 8.5 and later.)

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowCanSetupProxyDragImage`
If the bit specified by this mask is set, the window supports the `kWindowMsgSetupProxyDragImage` message. (Mac OS 8.5 and later.)

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowCanMeasureTitle`
If the bit specified by this mask is set, the window supports the `kWindowMsgMeasureTitle` message. (Mac OS 8.5 and later.)

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowWantsDisposeAtProcessDeath`

If the bit specified by this mask is set, the window definition function wants to receive a `wDispose` message for the window if it still exists when the application quits.

Previously, the Window Manager would send a `wDispose` message only if the application explicitly closed the window with calls to the `CloseWindow` or `DisposeWindow` functions. The Window Manager would delete a window that still existed when the application called `ExitToShell` without notifying the window definition function, as part of the destruction of the process.

Note that if a window has the `kWindowWantsDisposeAtProcessDeath` feature bit set, the Window Manager sends your window definition function a `wDispose` message for the window when the application exits for any cause, including if your application crashes.

A window might want to set this feature flag if it allocates data when it is initialized that lives outside of the application heap and that is not automatically disposed when the application quits. The `wDispose` message is sent very early in the termination process, so it is still safe for the window definition function to call the system back (for example, you may want to do this in order to dispose of any auxiliary data). However, to ensure compatibility and to create the minimum performance impact, the window definition function should try to do as little as possible after receiving a `wDispose` message sent during the termination process. (Mac OS 8.5 and later.)

This feature is only available in Mac OS 8 and 9. It is not supported in Mac OS X.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowSupportsGetGrowImageRegion`

(Mac OS 8.5 and later.)

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowIsOpaque`

Indicates that the window is entirely opaque. If this feature bit is set, the window will use less memory because no alpha channel information will be stored for the window's pixels.

Available in Mac OS X v10.1 and later.

Declared in `MacWindows.h`.

`kWindowDefSupportsColorGrafPort`

Indicates that the window definition does not require that the current port be the classic Window Manager port. Not supported in Mac OS X.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

Discussion

For descriptions of the messages that correspond to these feature flags, see "Window Definition Message Constants" (page 2036).

Window Part Code Constants

Indicate which part of the window was hit.

```
typedef SInt16 WindowPartCode;
enum {
    inDesk = 0,
    inNoWindow = 0,
    inMenuBar = 1,
    inSysWindow = 2,
    inContent = 3,
    inDrag = 4,
    inGrow = 5,
    inGoAway = 6,
    inZoomIn = 7,
    inZoomOut = 8,
    inCollapseBox = 11,
    inProxyIcon = 12,
    inToolbarButton = 13,
    inStructure = 15
};
```

Constants

`inDesk`

The cursor is in the desktop region, not in the menu bar, a driver window, or any window that belongs to your application. When `FindWindow` (page 1827) returns `inDesk`, your application doesn't need to do anything.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`inNoWindow`

The cursor is not in a window.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`inMenuBar`

The user has pressed the mouse button while the cursor is in the menu bar. When `FindWindow` returns `inMenuBar`, your application typically adjusts its menus and then calls the Menu Manager function `MenuSelect` to let the user choose menu items.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`inSysWindow`

Not supported by Carbon.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`inContent`

The user has pressed the mouse button while the cursor is in the content area (excluding the size box in an active window) of one of your application's windows. When `FindWindow` returns `inContent`, your application determines how to handle clicks in the content region.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`inDrag`

The user has pressed the mouse button while the cursor is in the drag region of a window. When `FindWindow` returns `inDrag`, your application typically calls `DragWindow` to let the user drag the window to a new location.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`inGrow`

The user has pressed the mouse button while the cursor is in an active window's size box. When `FindWindow` returns `inGrow`, your application typically calls `ResizeWindow` (page 1925).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`inGoAway`

The user has pressed the mouse button while the cursor is in an active window's close box. When `FindWindow` returns `inGoAway`, your application typically calls`TrackGoAway` (page 1962) to track mouse activity while the button is down and then calls its own function for closing a window if the user releases the button while the cursor is in the close box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`inZoomIn`

The user has pressed the mouse button while the cursor is in the zoom box of an active window that is currently in the standard state. When `FindWindow` returns `inZoomIn`, your application typically calls `TrackBox` to track mouse activity while the button is down and then calls its own function for zooming a window if the user releases the button while the cursor is in the zoom box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`inZoomOut`

The user has pressed the mouse button while the cursor is in the zoom box of an active window that is currently in the user state. When `FindWindow` returns `inZoomOut`, your application typically calls the function `TrackBox` to track mouse activity while the button is down. Your application then calls its own function for zooming a window if the user releases the button while the cursor is in the zoom box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`inCollapseBox`

The user has pressed the mouse button while the cursor is in an active window's collapse box. When `FindWindow` returns `inCollapseBox`, your application typically does nothing, because the system will collapse your window for you.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`inProxyIcon`

The user has pressed the mouse button while the cursor is in the proxy icon of a window. When `FindWindow` returns `inProxyIcon`, your application typically calls the function `TrackWindowProxyDrag` (page 1963).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`inToolbarButton`
The user has pressed the mouse button while the cursor is in the toolbar button. (Mac OS X only.)

Available in Mac OS X v10.1 and later.

Declared in `MacWindows.h`.

`inStructure`
The user has pressed the mouse button while the cursor is in the window's structure region.

Available in Mac OS X v10.1 and later.

Declared in `MacWindows.h`.

Discussion
When your application receives a mouse-down event, you typically call `FindWindow` (page 1827), which returns an integer that specifies the location, in global coordinates, of the cursor at the time the user pressed the mouse button.

Window Modality Options

Specify the modality of a window.

```
typedef UInt32 WindowModality;
enum {
    kWindowModalityNone = 0,
    kWindowModalitySystemModal = 1,
    kWindowModalityAppModal = 2,
    kWindowModalityWindowModal = 3
};
```

Constants
`kWindowModalityNone`
A window does not prevent interaction with any other window in the system.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowModalitySystemModal`
The window is system-modal. In Mac OS 9 and earlier, the user cannot perform any other action until the window is dismissed. In Mac OS X, this constant produces the same behavior as `kWindowModalityAppModal`, so there is no way to prevent the user from interacting with windows from other applications.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowModalityAppModal`
The window is application-modal; that is the user cannot perform any other action within the application until the window is dismissed. The user can switch to other applications, however.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowModalityWindowModal`
The window is document-modal; the user cannot perform any other action within the current document window until the modal window associated with it is dismissed. The user can switch to other windows or applications, however. Sheets are document-modal.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

Window Position Constants

Define where to place a window.

```
typedef UInt32 WindowPositionMethod;
enum {
    kWindowCenterOnMainScreen = 1,
    kWindowCenterOnParentWindow = 2,
    kWindowCenterOnParentWindowScreen = 3,
    kWindowCascadeOnMainScreen = 4,
    kWindowCascadeOnParentWindow = 5,
    kWindowCascadeOnParentWindowScreen = 6,
    kWindowCascadeStartAtParentWindowScreen = 10,
    kWindowAlertPositionOnMainScreen = 7,
    kWindowAlertPositionOnParentWindow = 8,
    kWindowAlertPositionOnParentWindowScreen = 9
};
```

Constants

`kWindowCenterOnMainScreen`
Center the window, both horizontally and vertically, on the screen that contains the menu bar.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowCenterOnParentWindow`
Center the window, both horizontally and vertically, on the parent window. If the window to be centered is wider than the parent window, its left edge is aligned with the parent window's left edge.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowCenterOnParentWindowScreen`
Center the window, both horizontally and vertically, on the screen containing the parent window. In Mac OS X v10.3 and later, the parent window may be the same as the positioned window. In CarbonLib and earlier versions of Mac OS X, the parent window must be different from the positioned window.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowCascadeOnMainScreen`
Place the window just below the menu bar at the left edge of the main screen. Subsequent windows are placed on the screen relative to the first window, such that the frame of the preceding window remains visible behind the current window. The exact amount by which windows are offset depends upon the dimensions of the window frame under a given appearance.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowCascadeOnParentWindow`
: Place the window a distance below and to the right of the upper-left corner of the parent window such that the frame of the parent window remains visible behind the current window. The exact amount by which windows are offset depends upon the dimensions of the window frame under a given appearance.

  Available in Mac OS X v10.0 and later.

  Declared in `MacWindows.h`.

`kWindowCascadeOnParentWindowScreen`
: Place the window just below the menu bar at the left edge of the screen containing the parent window. Subsequent windows are placed on the screen relative to the first window, such that the frame of the preceding window remains visible behind the current window. The exact amount by which windows are offset depends upon the dimensions of the window frame under a given appearance. In Mac OS X v10.3 and later, the parent window may be the same as the positioned window. In CarbonLib and earlier versions of Mac OS X, the parent window must be different from the positioned window.

  Available in Mac OS X v10.0 and later.

  Declared in `MacWindows.h`.

`kWindowCascadeStartAtParentWindowScreen`
: Cascade the window on the screen containing the largest portion of its parent window, starting below and to the right of its parent window. The parent window must be different from the positioned window. (Available in Mac OS X v10.2 and CarbonLib 1.6 and later.)

  Available in Mac OS X v10.2 and later.

  Declared in `MacWindows.h`.

`kWindowAlertPositionOnMainScreen`
: Center the window horizontally and position it vertically on the screen that contains the menu bar, such that about one-fifth of the screen is above it. In Mac OS X v10.3 and later, the parent window may be the same as the positioned window. In CarbonLib and earlier versions of Mac OS X, the parent window must be different from the positioned window.

  Available in Mac OS X v10.0 and later.

  Declared in `MacWindows.h`.

`kWindowAlertPositionOnParentWindow`
: Center the window horizontally and position it vertically such that about one-fifth of the parent window is above it.

  Available in Mac OS X v10.0 and later.

  Declared in `MacWindows.h`.

`kWindowAlertPositionOnParentWindowScreen`
: Center the window horizontally and position it vertically such that about one-fifth of the screen containing the parent window is above it.

  Available in Mac OS X v10.0 and later.

  Declared in `MacWindows.h`.

Discussion
To specify the factors that determine how a window should be positioned, you supply one of these `WindowPositionMethod` constants to the function `RepositionWindow` (page 1924) or in the `BasicWindowDescription` structure of a resource of type `'wind'`. Do not confuse the `WindowPositionMethod` constants with the pre–Mac OS 8.5 Window Manager window positioning constants or use the `WindowPositionMethod` constants where the older constants are required (such as in the `StandardAlert` function or in `'WIND'`, `'DLOG'`, or `'ALRT'` resources).

System 7 Window Positioning Constants

Define window positioning constants used in `'WIND'`, `'DLOG'`, or `'ALRT'` resources, as well as the `StandardAlert` function.

```
enum {
    kWindowNoPosition = 0x0000,
    kWindowDefaultPosition = 0x0000,
    kWindowCenterMainScreen = 0x280A,
    kWindowAlertPositionMainScreen = 0x300A,
    kWindowStaggerMainScreen = 0x380A,
    kWindowCenterParentWindow = 0xA80A,
    kWindowAlertPositionParentWindow = 0xB00A,
    kWindowStaggerParentWindow = 0xB80A,
    kWindowCenterParentWindowScreen = 0x680A,
    kWindowAlertPositionParentWindowScreen = 0x700A,
    kWindowStaggerParentWindowScreen = 0x780A
};
```

Constants

`kWindowNoPosition`
No position.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowDefaultPosition`
Use the initial location.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowCenterMainScreen`
Center the window on the main screen.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowAlertPositionMainScreen`
Place the window in the alert position on the main screen.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowStaggerMainScreen`
Stagger the window on the main screen.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowCenterParentWindow`
Center the window on the parent window.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowAlertPositionParentWindow`
Place the window in the alert position on the parent window.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowStaggerParentWindow`
Stagger the window relative to the parent window.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowCenterParentWindowScreen`
Center the window on the parent window screen.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowAlertPositionParentWindowScreen`
Place the window in the alert position on the parent window screen.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowStaggerParentWindowScreen`
Stagger the window on the parent window screen.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

Discussion
You can use these constants in the optional positioning specification field of the window resource and in the dialog resource to override the window position established by the rectangle specified for the window or dialog. These positioning constants are convenient when the user is creating new documents or when you are handling your own dialog boxes and alert boxes.

These constants are passed into the `StandardAlert` function and are used in' `WIND` ', 'DLOG', and 'ALRT' templates. `StandardAlert` uses zero to specify the default position. Other calls use zero to specify “no position”.

Do not pass these constants to the`RepositionWindow` function or store these constants in the `BasicWindowDescription` structure of a 'WIND' resource.

The meaning of the terms used in the window positioning constant descriptions are as follows:

- center

  Centered both horizontally and vertically, relative either to a screen or to another window (if a window to be centered relative to another window is wider than the window that preceded it, it is pinned to the left edge; a narrower window is centered)

- stagger

  Located 20 pixels to the right and 20 pixels below the upper-left corner of the last window (in the case of staggering relative to a screen, the first window is placed just below the menu bar at the left edge of the screen, and subsequent windows are placed on that screen relative to the first window)

- alert position

  Centered horizontally and placed in the “alert position” vertically, that is, with about one-fifth of the window or screen above the new window and the rest below

- parent window

  Place in the position of the window in which the user was last working based on the frontmost window before the new window comes up.

Window Region Constants

Define various window regions.

```
typedef UInt16 WindowRegionCode;
enum {
    kWindowTitleBarRgn = 0,
    kWindowTitleTextRgn = 1,
    kWindowCloseBoxRgn = 2,
    kWindowZoomBoxRgn = 3,
    kWindowDragRgn = 5,
    kWindowGrowRgn = 6,
    kWindowCollapseBoxRgn = 7,
    kWindowTitleProxyIconRgn = 8,
    kWindowStructureRgn = 32,
    kWindowContentRgn = 33,
    kWindowUpdateRgn = 34,
    kWindowOpaqueRgn = 35,
    kWindowGlobalPortRgn = 40,
    kWindowToolbarButtonRgn = 41
};
```

Constants

`kWindowTitleBarRgn`
The entire area occupied by a window's title bar, including the title text region.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowTitleTextRgn`
That portion of a window's title bar that is occupied by the name of the window.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowCloseBoxRgn`
The area occupied by a window's close box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowZoomBoxRgn`
The area occupied by a window's zoom box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowDragRgn`
The draggable area of the window frame; this area includes the title bar and window outline and excludes the size box, close box, zoom box, and collapse box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowGrowRgn`
The area occupied by a window's size box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowCollapseBoxRgn`
The area occupied by a window's collapse box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowTitleProxyIconRgn`
Specifies the region in the window's title area that contains the proxy icon. The proxy icon region is always located within the window's title text region.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowStructureRgn`
The entire area occupied by a window, including the frame and content region; the window may be partially off-screen but its structure region does not change.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowContentRgn`
The window's content region—the part of a window in which your application displays the contents of the window or dialog, including the size box and any controls.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowUpdateRgn`
The window's update region––the part of the window that needs to be redrawn.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowOpaqueRgn`
Area of window considered to be opaque. Only valid for windows with alpha channels. (Mac OS X only)

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowGlobalPortRgn`
Bounds of the window's port in global coordinates; not affected by `CollapseWindow` (page 1810).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowToolbarButtonRgn`
Bounds of the toolbar button area.

Available in Mac OS X v10.4 and later.

Declared in `MacWindows.h`.

Discussion
You can pass constants of type `WindowRegionCode` in the `inRegionCode` parameter of `GetWindowRegion` (page 1864) to obtain a handle to a specific window region. The `WindowRegionCode` constants are available with Appearance Manager 1.0 and later.

Version Notes
With the Window Manager in Mac OS 8.5 and later, you may pass the `kWindowTitleProxyIconRgn`, `kWindowStructureRgn`, and `kWindowContentRgn` constants to the function `GetWindowRegion` (page 1864).

Window Latent Visibility Constants

Defines window latent visibility constants.

```
typedef UInt32 WindowLatentVisibility;
enum {
    kWindowLatentVisibleFloater = 1 << 0,
    kWindowLatentVisibleSuspend = 1 << 1,
    kWindowLatentVisibleFullScreen = 1 << 2,
    kWindowLatentVisibleAppHidden = 1 << 3,
    kWindowLatentVisibleCollapsedOwner = 1 << 4,
    kWindowLatentVisibleCollapsedGroup = 1 << 5
};
```

Constants

`kWindowLatentVisibleFloater`
The window is a floating window, and floating windows are hidden.

Available in Mac OS X v10.1 and later.

Declared in `MacWindows.h`.

`kWindowLatentVisibleSuspend`
The window has `kWindowHideOnSuspendAttribute` set and the application is suspended.

Available in Mac OS X v10.1 and later.

Declared in `MacWindows.h`.

`kWindowLatentVisibleFullScreen`
The window has `kWindowHideOnFullScreenAttribute` set and the mode is full-screen.

Available in Mac OS X v10.1 and later.

Declared in `MacWindows.h`.

`kWindowLatentVisibleAppHidden`
The window's process is hidden.

Available in Mac OS X v10.1 and later.

Declared in `MacWindows.h`.

`kWindowLatentVisibleCollapsedOwner`
The window is in an owned group, and the owner was collapsed.

Available in Mac OS X v10.1 and later.

Declared in `MacWindows.h`.

`kWindowLatentVisibleCollapsedGroup`
The window is in a group for which `kWindowGroupAttrHideOnCollapse` is set, and another window in the group was collapsed.

Available in Mac OS X v10.1 and later.

Declared in `MacWindows.h`.

Basic Window Description Version Constants

Describe different Mac OS window versions.

```
enum {
    kWindowDefinitionVersionOne = 1,
    kWindowDefinitionVersionTwo = 2
};
```

Constants

`kWindowDefinitionVersionOne`

Specifies a pre–Mac OS 8.5 Window Manager window. Windows of this version are created using a window definition ID and a Boolean value indicating whether or not the window has a close box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowDefinitionVersionTwo`

Specifies a Mac OS 8.5 Window Manager window. Windows of this version are created using class and attribute information. For details on classes and attributes, see "Window Class Constants" (page 1988) and "Window Attributes" (page 1998) respectively.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

Discussion

You may supply one of these values in the `windowDefinitionVersion` field of a structure of type `BasicWindowDescription` (page 1979) to specify the version of the window definition used for a window.

Window Property Persistent Constant

Define the window property persistent constant.

```
enum {
    kWindowPropertyPersistent = 0x00000001
};
```

Constants

`kWindowPropertyPersistent`

Indicates this property gets saved when the window is archived. Note, however, that window properties are not archived at all in Mac OS X v10.4.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

Window Variant Constants

Specify window variants.

```
enum {
    kDocumentWindowVariantCode = 0,
    kModalDialogVariantCode = 1,
    kPlainDialogVariantCode = 2,
    kShadowDialogVariantCode = 3,
    kMovableModalDialogVariantCode = 5,
    kAlertVariantCode = 7,
    kMovableAlertVariantCode = 9,
    kSideFloaterVariantCode = 8
};
```

Constants

`kDocumentWindowVariantCode`
Variation code for a document window.
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

`kModalDialogVariantCode`
Variation code for modal dialog boxes. The code can be added to 16 x the resource ID constant `kStandardWindowDefinition` to create a standard, pre-Appearance modal dialog box window.
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

`kPlainDialogVariantCode`
Variation code for a plain dialog box.
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

`kShadowDialogVariantCode`
Variation code for a shadow dialog box.
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

`kMovableModalDialogVariantCode`
Variation code for movable modal dialog boxes. The code can be added to 16 x the resource ID constant `kStandardWindowDefinition` to create a standard, pre-Appearance movable modal dialog box.
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

`kAlertVariantCode`
Variation code for a standard alert box.
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

`kMovableAlertVariantCode`
Variation code for a movable alert box.
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

`kSideFloaterVariantCode`
Variation code for utility (floating) windows with a side title bar. The code can be added to 16 x the resource ID constant `kFloatingWindowDefinition` to create a standard, pre-Appearance utility (floating) window with a side title bar.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

Window Transition Action Constants

Specify the type of window action taking place.

```
typedef UInt32 WindowTransitionAction;
enum {
    kWindowShowTransitionAction = 1,
    kWindowHideTransitionAction = 2,
    kWindowMoveTransitionAction = 3,
    kWindowResizeTransitionAction = 4
};
```

Constants

`kWindowShowTransitionAction`
Specifies that the animation display the window opening, that is, transitioning from a closed to an open state.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowHideTransitionAction`
Specifies that the animation display the window closing, that is, transitioning from an open to a closed state.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowMoveTransitionAction`
Moves the window. Use with the Slide transition effect. The `inRect` parameter contains the global coordinates of the window's new structure bounds and cannot be `NULL`. (Available in Mac OS X, and in CarbonLib 1.5 and later.)

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowResizeTransitionAction`
Resizes the window. Use with the Slide transition effect. The `inRect` parameter contains the global coordinates of the window's new structure bounds and cannot be `NULL`. (Available in Mac OS X and in CarbonLib 1.5 and later.)

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

Discussion

You may pass these `WindowTransitionAction` constants to the function `TransitionWindow` (page 1965) to specify the direction of the animation effect that is to be performed for a window.

Window Transition Effect Constants

Designate the type of transition effect to use to show or hide the window.

```
typedef UInt32 WindowTransitionEffect;
enum {
    kWindowZoomTransitionEffect = 1,
    kWindowSheetTransitionEffect = 2,
    kWindowSlideTransitionEffect = 3,
    kWindowFadeTransitionEffect = 4,
    kWindowGenieTransitionEffect = 5
};
```

Constants

`kWindowZoomTransitionEffect`
Specifies an animation that displays the window zooming between the open and closed states. The direction of the animation, whether from open to closed, or closed to open, depends upon the `WindowTransitionAction` constant specified in conjunction with the `WindowTransitionEffect` constant; see"Window Transition Action Constants" (page 2026) for descriptions of possible values.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowSheetTransitionEffect`
Zoom in or out from the parent window. Use with `TransitionWindowAndParent` (page 1966) and Show or Hide transition actions. (Available in Mac OS X, and in CarbonLib 1.5 and later.)

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowSlideTransitionEffect`
Slide the window into its new position. Use with `TransitionWindow` (page 1965) and Move or Resize transition actions. (Available in Mac OS X and in CarbonLib 1.5 and later.)

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowFadeTransitionEffect`
Fade the window into or out of visibility. Use with the Show or Hide transition action. (Available in Mac OS X v10.3 and later.)

Available in Mac OS X v10.3 and later.

Declared in `MacWindows.h`.

`kWindowGenieTransitionEffect`
Use the Genie effect that the Dock uses to minimize or maximize a window to show or hide the window. Use with the Show or Hide transition action. (Available in Mac OS X v10.3 and later.)

Available in Mac OS X v10.3 and later.

Declared in `MacWindows.h`.

Discussion

You may pass this `WindowTransitionEffect` constant to the function `TransitionWindow` (page 1965) to specify the type of animation effect that is to be performed for a window.

Window Activation Scope Constants

Defines window activation scope constants.

```
typedef UInt32 WindowActivationScope;
enum {
    kWindowActivationScopeNone = 0,
    kWindowActivationScopeIndependent = 1,
    kWindowActivationScopeAll = 2
};
```

Constants

`kWindowActivationScopeNone`
Windows with this scope are never activated by the Window Manager. Use `kWindowActivationScopeNone` when the window's visual state does not change based on activation (for example, tooltip windows), or when the client wants to manually control all activation. The window owner is free to explicitly activate or deactivate a window by calling `ActivateWindow` (page 1800).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowActivationScopeIndependent`
Windows with this scope are always active if visible and are unaffected by the activation state of other windows. This activation scope is automatically used by floating windows.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowActivationScopeAll`
Windows with this scope are activated relative to other windows with the same scope in the current process. Only one window with this scope can be active in the entire process. This activation scope is automatically used by document and dialog windows.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

Window Constrain Options

Constrain options for window resize, growing, and so on.

```
typedef UInt32 WindowConstrainOptions;
enum {
    kWindowConstrainMayResize = (1L << 0),
    kWindowConstrainMoveRegardlessOfFit = (1L << 1),
    kWindowConstrainAllowPartial = (1L << 2),
    kWindowConstrainCalcOnly = (1L << 3),
    kWindowConstrainUseTransitionWindow = (1L << 4),
    kWindowConstrainMoveMinimum = 1 << 6,
    kWindowConstrainUseSpecifiedBounds = 1 << 8,
    kWindowConstrainStandardOptions = kWindowConstrainMoveRegardlessOfFit
};
```

Constants

`kWindowConstrainMayResize`
The window may be resized if necessary to make it fit onscreen.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowConstrainMoveRegardlessOfFit`
The window may be moved even if it doesn't fit entirely onscreen.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowConstrainAllowPartial`
Allow partial intersection of the specified window region with the screen instead of requiring total intersection.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowConstrainCalcOnly`
Calculate the new window bounds but don't actually move the window.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowConstrainUseTransitionWindow`
Use `TransitionWindow` (page 1965) with `kWindowSlideTransitionEffect` to move windows onscreen.

Available in Mac OS X v10.2 and later.

Declared in `MacWindows.h`.

`kWindowConstrainMoveMinimum`
Move the window the minimum amount necessary to be onscreen. This option is only supported by the function `HIWindowConstrain` (page 1877). This option applies if a partial fit is not allowed (`kWindowConstrainAllowPartial` is not specified) or a partial fit is allowed, but the window is not even partially visible. In either case, the window will be moved just enough to be slightly onscreen. You may customize the minimum amount that is required to be visible by passing the desired dimensions in the `inMinimumSize` parameter to `HIWindowConstrain`.

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kWindowConstrainUseSpecifiedBounds`
Use the specified bounds of the window region to be constrained. This option is only supported by the function `HIWindowConstrain` (page 1877). The bounds are specified using the `ioBounds` parameter, allowing you to constrain a window to a hypothetical location. For example, if you plan to move your window such that its content region is at a certain location, and you want to know in advance before moving the window whether the window would be offscreen at that location, you can use this option.

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kWindowConstrainStandardOptions`
Use the most common options: don't resize the window, move the window regardless of fit to the screen, require total intersection of the specified window region with the screen, and move the window.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

Window Kinds

Identify how a window was created.

```
enum {
    dialogKind = 2,
    userKind = 8,
    kDialogWindowKind = 2,
    kApplicationWindowKind = 8
};
```

Constants

`dialogKind`
Obsolete equivalent to `kDialogWindowKind`.
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

`userKind`
Obsolete equivalent to `kApplicationWindowKind`.
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

`kDialogWindowKind`
Identifies all dialog or alert windows, whether created by system software or, indirectly through the Dialog Manager, by your application. The Dialog Manager uses this field to track dialog and alert windows.
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

`kApplicationWindowKind`
Identifies a window created directly by your application.
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

Discussion

The Window Manager uses these constants in the `windowKind` field of a color window structure or window structure. Your application can use any value greater than 7.

Window Group Selection Constants

Indicate which window group to select.

```
enum {
    kNextWindowGroup = true,
    kPreviousWindowGroup = false
};
```

Constants

`kNextWindowGroup`
Move to the next window group (in the z-order).
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

`kPreviousWindowGroup`
Move to the previous window group (in the z-order).
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

Window Group Attributes

Define attributes for window groups.

```
typedef UInt32 WindowGroupAttributes;
enum {
    kWindowGroupAttrSelectAsLayer = 1 << 0,
    kWindowGroupAttrMoveTogether = 1 << 1,
    kWindowGroupAttrLayerTogether = 1 << 2,
    kWindowGroupAttrSharedActivation = 1 << 3,
    kWindowGroupAttrHideOnCollapse = 1 << 4,
    kWindowGroupAttrFixedLevel = 1 << 5
};
```

Constants

`kWindowGroupAttrSelectAsLayer`

Makes the group behave somewhat as a layer of windows that move together. When any window in the group is brought to the front of the group, the entire group will also be brought to the front of the containing group's child hierarchy. Use of this constant is not recommended; its behavior is rarely useful.

Available in Mac OS X v10.1 and later.

Declared in `MacWindows.h`.

`kWindowGroupAttrMoveTogether`

The positions of the contents of this group with respect to each other cannot be changed. When one item moves, all other items are moved simultaneously. Note that if one window's position is changed by calling a Window Manager function in Mac OS X v10.4 and later, the positions of the other windows in the group are updated asynchronously—that is, their bounds are not necessarily updated during the function call itself, even though visually the windows move together.

Available in Mac OS X v10.1 and later.

Declared in `MacWindows.h`.

`kWindowGroupAttrLayerTogether`

The z-order of the contents of this group with respect to each other cannot be changed. When one item changes z-order, all other items are moved simultaneously. For purposes of z-ordering, the group and all its subgroups are effectively treated as if they were a single window in the parent group of this group.

Available in Mac OS X v10.1 and later.

Declared in `MacWindows.h`.

`kWindowGroupAttrSharedActivation`

The active state of the windows in this group is shared. The windows in the group are activated or deactivated according to the activation scope of the group, but when any window in the group changes activation, all other windows change to match.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowGroupAttrHideOnCollapse`

When any window in this group is collapsed, all other windows in this group are hidden. All subgroups of this group are also examined for this attribute, and any the windows of any subgroup with this attribute are also hidden. All windows will be shown again when the collapsed window is expanded.

Available in Mac OS X v10.1 and later.

Declared in `MacWindows.h`.

`kWindowGroupAttrFixedLevel`
If this attribute is specified, this window group's window level should be left unchanged. If this attribute is not specified, this window group's window level will be promoted to a value equal to the level of the next fixed-level window group beneath it in the window group hierarchy. (Available in Mac OS X v10.4 and later.)

Available in Mac OS X v10.4 and later.

Declared in `MacWindows.h`.

Version Notes
In Mac OS X v10.2.4 and later, the HIToolbox framework improved its use of the window group API so that showing a sheet on a window that was already grouped with another window would not break the existing grouping. To make this change work properly, applications that create their own window groups using the `kWindowGroupAttrMoveTogether` and `kWindowGroupAttrLayerTogether` attributes should also specify the `kWindowGroupAttrHideOnCollapse` and `kWindowGroupAttrSharedActivation` attributes.

Obsolete Window Group Attributes

Define obsolete window group attribute names.

```
enum {
    kWindowGroupAttrSelectable = kWindowGroupAttrSelectAsLayer,
    kWindowGroupAttrPositionFixed = kWindowGroupAttrMoveTogether,
    kWindowGroupAttrZOrderFixed = kWindowGroupAttrLayerTogether
};
```

Constants
`kWindowGroupAttrSelectable`
Obsolete name for `kWindowGroupAttrSelectAsLayer`.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowGroupAttrPositionFixed`
Obsolete name; use `kWindowGroupAttrMoveTogether` instead.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowGroupAttrZOrderFixed`
Obsolete name; use `kWindowGroupAttrLayerTogether` instead.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

Window Group Content Options

Window group counting options.

```
typedef UInt32 WindowGroupContentOptions;
enum {
    kWindowGroupContentsReturnWindows = 1 << 0,
    kWindowGroupContentsRecurse = 1 << 1,
    kWindowGroupContentsVisible = 1 << 2
};
```

Constants

`kWindowGroupContentsReturnWindows`
Count only windows in the window group.
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

`kWindowGroupContentsRecurse`
Recursively count windows of any subgroups of windows in the specified window group.
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

`kWindowGroupContentsVisible`
Counts only visible windows.
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

Discussion
You use these constants with the `CountWindowGroupContents` (page 1813) function.

Window Class Position Constants

Specify which window in the class to select.

```
enum {
    kFirstWindowOfClass = -1,
    kLastWindowOfClass = 0
};
```

Constants

`kFirstWindowOfClass`
Select the first window in the class.

`kLastWindowOfClass`
Select the last window in the class.

Discussion
These constants describe special cases for the “behind” parameter in window creation calls.

Window Definition Type Constants

Defines the type of custom window definition.

```
typedef UInt32 WindowDefType;
enum {
    kWindowDefProcPtr = 0,
    kWindowDefObjectClass = 1,
    kWindowDefProcID = 2,
    kWindowDefHIView = 3
};
```

Constants

`kWindowDefProcPtr`
The definition is procedure pointer–based.
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

`kWindowDefObjectClass`
The definition is a toolbox object.
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

`kWindowDefProcID`
An ID that identifies a particular `'WDEF'` and would typically be one of the constants described in "Appearance-Compliant Window Definition ID Constants" (page 2004).
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

`kWindowDefHIView`
The definition is an HIView-based object.
Available in Mac OS X v10.2 and later.
Declared in `MacWindows.h`.

Window Definition Procedure Constant

Define the window definition procedure constant.

```
enum {
    kWindowDefProcType = 'WDEF'
};
```

Constants

`kWindowDefProcType`
Window definition type.
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

Window Definition Hit Test Result Code Constants

Defines result constants to be used by window definition hit testing.

```
typedef SInt16 WindowDefPartCode;
enum {
    wNoHit = 0,
    wInContent = 1,
    wInDrag = 2,
    wInGrow = 3,
    wInGoAway = 4,
    wInZoomIn = 5,
    wInZoomOut = 6,
    wInCollapseBox = 9,
    wInProxyIcon = 10,
    wInToolbarButton = 11,
    wInStructure = 13
};
```

Constants

`wNoHit`

The mouse-down event did not occur in the content region or the drag region of any active or inactive window or in the close, size, zoom, or collapse box of an active window. The return value `wNoHit` might also mean that the point isn't in the window. The standard window definition functions, for example, return `wNoHit` if the point is in the window frame but not in the title bar.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`wInContent`

The mouse-down event occurred in the content region of an active or inactive window (with the exception of the size box).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`wInDrag`

The mouse-down event occurred in the drag region of an active or inactive window.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`wInGrow`

The mouse-down occurred in the size box of an active window.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`wInGoAway`

The mouse-down event occurred in the close box of an active window.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`wInZoomIn`

The mouse-down event occurred in the zoom box of an active window that is currently in the standard state.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`wInZoomOut`
: The mouse-down event occurred in the zoom box of an active window that is currently in the user state.

  Available in Mac OS X v10.0 and later.

  Declared in `MacWindows.h`.

`wInCollapseBox`
: The mouse-down event occurred in the collapse box of an active window.

  Available with Appearance Manager 1.0 and later.

  Declared in `MacWindows.h`.

`wInProxyIcon`
: The mouse-down event occurred in the proxy icon of a window.

  Available in Mac OS X v10.0 and later.

  Declared in `MacWindows.h`.

`wInToolbarButton`
: The mouse-down event occurred in the toolbar button.

  Available in Mac OS X v10.1 and later.

  Declared in `MacWindows.h`.

`wInStructure`
: The mouse-down event occurred in the window's structure region.

  Available in Mac OS X v10.1 and later.

  Declared in `MacWindows.h`.

Discussion
In response to the `wHit` message, your window definition function should return one of these constants.

Return the constants `wInGrow`, `wInGoAway`, `wInZoomIn`, `wInZoomOut`, and `wInCollapseBox` only if the window is active—by convention, the size box, close box, zoom box, and collapse box aren't drawn if the window is inactive. In an inactive document window, for example, a mouse-down event in the part of the title bar that would contain the close box if the window were active is reported as `wInDrag`.

With the Mac OS 8.5 Window Manager and later, your window definition function may return the `wInProxyIcon` constant to report that a mouse-down event occurred in your window's proxy icon.

Window Definition Message Constants

Defines messages sent to non Carbon Event–based window definitions.

```
enum {
    kWindowMsgDraw = 0,
    kWindowMsgHitTest = 1,
    kWindowMsgCalculateShape = 2,
    kWindowMsgInitialize = 3,
    kWindowMsgCleanUp = 4,
    kWindowMsgDrawGrowOutline = 5,
    kWindowMsgDrawGrowBox = 6
};
enum {
    kWindowMsgGetFeatures = 7,
    kWindowMsgGetRegion = 8
};
enum {
    kWindowMsgDragHilite = 9,
    kWindowMsgModified = 10,
    kWindowMsgDrawInCurrentPort = 11,
    kWindowMsgSetupProxyDragImage = 12,
    kWindowMsgStateChanged = 13,
    kWindowMsgMeasureTitle = 14
};
enum {
    kWindowMsgGetGrowImageRegion = 19
};
```

Constants

`kWindowMsgDraw`
Draw the window's frame.
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

`kWindowMsgHitTest`
Report the location of a mouse-down event.
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

`kWindowMsgCalculateShape`
Calculate the structure region and the content region.
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

`kWindowMsgInitialize`
Perform additional initialization.
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

`kWindowMsgCleanUp`
Perform additional disposal.
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

`kWindowMsgDrawGrowOutline`
Draw the dotted outline of the window that you see during a resizing operation.
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

`kWindowMsgDrawGrowBox`
: Draw the outlines for the size box and the scroll bar.

  Available in Mac OS X v10.0 and later.

  Declared in `MacWindows.h`.

`kWindowMsgGetFeatures`
: Report the window's features.

  Available with Appearance Manager 1.0 and later.

  Declared in `MacWindows.h`.

`kWindowMsgGetRegion`
: Report the location of a specific window region.

  Available with Appearance Manager 1.0 and later.

  Declared in `MacWindows.h`.

`kWindowMsgDragHilite`
: Redraw the window's structure region to reflect the window's validity as a drag-and-drop destination. The Window Manager passes an accompanying Boolean value in your window definition function's `param` parameter. If the value passed is `true`, this indicates that the window's structure region should be highlighted. If the value passed is `false`, the structure region should be unhighlighted. Your window definition function should return 0 as the function result. (Mac OS 8.5 and later.)

  Available in Mac OS X v10.0 and later.

  Declared in `MacWindows.h`.

`kWindowMsgModified`
: Track the window's modification state. The Window Manager sends this message when the function `SetWindowModified` (page 1946) is called. The Window Manager passes an accompanying Boolean value in your window definition function's `param` parameter. If the value passed is `true`, the document contained in the window has been modified. If the value passed is `false`, the document has been saved to disk. You should redraw the window's structure region to reflect the new modification state, if appropriate. For example, system-defined document windows dim the proxy icon to indicate that the document has been modified by the user and cannot be moved at that time. Your window definition function should return 0 as the function result. (Mac OS 8.5 and later.)

  Available in Mac OS X v10.0 and later.

  Declared in `MacWindows.h`.

`kWindowMsgDrawInCurrentPort`
: Draw the window's frame in the current graphics port. Other than restricting drawing to the current port, this message is similar to the pre–Mac OS 8.5 Window Manager window definition message constant `wDraw`. (Mac OS 8.5 and later.)

  Available in Mac OS X v10.0 and later.

  Declared in `MacWindows.h`.

`kWindowMsgSetupProxyDragImage`

Create the image of the window's proxy icon that the Drag Manager uses to represent the icon while it is being dragged. When your application calls the function `TrackWindowProxyDrag` (page 1963), the Window Manager passes this message in your window definition function's `message` parameter and an accompanying pointer to a structure of type `SetupWindowProxyDragImageRec` (page 1984) in the `param` parameter. Your window definition function is responsible for setting the contents of the structure to contain the data describing the proxy icon's drag image. Your window definition function should return 0 as the function result. (Mac OS 8.5 and later.)

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowMsgStateChanged`

Be informed that some aspect of the window's public state has changed. The Window Manager passes this message in your window definition function's `message` parameter and an accompanying flag in the `param` parameter that indicates what part of the window's state has been altered. This message is simply a notification message—no response by the window definition function is required. Your window definition function should return 0 as the function result. The `kWindowMsgStateChanged` message is sent after the window's internal data has been updated, but before any redraw occurs onscreen. A window definition function should not redraw the window frame in response to this message. If it is necessary to redraw the window frame, the Window Manager notifies the window definition function with a `wDraw` message. See "Window Definition State-Changed Constant" (page 2039) for descriptions of the values that the Window Manager can pass to specify the state change that has occurred. (Mac OS 8.5 and later.)

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowMsgMeasureTitle`

Measure and return the ideal title width. The Window Manager passes this message in the window definition function's `message` parameter and an accompanying pointer to a structure of type `MeasureWindowTitleRec` (page 1981) in the `param` parameter. Your window definition function is responsible for setting the contents of the structure to contain data describing the title width. You should return 0 as the function result. (Mac OS 8.5 and later.)

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowMsgGetGrowImageRegion`

Obtain a region to XOR with window during grow or resize. Alter the `GetGrowImageRegionRec` structure passed with the message to the region to be XOR'd. (

Available in Carbon only.)

Declared in `MacWindows.h`.

Discussion
The Window Manager may pass one of these constants in the `message` parameter of your window definition function to specify the action that your function must perform. For descriptions of the feature bits that correspond to these messages, see "Window Feature Bits" (page 2011). Other messages are reserved for internal use by the system.

Window Definition State-Changed Constant

Define the window definition state-changed constant.

```
enum {
    kWindowStateTitleChanged = (1 << 0)
};
```

Constants

`kWindowStateTitleChanged`

If the bit specified by this mask is set, the window's title has changed.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

Discussion

If you implement a custom window definition function, when the Window Manager passes the `kWindowMsgStateChanged` message in your window definition function's `message` parameter it may also pass a value in the `param` parameter with one or more bits set to indicate what part of the window's state has changed. You may use this mask to test this value. For a description of the `kWindowMsgStateChanged` message, see "Window Definition Message Constants" (page 2036).

Special Considerations

Drawer State Constants

Define constants that indicate the current drawer state.

```
typedef UInt32 WindowDrawerState;
enum {
    kWindowDrawerOpening = 1,
    kWindowDrawerOpen = 2,
    kWindowDrawerClosing = 3,
    kWindowDrawerClosed = 4
};
```

Constants

`kWindowDrawerOpening`

The drawer is opening.

Available in Mac OS X v10.2 and later.

Declared in `MacWindows.h`.

`kWindowDrawerOpen`

The drawer is open.

Available in Mac OS X v10.2 and later.

Declared in `MacWindows.h`.

`kWindowDrawerClosing`

The drawer is closing.

Available in Mac OS X v10.2 and later.

Declared in `MacWindows.h`.

`kWindowDrawerClosed`

The drawer is closed.

Available in Mac OS X v10.2 and later.

Declared in `MacWindows.h`.

Version Notes

Introduced in Mac OS X v10.2.

Window Edge Constants

Specify the edge from which a drawer should appear.

```
enum {
    kWindowEdgeDefault = 0,
    kWindowEdgeTop = 1 << 0,
    kWindowEdgeLeft = 1 << 1,
    kWindowEdgeBottom = 1 << 2,
    kWindowEdgeRight = 1 << 3
};
```

Constants

`kWindowEdgeDefault`
The drawer should be opened on whatever edge of the parent window has previously been set as the drawer's preferred edge.

Available in Mac OS X v10.2 and later.

Declared in `MacWindows.h`.

`kWindowEdgeTop`
The drawer should slide out from the top edge.

Available in Mac OS X v10.2 and later.

Declared in `MacWindows.h`.

`kWindowEdgeLeft`
The drawer should slide out from the left edge.

Available in Mac OS X v10.2 and later.

Declared in `MacWindows.h`.

`kWindowEdgeBottom`
The drawer should slide out from the bottom edge.

Available in Mac OS X v10.2 and later.

Declared in `MacWindows.h`.

`kWindowEdgeRight`
The drawer should slide out from the right edge.

Available in Mac OS X v10.2 and later.

Declared in `MacWindows.h`.

Rotating Window Menu Item Constant

Indicates whether to add the rotating window item to the Window menu.

```
enum {
    kWindowMenuIncludeRotate = 1 << 0
};
```

Constants

`kWindowMenuIncludeRotate`
Requests that the standard window menu include a Rotate Windows menu item.

Available in Mac OS X v10.2 and later.

Declared in `MacWindows.h`.

Discussion
This constant is used with the function `CreateStandardWindowMenu` (page 1816).

Window Menu Item Property Constants

Constants used to access property data of items in the standard window menu.

```
enum {
    kHIWindowMenuCreator = 'wind',
    kHIWindowMenuWindowTag = 'wind'
};
```

Constants

`kHIWindowMenuCreator`
The property creator for accessing standard window menu item properties.
Available in Mac OS X v10.5 and later.
Declared in `MacWindows.h`.

`kHIWindowMenuWindowTag`
The property tag for accessing standard window menu item properties that hold windows (values of type `WindowRef`). Menu items with the `kHICommandSelectWindow` command ID will have a property with this tag that contains the window to be activated when that item is selected.
Available in Mac OS X v10.5 and later.
Declared in `MacWindows.h`.

Discussion
These constants are used with the Menu Manager functions `GetMenuItemProperty` (page 1281) and `SetMenuItemProperty` (page 1341).

Toolbar View Background Tag

A tag used to inform a custom toolbar view whether to draw its background or leave its background transparent.

```
enum {
    kHIToolbarViewDrawBackgroundTag = 'back'
};
```

Constants

`kHIToolbarViewDrawBackgroundTag`
A `SetControlData` (page 652) tag that is used by the standard window frame view to inform the toolbar view whether the view should draw its background or leave its background transparent. The data for this tag is a Boolean. If the data value is true, the toolbar view should draw its background as it desires. If the data value is false, the toolbar view should leave its background transparent so that the window's root view can show through the toolbar view. Currently, the toolbar view will be asked to leave its background transparent for windows with the textured or unified appearance.
Available in Mac OS X v10.4 and later.
Declared in `MacWindows.h`.

Window Paint Callback Options

Define options to use with the window paint callback function.

```
typedef OptionBits WindowPaintProcOptions;
enum {
    kWindowPaintProcOptionsNone = 0
};
```

Constants

`kWindowPaintProcOptionsNone`
No options.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

Part Identifier Constants

Used in the `value` field of the `ColorSpec` structure, define which part of the window the color affects.

```
enum {
    wContentColor = 0,
    wFrameColor = 1,
    wTextColor = 2,
    wHiliteColor = 3,
    wTitleBarColor = 4
};
```

Constants

`wContentColor`
Produces background color for content region of window.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`wFrameColor`
Produces color of window's outline.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`wTextColor`
Produces color of window's title and button text.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`wHiliteColor`
Reserved.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`wTitleBarColor`
Reserved.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

Discussion
When the Appearance Manager is available and you are using standard windows, all the fields of the window color table structure are ignored except the part identifier constant `wContentColor` in the `value` field of the `ColorSpec` structure, which produces the background color for the window's content region.

If you are creating your own custom windows, the window color table structure and all its part identifier constants can still be used.

Desk Pattern Resource ID

The resource ID of the desktop pattern.

```
enum {
    deskPatID = 16
};
```

Constants
`deskPatID`
The resource ID of the desktop pattern.
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

Discussion
The Window Manager provides the desk pattern resource ID constants, which is the ID of `Pattern` and `PixPat` resources that the operating system uses to draw the desktop. The operating system uses the `deskPatID` constant while the desktop is being drawn. It looks for a resource with this ID and uses the contents of the resource to draw the desktop.

Window Scrolling Options

Options for scrolling windows.

```
typedef UInt32 ScrollWindowOptions;
enum {
    kScrollWindowNoOptions = 0,
    kScrollWindowInvalidate = (1L << 0),
    kScrollWindowEraseToPortBackground = (1L << 1)
};
```

Constants
`kScrollWindowNoOptions`
No options.
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

`kScrollWindowInvalidate`
Add the exposed area to the window's update region.
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

`kScrollWindowEraseToPortBackground`
Erase the exposed area using the background color/pattern of the window's graphics port.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

Discussion
Use these constants with the `ScrollWindowRect` (page 1927) and `ScrollWindowRegion` (page 1928) functions.

Availability
Available in Mac OS 8.1 and later.

'wind' Resource Default Collection Item Constants

Specify default collection items in a window (`'wind'`) resource.

```
enum {
    kStoredWindowSystemTag = 'appl',
    kStoredBasicWindowDescriptionID = 'sbas',
    kStoredWindowPascalTitleID = 's255',
    kStoredWindowTitleCFStringID = 'cfst'
};
```

Constants
`kStoredWindowSystemTag`
This item tag specifies a system-defined collection item. Note that the `'appl'` collection item tag is reserved for use by Apple Computer, Inc. Do not define new collection items using this tag.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kStoredBasicWindowDescriptionID`
In combination with `kStoredWindowSystemTag`, this item ID specifies an item of type `BasicWindowDescription`. See `BasicWindowDescription` (page 1979) for details on this type.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kStoredWindowPascalTitleID`
In combination with `kStoredWindowSystemTag`, this item ID specifies a Pascal title string.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kStoredWindowTitleCFStringID`
This item tag specifies the CFString title string.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

Discussion
These constants specify the tag and the IDs that identify the default collection items contained in a resource of type `'wind'`.

Window Resource IDs

Define standard resource IDs for windows.

```
enum {
    kStandardWindowDefinition = 0,
    kRoundWindowDefinition = 1,
    kFloatingWindowDefinition = 124
};
```

Constants

`kStandardWindowDefinition`

Defines pre-Appearance standard document windows and dialog boxes. When mapping is enabled, this resource ID is mapped to `kWindowDocumentDefProcResID` or `kWindowDialogDefProcResID`. When mapped to `kWindowDocumentDefProcResID`, this produces an Appearance-compliant standard document window with no size box and no vertical or horizontal zoom box. When mapped to `kWindowDialogDefProcResID`, this produces an Appearance-compliant dialog box with no size box and a 3-pixel space between the dialog box's content and structure region.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kRoundWindowDefinition`

Defines pre-Appearance standard desk-accessory style windows. This resource ID is not mapped to any Appearance-compliant resource ID when mapping is enabled.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kFloatingWindowDefinition`

Defines pre-Appearance utility (floating) windows. When mapping is enabled, this resource ID is mapped to `kWindowUtilityDefProcResID` or `kWindowUtilitySideTitleDefProcResID`. When mapped to `kWindowUtilityDefProcResID`, this produces an Appearance-compliant utility window. When mapped to `kWindowUtilitySideTitleDefProcResID`, it produces an Appearance-compliant utility window with a side title bar.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

Discussion

Window resource IDs are changed with Appearance Manager 1.0. The Window Manager now provides many new standard, Appearance-compliant window resource IDs for your program. For a description of the Appearance-compliant window resource IDs, see "Appearance-Compliant Window Resource IDs" (page 2003).

You can use a window resource ID constant to create a window definition ID; see "Pre-Appearance Window Definition IDs" in *Window Manager Legacy Reference* for more details.

Resource IDs 0 through 127 are reserved for use by the system.

Window Availability Constants

Define window availability constants for Exposé and Spaces.

```
typedef OptionBits HIWindowAvailability;
enum {
    kHIWindowExposeHidden = 1 << 0,
    kHIWindowVisibleInAllSpaces = 1 << 8
};
```

Constants

`kHIWindowExposeHidden`

If this bit is set, the window is hidden during the "All Windows" and "Application windows" modes of Exposé. If this bit is not set, the window is visible during those modes.

Available in Mac OS X v10.4 and later.

Declared in `MacWindows.h`.

`kHIWindowVisibleInAllSpaces`

If this bit is set, the window is visible in all Spaces workspaces. If this bit is not set, the window is only visible in the workspace in which it was created.

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

Discussion

These mask bits are used with the function `HIWindowChangeAvailability` (page 1875) to override the default behavior of the Window Manager in determining whether a window is visible during Exposé or in all Spaces workspaces. By default, newly created windows of class `kDocumentWindowClass` are given an availability of `0` (meaning that they are available during Exposé), and windows from all other window classes are given an availability of `kHIWindowExposeHidden`.

Window Scale Mode Constants

Define window scale mode constants.

```
typedef UInt32 HIWindowScaleMode;
enum {
    kHIWindowScaleModeUnscaled = 0,
    kHIWindowScaleModeMagnified = 1,
    kHIWindowScaleModeFrameworkScaled = 2
};
```

Constants

`kHIWindowScaleModeUnscaled`

The window is not scaled at all because the display scale factor is 1.0.

Available in Mac OS X v10.4 and later.

Declared in `MacWindows.h`.

`kHIWindowScaleModeMagnified`

The window's backing store is being magnified by the Window Server because the display scale factor is not equal to 1.0 and because the window was not created with the `kWindowFrameworkScaledAttribute`.

Available in Mac OS X v10.4 and later.

Declared in `MacWindows.h`.

`kHIWindowScaleModeFrameworkScaled`
The window's contents are scaled to match the display scale factor because the display scale factor is not equal to 1.0 and because the window was created with `kWindowFrameworkScaledAttribute`.

Available in Mac OS X v10.4 and later.

Declared in `MacWindows.h`.

Discussion
A window's scale mode indicates in which resolution-independent scale mode it is operating.

Window Group Level Constants

Define window group level constants.

```
enum {
kWindowGroupLevelActive = 1,
kWindowGroupLevelInactive = 2,
kWindowGroupLevelPromoted = 3,
};
```

Constants
`kWindowGroupLevelActive`
The window level that is nominally used for windows in the group when the application is active. However, if a group with a higher window level is positioned below this group in the window group hierarchy, this group's active level will be promoted to match the level of the group in front of it. Use `kWindowGroupLevelPromoted` to determine the actual window level in use for a group.

Available in Mac OS X v10.4 and later.

Declared in `MacWindows.h`.

`kWindowGroupLevelInactive`
The window level for windows in the group when the application is inactive.

Available in Mac OS X v10.4 and later.

Declared in `MacWindows.h`.

`kWindowGroupLevelPromoted`
The window level that is actually used for windows in the group when the application is active. This level is the same as the Active window level or is a larger value to match the level of a group below this group. Setting the promoted window level explicitly is not recommended because the promoted level is reset by the Window Manager whenever the window group hierarchy structure changes. Therefore any changes that you make to the promoted level can be overwritten.

Available in Mac OS X v10.4 and later.

Declared in `MacWindows.h`.

Discussion
These constants are used when calling `GetWindowGroupLevelOfType` (page 1851) and `SetWindowGroupLevelOfType` (page 1942).

Pre-Appearance Window Definition IDs

Older window definition IDs used before the introduction of the Appearance Manager.

```
enum {
    documentProc = 0,
    dBoxProc = 1,
    plainDBox = 2,
    altDBoxProc = 3,
    noGrowDocProc = 4,
    movableDBoxProc = 5,
    zoomDocProc = 8,
    zoomNoGrow = 12,
    rDocProc = 16,
    floatProc = 1985,
    floatGrowProc = 1987,
    floatZoomProc = 1989,
    floatZoomGrowProc = 1991,
    floatSideProc = 1993,
    floatSideGrowProc = 1995,
    floatSideZoomProc = 1997,
    floatSideZoomGrowProc = 1999
};
```

Constants

`documentProc`

Pre-Appearance document window (movable window with size box).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`dBoxProc`

Pre-Appearance modal dialog box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`plainDBox`

Pre-Appearance modeless dialog box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`altDBoxProc`

Pre-Appearance modeless dialog box with shadow.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`noGrowDocProc`

Pre-Appearance movable window with no size box or zoom box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`movableDBoxProc`

Pre-Appearance movable modal dialog box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`zoomDocProc`
: Pre-Appearance movable window with size box and full zoom box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`zoomNoGrow`
: Pre-Appearance window with full zoom box and no size box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`rDocProc`
: Pre-Appearance rounded-corner window. You can control the diameter of curvature of a rounded-corner window (window type `rDocProc`) by adding one of these integers to the `rDocProc` constant:

`rDocProc` (diameters of curvature: 16, 16)

`rDocProc` + 2 (diameters of curvature: 4, 4)

`rDocProc` + 4 (diameters of curvature: 6, 6)

`rDocProc` + 6 (diameters of curvature: 10, 10)

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared in `MacWindows.h`.

`floatProc`
: Pre-Appearance utility (floating) window with no size box or zoom box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`floatGrowProc`
: Pre-Appearance utility (floating) window with size box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`floatZoomProc`
: Pre-Appearance utility (floating) window with zoom box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`floatZoomGrowProc`
: Pre-Appearance utility (floating) window with size box and zoom box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`floatSideProc`
: Pre-Appearance utility (floating) window with side title bar and no size or zoom box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`floatSideGrowProc`
: Pre-Appearance utility (floating) window with side title bar and size box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`floatSideZoomProc`

Pre-Appearance utility (floating) window with side title bar and zoom box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`floatSideZoomGrowProc`

Pre-Appearance utility (floating) window with side title bar, size box, and zoom box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

Discussion
Note that window definition IDs are changed with Appearance Manager 1.0. The Window Manager now provides many new, standard, Appearance-compliant window types.

Your application typically supplies a window definition ID to a resource of type `'WIND'` or to a window-creation function to specify which window definition function to use in creating the window. A variation code may also be used to describe variations of the same basic window.

The window definition ID is an integer that contains the resource ID of the window definition function in its upper 12 bits and a variation code in its lower 4 bits. For a given resource ID and variation code, the window definition ID is derived as follows: window definition ID = (16 x resource ID) + variation code.

The window definition IDs for dialog boxes and utility (floating) windows pertain to the appearances of these windows only, not their behaviors. For example, if you want a utility window to have the proper behavior, that is, float, your application must provide for it.

When mapping is enabled, standard pre-Appearance window definition function IDs will be mapped to their Appearance-compliant equivalents.

Result Codes

The table below lists result codes defined for the Window Manager.

Result Code	Value	Description
`errInvalidWindowRef`	-5600	The window is not valid. Available in Mac OS X v10.0 and later.
`errUnsupportedWindowAttributesForClass`	-5601	Attribute bits are inappropriate for the specified window class. Available in Mac OS X v10.0 and later.
`errWindowDoesNotHaveProxy`	-5602	No proxy attached to window. Available in Mac OS X v10.0 and later.
`errInvalidWindowProperty`	-5603	`'appl'` creator code not allowed. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
errWindowPropertyNotFound	-5604	The window property does not exist. Available in Mac OS X v10.0 and later.
errUnrecognizedWindowClass	-5605	Unknown window class. Available in Mac OS X v10.0 and later.
errCorruptWindowDescription	-5606	Incorrect size or version supplied in the BasicWindowDescription structure. Available in Mac OS X v10.0 and later.
errUserWantsToDragWindow	-5607	Entire window is being dragged, not proxy icon. Available in Mac OS X v10.0 and later.
errWindowsAlreadyInitialized	-5608	Called InitFloatingWindows twice, or called InitWindows and then InitFloatingWindows. Available in Mac OS X v10.0 and later.
errFloatingWindowsNotInitialized	-5609	Called HideFloatingWindows or ShowFloatingWindows without calling InitFloatingWindows. Available in Mac OS X v10.0 and later.
errWindowNotFound	-5610	No window was found that satisfies the search criteria. Available in Mac OS X v10.0 and later.
errWindowDoesNotFitOnscreen	-5611	The window does not fit on a single screen. Available in Mac OS X v10.0 and later.
windowAttributeImmutableErr	-5612	Tried to change a window attribute that can't be changed after the window is created. Available in Mac OS X v10.0 and later.
windowAttributesConflictErr	-5613	Passed two window attributes that are mutually exclusive. Available in Mac OS X v10.0 and later.
windowManagerInternalErr	-5614	Internal error in the Window Manager. Available in Mac OS X v10.0 and later.
windowWrongStateErr	-5615	The window state makes the current action invalid. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
windowGroupInvalidErr	-5616	The window group is not valid. Available in Mac OS X v10.0 and later.
windowAppModalStateAlreadyExistsErr	-5617	The window is already application modal. Available in Mac OS X v10.1 and later.
windowNoAppModalStateErr	-5618	The window is not currently application modal. Available in Mac OS X v10.1 and later.
errWindowDoesntSupportFocus	-30583	Not used. Available in Mac OS X v10.0 and later.
errWindowRegionCodeInvalid	-30593	The window region code is not valid. Available in Mac OS X v10.0 and later.

Other References

Apple Help Reference

Framework: Carbon/Carbon.h

Declared in AppleHelp.h

Overview

Apple Help is an update and enhancement to previous versions of the Mac OS help system. It is available in Mac OS 8.6 and later, including Mac OS X.

This reference describes the application programming interface (API) for registering help books and calling the Help Viewer application.

Functions

AHGotoMainTOC

Tells the Help Viewer to load the specified main table-of-contents page. (Deprecated in Mac OS X v10.4. Use `AHGotoPage` (page 2058) to jump to different books or anchors.)

```
OSStatus AHGotoMainTOC (
   AHTOCType toctype
);
```

Parameters

toctype

A value specifying which Help Center page should be loaded. If you pass the `kAHTOCTypeUser` constant, for example, the Help Viewer loads the Help Center page.

Return Value

A result code. See "Apple Help Result Codes" (page 2061).

Discussion

This function is synchronous. Calling this function opens the Help Viewer application, if required, and loads the specified Help Center page.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`AppleHelp.h`

AHGotoPage

Tells the Help Viewer to load the specified HTML page.

```
OSStatus AHGotoPage (
   CFStringRef bookname,
   CFStringRef path,
   CFStringRef anchor
);
```

Parameters

bookname

A string specifying the name of the help book that contains the page to be loaded. If the name of the help book (as specified by the `AppleTitle` meta tag) is SurfWriter Help, for example, you pass a string containing the value `SurfWriter Help`. You may pass `NULL` in this parameter.

path

A string specifying a path for the page to be loaded. Specify the path relative to the help book given in the `bookname` parameter. If you pass `NULL` in this parameter, the Help Viewer opens to your help book's title page. If you pass `NULL` in the `bookname` parameter, you must specify the value passed in the `path` parameter as a full URL in the form `file://`.

anchor

An optional string specifying an anchor to which the Help Viewer scrolls after loading the specified page. If you do not specify an anchor, the Help Viewer scrolls to the top of the page.

Return Value

A result code. See "Apple Help Result Codes" (page 2061).

Discussion

This function is synchronous. Calling this function opens the Help Viewer application, if required, and loads the specified HTML page.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`AppleHelp.h`

AHLookupAnchor

Tells the Help Viewer to search for a specified anchor and load it.

```
OSStatus AHLookupAnchor (
   CFStringRef bookname,
   CFStringRef anchor
);
```

Parameters

bookname

A string specifying the name of the help book to search for the desired anchor. If the name of the help book (as specified by the `AppleTitle` meta tag) is SurfWriter Help, for example, you pass a string containing the value `SurfWriter Help`. If you do not specify a value in this parameter, the Help Viewer searches all available help books for the specified anchor.

anchor

A string specifying an anchor to load. You should ensure that each anchor name in your help book is unique. If you specify an anchor name that is not unique, the Help Viewer displays a list of links to all the anchors with that name.

Return Value
A result code. See "Apple Help Result Codes" (page 2061).

Discussion
This function is synchronous. Calling this function opens the Help Viewer application, if required, and loads the specified anchor or the list of links to multiple anchors, as appropriate. You must specify anchor indexing when you index your help book to perform anchor lookup.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AppleHelp.h`

AHRegisterHelpBook

Registers a help book.

```
OSStatus AHRegisterHelpBook (
   const FSRef *appBundleRef
);
```

Parameters

appBundleRef

A file system reference for the bundle containing the help book you wish to register.

Return Value
A result code. See "Apple Help Result Codes" (page 2061).

Discussion
Carbon applications must call this function in order to register a help book. If your Cocoa application provides appropriate key/value pairs specifying a single help book in your application's property list, as described in *Providing User Assistance with Apple Help*, you only need to call `AHRegisterHelpBook` if you wish to call the Apple Help functions to access your help content yourself. If you install an additional help book that is not described in your property list, you must call the `AHRegisterHelpBook` function in order to have the new book appear in the Help Center. If you call this function and specify a help book that is already registered, the `AHRegisterHelpBook` function returns a result of `noErr`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AppleHelp.h`

AHSearch

Tells the Help Viewer to search for a specified string in a given help book.

```
OSStatus AHSearch (
   CFStringRef bookname,
   CFStringRef query
);
```

Parameters

bookname

A string specifying the name of the help book to be searched. If the name of the help book (as specified by the `AppleTitle` meta tag) is SurfWriter Help, for example, you pass a string containing the value `SurfWriter Help`. If you pass `NULL` in this parameter, the Help Viewer searches all available help books.

query

A string containing the text to search for.

Return Value
A result code. See "Apple Help Result Codes" (page 2061).

Discussion
This function is synchronous. Calling this function opens the Help Viewer application, if required, and displays the results of the search.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AppleHelp.h`

Constants

TOC Specification Constants

Passed to the `AHGotoMainTOC` function to identify the table of contents page to open.

```
typedef SInt16 AHTOCType;
enum {
    kAHTOCTypeUser = 0,
    kAHTOCTypeDeveloper = 1
};
```

Constants

`kAHTOCTypeUser`

Opens the Help Center.

Available in Mac OS X v10.0 and later.

Declared in `AppleHelp.h`.

`kAHTOCTypeDeveloper`

Opens the Developer Help Center.

Available in Mac OS X v10.0 and later.

Declared in `AppleHelp.h`.

Result Codes

The most common result codes returned by Apple Help are listed below.

Result Code	Value	Description
`kAHInternalErr`	-10790	The requested operation could not be completed. Available in Mac OS X v10.0 and later.
`kAHInternetConfigPrefErr`	-10791	There was an error while attempting to read Internet Config settings. Available in Mac OS X v10.0 and later.

Carbon Printing Reference

Framework:	Carbon/Carbon.h
Declared in	PMApplication.h PMApplicationDeprecated.h
Companion guide	Supporting Printing in Your Carbon Application

Overview

Carbon Printing is a C API used by Carbon applications to display a user interface for printing. Carbon Printing provides functions to:

- Display the Page Setup dialog, which allows a user to modify the page format of a document to be printed
- Display the Print dialog, which allows a user to modify the default settings for a print job
- Execute a print loop that displays a printing status dialog

Carbon applications also need to use Core Printing to fully implement their printing features. For information about Core Printing, see *Core Printing Reference*.

Note: Carbon Printing and the Carbon Human Interface Toolbox are not available to 64-bit applications. To build a 64-bit application with a user interface, you must use Cocoa. For information about printing in Cocoa applications, see *Printing Programming Topics for Cocoa*.

Functions by Task

Displaying the Page Setup and Print Dialogs

`PMSessionUseSheets` (page 2093)
Specifies that a printing dialog should be displayed as a sheet and specifies a function to call when the user dismisses the printing dialog.

`PMSessionPageSetupDialog` (page 2088)
Displays the Page Setup dialog and records the user's selections in a page format object.

`PMSessionPrintDialog` (page 2091)
Displays the Print dialog and records the user's selections in a print settings object.

PMShowPageSetupDialogAsSheet (page 2097)
Displays a Page Setup dialog as a sheet attached to a window.

PMShowPrintDialogWithOptions (page 2098)
Displays a Print dialog with a specified set of controls.

PMShowPrintDialogWithOptionsAsSheet (page 2099)
Displays a Print dialog with a specified set of controls as a sheet attached to a window.

PMSessionEnablePrinterPresets (page 2086)
Enables the use of printer presets in the Print dialog.

PMSessionDisablePrinterPresets (page 2086)
Disables the use of printer presets in the Print dialog.

Print Loop Functions

PMSessionBeginCGDocument (page 2082)
Begins a print job in which all drawing is to a Quartz graphics context. A printing status dialog informs the user of the job's progress.

PMSessionEndDocument (page 2087)
Ends a print job started by calling the function PMSessionBeginCGDocument (page 2082) or PMSessionBeginDocument (page 2083).

PMSessionBeginPage (page 2084)
Starts a new page for printing in the specified printing session.

PMSessionEndPage (page 2087)
Indicates the end of drawing the current page for the specified printing session.

PMSessionBeginDocument (page 2083) Deprecated in Mac OS X v10.5
Begins a print job in which, by default, all drawing is to a QuickDraw graphics port. A printing status dialog informs the user of the job's progress. (Deprecated. Use PMSessionBeginCGDocument (page 2082) instead.)

Creating, Calling, and Deleting Universal Procedure Pointers

NewPMSheetDoneUPP (page 2072)
Creates a new universal procedure pointer (UPP) to a sheet-done callback.

InvokePMSheetDoneUPP (page 2070)
Calls a sheet-done callback.

DisposePMSheetDoneUPP (page 2068)
Disposes of a universal procedure pointer (UPP) to a sheet-done callback.

DisposePMItemUPP (page 2067) Deprecated in Mac OS X v10.4
Disposes of a universal procedure pointer (UPP) to a dialog item event handling callback. (Deprecated. Use a printing dialog extension instead.)

DisposePMPageSetupDialogInitUPP (page 2067) Deprecated in Mac OS X v10.4
Disposes of a universal procedure pointer (UPP) to a Page Setup dialog initialization callback. (Deprecated. Use a printing dialog extension instead.)

DisposePMPrintDialogInitUPP (page 2068) Deprecated in Mac OS X v10.4
: Disposes of a universal procedure pointer (UPP) to a Print dialog initialization callback function. (Deprecated. Use a printing dialog extension instead.)

InvokePMItemUPP (page 2068) Deprecated in Mac OS X v10.4
: Calls a dialog item event handling callback. (Deprecated. Use a printing dialog extension instead.)

InvokePMPageSetupDialogInitUPP (page 2069) Deprecated in Mac OS X v10.4
: Calls a Page Setup dialog initialization callback. (Deprecated. Use a printing dialog extension instead.)

InvokePMPrintDialogInitUPP (page 2069) Deprecated in Mac OS X v10.4
: Calls a Print dialog initialization callback. (Deprecated. Use a printing dialog extension instead.)

NewPMItemUPP (page 2070) Deprecated in Mac OS X v10.4
: Creates a new universal procedure pointer (UPP) to a dialog item event handling callback. (Deprecated. Use a printing dialog extension instead.)

NewPMPageSetupDialogInitUPP (page 2071) Deprecated in Mac OS X v10.4
: Creates a new universal procedure pointer (UPP) to a Page Setup dialog initialization callback. (Deprecated. Use a printing dialog extension instead.)

NewPMPrintDialogInitUPP (page 2071) Deprecated in Mac OS X v10.4
: Creates a new universal procedure pointer (UPP) to a Print dialog initialization callback. (Deprecated. Use a printing dialog extension instead.)

Customizing the Page Setup and Print Dialogs

The functions in this section make it possible to extend a printing dialog using the so-called AppendDITL method. Instead of using these functions, you should create a printing dialog extension. If you use these functions to extend a printing dialog, you cannot use sheets.

PMGetDialogAccepted (page 2075) Deprecated in Mac OS X v10.4
: Determines whether the user has confirmed a printing dialog. (Deprecated. Use a printing dialog extension instead.)

PMGetDialogDone (page 2075) Deprecated in Mac OS X v10.4
: Determines whether the user has finished with a dialog. (Deprecated. Use a printing dialog extension instead.)

PMGetDialogPtr (page 2076) Deprecated in Mac OS X v10.4
: Obtains the dialog reference for a dialog. (Deprecated. Use a printing dialog extension instead.)

PMGetItemProc (page 2076) Deprecated in Mac OS X v10.4
: Obtains the item proc callback function for a custom dialog. (Deprecated. Use a printing dialog extension instead.)

PMGetModalFilterProc (page 2077) Deprecated in Mac OS X v10.4
: Obtains the event handling callback function for a modal dialog. (Deprecated. Use a printing dialog extension instead.)

PMSessionPageSetupDialogInit (page 2089) Deprecated in Mac OS X v10.4
: Initializes a custom Page Setup dialog. (Deprecated. Use a printing dialog extension instead.)

PMSessionPageSetupDialogMain (page 2090) Deprecated in Mac OS X v10.4
: Displays your application's custom Page Setup dialog. (Deprecated. Use a printing dialog extension instead.)

PMSessionPrintDialogInit (page 2091) Deprecated in Mac OS X v10.4
: Initializes a custom Print dialog. (Deprecated. Use a printing dialog extension instead.)

`PMSessionPrintDialogMain` (page 2092) Deprecated in Mac OS X v10.4
Displays your application's custom Print dialog. (Deprecated. Use a printing dialog extension instead.)

`PMSetDialogAccepted` (page 2094) Deprecated in Mac OS X v10.4
Indicates that the user confirmed a custom dialog. (Deprecated. Use a printing dialog extension instead.)

`PMSetDialogDone` (page 2095) Deprecated in Mac OS X v10.4
Indicates that the user finished with a custom dialog. (Deprecated. Use a printing dialog extension instead.)

`PMSetItemProc` (page 2096) Deprecated in Mac OS X v10.4
Installs an item proc callback function for items in a custom dialog. (Deprecated. Use a printing dialog extension instead.)

`PMSetModalFilterProc` (page 2096) Deprecated in Mac OS X v10.4
Installs an event handling callback function for a modal dialog. (Deprecated. Use a printing dialog extension instead.)

Legacy Carbon Printing Functions

The functions in this section are deprecated and have been replaced by session-based functions that take a `PMPrintSession` object. In some cases, the replacement function is also deprecated.

`PMBeginDocument` (page 2072) Deprecated in Mac OS X v10.4
Establishes a graphics context for imaging a document. This context is an opaque `grafPort`. (Deprecated. Use `PMSessionBeginCGDocument` (page 2082) instead.)

`PMBeginPage` (page 2073) Deprecated in Mac OS X v10.4
Initializes a scaling rectangle for printing a page. (Deprecated. Use `PMSessionBeginPage` (page 2084) instead.)

`PMEndDocument` (page 2074) Deprecated in Mac OS X v10.4
Closes the context created for imaging a document. (Deprecated. Use `PMSessionEndDocument` (page 2087) instead.)

`PMEndPage` (page 2074) Deprecated in Mac OS X v10.4
Finishes printing the current page. (Deprecated. Use `PMSessionEndPage` (page 2087) instead.)

`PMPageSetupDialog` (page 2078) Deprecated in Mac OS X v10.4
Displays the Page Setup dialog and records the user's selections in a `PMPageFormat` object. (Deprecated. Use `PMSessionPageSetupDialog` (page 2088) or `PMShowPageSetupDialogAsSheet` (page 2097) instead.)

`PMPageSetupDialogInit` (page 2078) Deprecated in Mac OS X v10.4
Initializes a custom Page Setup dialog. (Deprecated. Use a printing dialog extension instead.)

`PMPageSetupDialogMain` (page 2079) Deprecated in Mac OS X v10.4
Displays your application's customized Page Setup dialog. (Deprecated. Use a printing dialog extension instead.)

`PMPrintDialog` (page 2080) Deprecated in Mac OS X v10.4
Displays the Print dialog and records the user's selections in a `PMPrintSettings` object. (Deprecated. Use `PMSessionPrintDialog` (page 2091), `PMShowPrintDialogWithOptions` (page 2098), or `PMShowPrintDialogWithOptionsAsSheet` (page 2099) instead.)

`PMPrintDialogInit` (page 2080) Deprecated in Mac OS X v10.4
Initializes a custom Print dialog. (Deprecated. Use a printing dialog extension instead.)

`PMPrintDialogInitWithPageFormat` (page 2081) Deprecated in Mac OS X v10.4
Initializes a custom Print dialog. (Deprecated. Use a printing dialog extension instead.)

`PMPrintDialogMain` (page 2082) Deprecated in Mac OS X v10.4
Displays your application's custom Print dialog. (Deprecated. Use a printing dialog extension instead.)

Functions

DisposePMItemUPP

Disposes of a universal procedure pointer (UPP) to a dialog item event handling callback. (Deprecated in Mac OS X v10.4. Use a printing dialog extension instead.)

```
void DisposePMItemUPP (
   PMItemUPP userUPP
);
```

Parameters

userUPP
The universal procedure pointer.

Discussion

See the `PMItemProcPtr` (page 2100) callback function.

Availability

Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In

`PMApplicationDeprecated.h`

DisposePMPageSetupDialogInitUPP

Disposes of a universal procedure pointer (UPP) to a Page Setup dialog initialization callback. (Deprecated in Mac OS X v10.4. Use a printing dialog extension instead.)

```
void DisposePMPageSetupDialogInitUPP (
   PMPageSetupDialogInitUPP userUPP
);
```

Parameters

userUPP
The universal procedure pointer.

Discussion

See the `PMPageSetupDialogInitProcPtr` (page 2101) callback function.

Availability

Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
PMApplicationDeprecated.h

DisposePMPrintDialogInitUPP

Disposes of a universal procedure pointer (UPP) to a Print dialog initialization callback function. (Deprecated in Mac OS X v10.4. Use a printing dialog extension instead.)

```
void DisposePMPrintDialogInitUPP (
   PMPrintDialogInitUPP userUPP
);
```

Parameters

userUPP

The universal procedure pointer.

Discussion
See the PMPrintDialogInitProcPtr (page 2102) callback function.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
PMApplicationDeprecated.h

DisposePMSheetDoneUPP

Disposes of a universal procedure pointer (UPP) to a sheet-done callback.

```
void DisposePMSheetDoneUPP (
   PMSheetDoneUPP userUPP
);
```

Parameters

userUPP

A UPP to your sheet-done callback.

Discussion
See the callback PMSheetDoneProcPtr (page 2103) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
PMApplication.h

InvokePMItemUPP

Calls a dialog item event handling callback. (Deprecated in Mac OS X v10.4. Use a printing dialog extension instead.)

```
void InvokePMItemUPP (
   DialogRef theDialog,
   short item,
   PMItemUPP userUPP
);
```

Discussion
You should not need to use the function `InvokePMInvokeUPP`, as the printing system calls your item event handling callback function for you. See the `PMItemProcPtr` (page 2100) callback function.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`PMApplicationDeprecated.h`

InvokePMPageSetupDialogInitUPP

Calls a Page Setup dialog initialization callback. (Deprecated in Mac OS X v10.4. Use a printing dialog extension instead.)

```
void InvokePMPageSetupDialogInitUPP (
   PMPageFormat pageFormat,
   PMDialog *theDialog,
   PMPageSetupDialogInitUPP userUPP
);
```

Discussion
You should not need to use the function `InvokePMPageSetupDialogInitUPP`, as the printing system calls your Page Setup dialog initialization callback function for you. See the `PMPageSetupDialogInitProcPtr` (page 2101) callback function.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`PMApplicationDeprecated.h`

InvokePMPrintDialogInitUPP

Calls a Print dialog initialization callback. (Deprecated in Mac OS X v10.4. Use a printing dialog extension instead.)

```
void InvokePMPrintDialogInitUPP (
   PMPrintSettings printSettings,
   PMDialog *theDialog,
   PMPrintDialogInitUPP userUPP
);
```

Discussion
You should not need to use the function `InvokePMPrintDialogInitUPP`, as the printing system calls your Print dialog initialization callback function for you. See the `PMPrintDialogInitProcPtr` (page 2102) callback function.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`PMApplicationDeprecated.h`

InvokePMSheetDoneUPP

Calls a sheet-done callback.

```
void InvokePMSheetDoneUPP (
   PMPrintSession printSession,
   WindowRef documentWindow,
   Boolean accepted,
   PMSheetDoneUPP userUPP
);
```

Discussion
You should not need to use the function `InvokePMSheetDoneUPP`, as the printing system calls your sheet-done callback function for you. See the callback `PMSheetDoneProcPtr` (page 2103) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`PMApplication.h`

NewPMItemUPP

Creates a new universal procedure pointer (UPP) to a dialog item event handling callback. (Deprecated in Mac OS X v10.4. Use a printing dialog extension instead.)

```
PMItemUPP NewPMItemUPP (
   PMItemProcPtr userRoutine
);
```

Parameters
userRoutine
A pointer to your dialog item event handling callback.

Return Value
On return, a UPP to the dialog item event handling callback.

Discussion
See the `PMItemProcPtr` (page 2100) callback function.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`PMApplicationDeprecated.h`

NewPMPageSetupDialogInitUPP

Creates a new universal procedure pointer (UPP) to a Page Setup dialog initialization callback. (Deprecated in Mac OS X v10.4. Use a printing dialog extension instead.)

```
PMPageSetupDialogInitUPP NewPMPageSetupDialogInitUPP (
   PMPageSetupDialogInitProcPtr userRoutine
);
```

Parameters
userRoutine
A pointer to your Page Setup dialog initialization callback.

Return Value
On return, a UPP to the Page Setup dialog initialization callback.

Discussion
See the `PMPageSetupDialogInitProcPtr` (page 2101) callback function.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`PMApplicationDeprecated.h`

NewPMPrintDialogInitUPP

Creates a new universal procedure pointer (UPP) to a Print dialog initialization callback. (Deprecated in Mac OS X v10.4. Use a printing dialog extension instead.)

```
PMPrintDialogInitUPP NewPMPrintDialogInitUPP (
   PMPrintDialogInitProcPtr userRoutine
);
```

Parameters
userRoutine
A pointer to your Print dialog initialization callback.

Return Value
On return, a UPP to the Print dialog initialization callback.

Discussion
See the `PMPrintDialogInitProcPtr` (page 2102) callback function.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`PMApplicationDeprecated.h`

NewPMSheetDoneUPP

Creates a new universal procedure pointer (UPP) to a sheet-done callback.

```
PMSheetDoneUPP NewPMSheetDoneUPP (
   PMSheetDoneProcPtr userRoutine
);
```

Parameters
`userRoutine`
A pointer to your sheet-done callback.

Return Value
On return, a UPP to the sheet-done callback.

Discussion
See the callback `PMSheetDoneProcPtr` (page 2103) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`PMApplication.h`

PMBeginDocument

Establishes a graphics context for imaging a document. This context is an opaque `grafPort`. (Deprecated in Mac OS X v10.4. Use `PMSessionBeginCGDocument` (page 2082) instead.)

```
OSStatus PMBeginDocument (
    PMPrintSettings printSettings,
    PMPageFormat pageFormat,
    PMPrintContext *printContext
);
```

Parameters
`printSettings`
A `PMPrintSettings` object.

pageFormat
A PMPageFormat object.

printContext
On return, an initialized PMPrintContext object.

Return Value
A result code. See Core Printing Result Codes.

Discussion
Valid within a PMBegin/PMEnd block. You must balance a call to PMBeginDocument with a call to PMEndDocument.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
PMApplicationDeprecated.h

PMBeginPage

Initializes a scaling rectangle for printing a page. (Deprecated in Mac OS X v10.4. Use PMSessionBeginPage (page 2084) instead.)

```
OSStatus PMBeginPage (
    PMPrintContext printContext,
    const PMRect *pageFrame
);
```

Parameters

printContext
A PMPrintContext object.

pageFrame
A pointer to a bounding rectangle for drawing the page. This value is usually obtained from the function PMGetAdjustedRect, but if no scaling is needed, pass NULL. In Mac OS X, this parameter is ignored.

Return Value
A result code. See Core Printing Result Codes.

Discussion
Valid after calling PMBegin and PMBeginDocument.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
PMApplicationDeprecated.h

PMEndDocument

Closes the context created for imaging a document. (Deprecated in Mac OS X v10.4. Use `PMSessionEndDocument` (page 2087) instead.)

```
OSStatus PMEndDocument (
    PMPrintContext printContext
);
```

Parameters

printContext

On return, an invalidated `PMPrintContext` object.

Return Value

A result code. See Core Printing Result Codes.

Discussion

Valid after calling `PMBegin` and `PMBeginDocument`.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`PMApplicationDeprecated.h`

PMEndPage

Finishes printing the current page. (Deprecated in Mac OS X v10.4. Use `PMSessionEndPage` (page 2087) instead.)

```
OSStatus PMEndPage (
    PMPrintContext printContext
);
```

Parameters

printContext

A `PMPrintContext` object.

Return Value

A result code. See Core Printing Result Codes.

Discussion

Valid after calling `PMBegin`, `PMBeginDocument`, and `PMBeginPage`.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`PMApplicationDeprecated.h`

PMGetDialogAccepted

Determines whether the user has confirmed a printing dialog. (Deprecated in Mac OS X v10.4. Use a printing dialog extension instead.)

```
OSStatus PMGetDialogAccepted (
   PMDialog pmDialog,
   Boolean *process
);
```

Parameters

pmDialog

A `PMDialog` object representing your customized Page Setup or Print dialog.

process

Returns `true` if the user confirms the dialog. In the case of the Page Setup dialog, this means the user clicked the OK button; in the case of the Print dialog, this means the user clicked the Print or Preview button. Returns `false` if the user clicks the Cancel button.

Return Value

A result code. See Core Printing Result Codes.

Discussion

The function `PMGetDialogAccepted` is valid between the creation and release of a printing session and while displaying your custom Page Setup or Print dialog.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`PMApplicationDeprecated.h`

PMGetDialogDone

Determines whether the user has finished with a dialog. (Deprecated in Mac OS X v10.4. Use a printing dialog extension instead.)

```
OSStatus PMGetDialogDone (
   PMDialog pmDialog,
   Boolean *done
);
```

Parameters

pmDialog

A `PMDialog` object representing your customized Page Setup or Print dialog.

done

Returns `true` if the user clicked the OK, Print, or Cancel button; `false` if the user did not click any of these buttons.

Return Value

A result code. See Core Printing Result Codes.

Discussion
The function `PMGetDialogDone` is valid between the creation and release of a printing session and while displaying your Page Setup or Print dialog.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`PMApplicationDeprecated.h`

PMGetDialogPtr

Obtains the dialog reference for a dialog. (Deprecated in Mac OS X v10.4. Use a printing dialog extension instead.)

```
OSStatus PMGetDialogPtr (
   PMDialog pmDialog,
   DialogRef *theDialog
);
```

Parameters

pmDialog
The `PMDialog` object from which you wish to extract the `DialogRecord` structure.

theDialog
On return, a pointer to a Dialog Manager `DialogRef` object.

Return Value
A result code. See Core Printing Result Codes.

Discussion
The function `PMGetDialogPtr` is valid between the creation and release of a printing session and after calling `PMSessionPageSetupDialogInit` to initialize a Page Setup dialog, or calling `PMSessionPrintDialogInit` to initialize a Print dialog.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`PMApplicationDeprecated.h`

PMGetItemProc

Obtains the item proc callback function for a custom dialog. (Deprecated in Mac OS X v10.4. Use a printing dialog extension instead.)

```
OSStatus PMGetItemProc (
   PMDialog pmDialog,
   PMItemUPP *itemProc
);
```

Parameters

pmDialog

A `PMDialog` object representing the Page Setup or Print dialog for which you want to obtain the item proc function.

itemProc

On return, a pointer to the current dialog item proc function.

Return Value
A result code. See Core Printing Result Codes.

Discussion
The function `PMGetItemProc` is valid only between the creation and release of a printing session. You can call `PMGetItemProc` to get the existing item proc function before you replace it with a call to the function `PMSetItemProc`. You should call this function only if you need to handle events in a custom dialog you create using the `AppendDITL` method. In Mac OS X, you should create a custom dialog by writing a printing dialog extension. If you use this function to extend a printing dialog in Mac OS X, you cannot use sheets.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`PMApplicationDeprecated.h`

PMGetModalFilterProc

Obtains the event handling callback function for a modal dialog. (Deprecated in Mac OS X v10.4. Use a printing dialog extension instead.)

```
OSStatus PMGetModalFilterProc (
   PMDialog pmDialog,
   ModalFilterUPP *filterProc
);
```

Parameters

pmDialog

A `PMDialog` object representing the Page Setup or Print dialog for which you want to obtain the event handling function.

filterProc

On return, a pointer to the current dialog event handling function.

Return Value
A result code. See Core Printing Result Codes.

Discussion
The function `PMGetModalFilterProc` is valid only between the creation and release of a printing session. You can call `PMGetModalFilterProc` to get the existing event handling function before you replace it with a call to the function `PMSetModalFilterProc`. The event handling function is not called in Mac OS X.

You should call the function `PMGetModalFilterProc` only if you create a custom dialog using the `AppendDITL` method. In Mac OS X, you should create a custom dialog by writing a printing dialog extension. If you use this function to extend a printing dialog in Mac OS X, you cannot use sheets.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`PMApplicationDeprecated.h`

PMPageSetupDialog

Displays the Page Setup dialog and records the user's selections in a `PMPageFormat` object. (Deprecated in Mac OS X v10.4. Use `PMSessionPageSetupDialog` (page 2088) or `PMShowPageSetupDialogAsSheet` (page 2097) instead.)

```
OSStatus PMPageSetupDialog (
    PMPageFormat pageFormat,
    Boolean *accepted
);
```

Parameters

pageFormat
A `PMPageFormat` object.

accepted
Returns `true` if the user clicks the OK button, or `false` if the user clicks Cancel.

Return Value
A result code. See Core Printing Result Codes.

Discussion
Valid after calling `PMBegin` and creating a page format object. Never call `PMPageSetupDialog` between the pages of a document.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`PMApplicationDeprecated.h`

PMPageSetupDialogInit

Initializes a custom Page Setup dialog. (Deprecated in Mac OS X v10.4. Use a printing dialog extension instead.)

```
OSStatus PMPageSetupDialogInit (
    PMPageFormat pageFormat,
    PMDialog *newDialog
);
```

Parameters

pageFormat

A `PMPageFormat` object.

newDialog

On return, a pointer to an initialized `PMDialog` object, ready for customization by your application. Because the `PMPageSetupDialogMain` function does not include a parameter for passing this `PMDialog` object to your dialog initialization callback function, your application should store this pointer in a global variable or as extended data in the `PMPageFormat` object. See the discussion of the `PMPageSetupDialogInitProcPtr` callback function for more information.

Return Value
A result code. See Core Printing Result Codes.

Discussion
Valid after calling `PMBegin` and creating a page format object.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`PMApplicationDeprecated.h`

PMPageSetupDialogMain

Displays your application's customized Page Setup dialog. (Deprecated in Mac OS X v10.4. Use a printing dialog extension instead.)

```
OSStatus PMPageSetupDialogMain (
    PMPageFormat pageFormat,
    Boolean *accepted,
    PMPageSetupDialogInitUPP myInitProc
);
```

Parameters

pageFormat

A `PMPageFormat` object.

accepted

Returns `true` if the user clicks the OK button, or `false` if the user clicks Cancel.

myInitProc

A universal procedure pointer to your dialog initialization function. Your initialization function is defined by the callback `PMPageSetupDialogInitProcPtr`.

Return Value
A result code. See Core Printing Result Codes.

Discussion
Valid after calling `PMBegin` and creating a page format object

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`PMApplicationDeprecated.h`

PMPrintDialog

Displays the Print dialog and records the user's selections in a `PMPrintSettings` object. (Deprecated in Mac OS X v10.4. Use `PMSessionPrintDialog` (page 2091), `PMShowPrintDialogWithOptions` (page 2098), or `PMShowPrintDialogWithOptionsAsSheet` (page 2099) instead.)

```
OSStatus PMPrintDialog (
    PMPrintSettings printSettings,
    PMPageFormat constPageFormat,
    Boolean *accepted
);
```

Parameters

printSettings
A `PMPrintSettings` object.

constPageFormat
A `PMPageFormat` object.

accepted
Returns `true` if the user clicks the OK button, or `false` if the user clicks Cancel.

Return Value
A result code. See Core Printing Result Codes.

Discussion
Valid after calling `PMBegin` and creating a page format and print settings object.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`PMApplicationDeprecated.h`

PMPrintDialogInit

Initializes a custom Print dialog. (Deprecated in Mac OS X v10.4. Use a printing dialog extension instead.)

```
OSStatus PMPrintDialogInit (
    PMPrintSettings printSettings,
    PMDialog *newDialog
);
```

Parameters

printSettings
A `PMPrintSettings` object.

newDialog

On return, a pointer to an initialized `PMDialog` object, ready for customization by your application. Because the `PMPrintDialogMain` function does not include a parameter for passing this `PMDialog` object to your dialog initialization callback function, your application should store this pointer in a global variable or as extended data in the `PMPrintSettings` object. See the discussion of the `PMPrintDialogInitProcPtr` callback function for more information.

Return Value
A result code. See Core Printing Result Codes.

Discussion
Valid after calling `PMBegin` and creating a page format and print settings object.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`PMApplicationDeprecated.h`

PMPrintDialogInitWithPageFormat

Initializes a custom Print dialog. (Deprecated in Mac OS X v10.4. Use a printing dialog extension instead.)

```
OSStatus PMPrintDialogInitWithPageFormat (
    PMPrintSettings printSettings,
    PMPageFormat constPageFormat,
    PMDialog *newDialog
);
```

Parameters

printSettings

A `PMPrintSettings` object.

constPageFormat

A `PMPageFormat` object.

newDialog

On return, a pointer to an initialized `PMDialog` object, ready for customization by your application. Because the `PMPrintDialogMain` function does not include a parameter for passing this `PMDialog` object to your dialog initialization callback function, your application should store this pointer in a global variable or as extended data in the `PMPrintSettings` object. See the discussion of the `PMPrintDialogInitProcPtr` callback function for more information.

Return Value
A result code. See Core Printing Result Codes.

Discussion
Valid after calling `PMBegin` and creating a page format and print settings object.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`PMApplicationDeprecated.h`

PMPrintDialogMain

Displays your application's custom Print dialog. (Deprecated in Mac OS X v10.4. Use a printing dialog extension instead.)

```
OSStatus PMPrintDialogMain (
    PMPrintSettings printSettings,
    PMPageFormat constPageFormat,
    Boolean *accepted,
    PMPrintDialogInitUPP myInitProc
);
```

Parameters

printSettings

A `PMPrintSettings` object.

constPageFormat

A `PMPageFormat` object.

accepted

Returns `true` if the user clicks the OK button, or `false` if the user clicks Cancel.

myInitProc

A universal procedure pointer to your dialog initialization function. Your initialization function is defined by the callback `PMPrintDialogIinitProcPtr`.

Return Value

A result code. See Core Printing Result Codes.

Discussion

Valid after calling `PMBegin` and creating a page format and print settings object.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`PMApplicationDeprecated.h`

PMSessionBeginCGDocument

Begins a print job in which all drawing is to a Quartz graphics context. A printing status dialog informs the user of the job's progress.

```
OSStatus PMSessionBeginCGDocument (
   PMPrintSession printSession,
   PMPrintSettings printSettings,
   PMPageFormat pageFormat
);
```

Parameters

printSession

The printing session for the new print job.

printSettings

The print settings for the new print job.

pageFormat
The page format for the new print job.

Return Value
A result code. See Core Printing Result Codes.

Discussion
This function starts a print job in which your application draws in a Quartz graphics context, and should be called within your application's print loop. This function is similar to the function `PMSessionBeginCGDocumentNoDialog` except that the printing status dialog is displayed.

You must call this function between the creation and release of a printing session. See the function `PMCreateSession`. If you present a printing dialog before you call this function, when calling this function you should use the same `PMPrintSession` object you used to present the dialog.

Before you call `PMSessionBeginCGDocument`, you should call `PMSessionValidatePrintSettings` and `PMSessionValidatePageFormat` to make sure the specified print settings and page format objects are updated and valid. After you call `PMSessionBeginCGDocument`, if you call a function that changes the specified print settings or page format object, the change is ignored for the current print job.

This function must be called before its corresponding `End` function (`PMSessionEndDocument` (page 2087)). If the function `PMSessionBeginCGDocument` returns `noErr`, you must later call the `End` function, even if errors occur within the scope of the `Begin` and `End` functions.

The printing system automatically handles printing multiple copies. Your application does not need to perform any tasks other than specifying the number of copies in the printing session.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

See Also
`PMSessionBeginCGDocumentNoDialog`

Declared In
`PMApplication.h`

PMSessionBeginDocument

Begins a print job in which, by default, all drawing is to a QuickDraw graphics port. A printing status dialog informs the user of the job's progress. (Deprecated in Mac OS X v10.5. Use `PMSessionBeginCGDocument` (page 2082) instead.)

```
OSStatus PMSessionBeginDocument (
   PMPrintSession printSession,
   PMPrintSettings printSettings,
   PMPageFormat pageFormat
);
```

Parameters
printSession
The printing session for the new print job.

printSettings
The print settings for the new print job.

pageFormat
The page format for the new print job.

Return Value
A result code. See Core Printing Result Codes.

Discussion
The function `PMSessionBeginDocument` starts a print job and should be called within your application's print loop. This function is similar to the function `PMSessionBeginDocumentNoDialog` except that the printing status dialog is displayed.

You must call this function between the creation and release of a printing session. See the function `PMCreateSession`. If you present a printing dialog before you call `PMSessionBeginDocument`, you should use the same `PMPrintSession` object you used to present the dialog.

Before you call `PMSessionBeginDocument`, you should call `PMSessionValidatePrintSettings` and `PMSessionValidatePageFormat` to make sure the specified print settings and page format objects are updated and valid. After you call `PMSessionBeginDocument`, if you call a function that changes the specified print settings or page format object, the change is ignored for the current print job.

This function must be called before its corresponding `End` function (`PMSessionEndDocument` (page 2087)). If the function `PMSessionBeginDocument` function returns `noErr`, you must later call the `End` function, even if errors occur within the scope of the `Begin` and `End` functions.

The printing system automatically handles printing multiple copies. Your application does not need to perform any tasks other than specifying the number of copies in the printing session.

Special Considerations

In Mac OS X v10.4 and later, Apple recommends using the function `PMSessionBeginCGDocument` (page 2082) instead of this function. QuickDraw is deprecated and your application should be using Quartz 2D for its rendering.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Related Sample Code
CarbonSketch

Declared In
`PMApplicationDeprecated.h`

PMSessionBeginPage

Starts a new page for printing in the specified printing session.

```
OSStatus PMSessionBeginPage (
    PMPrintSession printSession,
    PMPageFormat pageFormat,
    const PMRect *pageFrame
);
```

Parameters

printSession

The printing session for the print job.

pageFormat

The page format for the new page. If you pass `NULL`, the printing system uses the page format you passed to `PMSessionBeginCGDocument` (page 2082).

pageFrame

You should pass `NULL`, as this parameter is currently unsupported.

Return Value

A result code. If the user cancels the print job, this function returns `kPMCancel`.

Discussion

You must call this function between the creation and release of a printing session. See the function `PMCreateSession`. You must call the functions `PMSessionBeginPage` and `PMSessionEndPage` (page 2087) within the scope of calls to the `Begin` print job function (`PMSessionBeginCGDocument` (page 2082)) and the `End` print job function (`PMSessionEndDocument` (page 2087)).

You should call the function `PMSessionError` immediately before you call `PMSessionBeginPage`. If `PMSessionError` returns an error, then you should not call the function `PMSessionBeginPage`. Because `PMSessionBeginPage` also initializes the printing graphics context, your application should not make assumptions about the state of the context (for example, the current font) between successive pages. After each call to `PMSessionBeginPage`, your application should call `PMSessionGetCGGraphicsContext` to obtain the current printing context.

If the function `PMSessionBeginPage` returns `noErr`, you must later call the function `PMSessionEndPage`, even if errors occur within the scope of `PMSessionBeginPage` and `PMSessionEndPage`.

The printing system automatically handles printing multiple copies. Your application does not need to perform any tasks other than specifying the number of copies in the printing session.

Special Considerations

Prior to Mac OS X v10.5, the *pageFormat* parameter is ignored. In Mac OS X v10.5 and later, the printing system supports multiple orientations within a print job. When you call this function and supply a page format, the orientation specified in the page format is used for the current page. Other settings in the page format, such as paper size or scaling, are ignored.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

CarbonSketch

Declared In

`PMApplication.h`

PMSessionDisablePrinterPresets

Disables the use of printer presets in the Print dialog.

```
OSStatus PMSessionDisablePrinterPresets (
    PMPrintSession session
);
```

Parameters

session

The printing session you use to present the Print dialog.

Return Value

A result code. See Core Printing Result Codes.

Discussion

When you call this function with the specified printing session, the Print dialog for that session does not show any printer presets. Presets are disabled by default.

Availability

Available in Mac OS X v10.2 and later.

Not available to 64-bit applications.

See Also

`PMSessionEnablePrinterPresets` (page 2086)

Declared In

`PMApplication.h`

PMSessionEnablePrinterPresets

Enables the use of printer presets in the Print dialog.

```
OSStatus PMSessionEnablePrinterPresets (
    PMPrintSession session,
    CFStringRef graphicsType
);
```

Parameters

session

The printing session you use to present the Print dialog.

graphicsType

The printer presets in the dialog should be suitable for rendering this type of graphic. Currently, the only defined graphics type is `"Photo"`.

Return Value

A result code. See Core Printing Result Codes.

Discussion

When you call this function with the specified printing session, the Print dialog shows the presets available for the specified graphics type.

Availability

Available in Mac OS X v10.2 and later.

Not available to 64-bit applications.

See Also
PMSessionDisablePrinterPresets (page 2086)

Declared In
PMApplication.h

PMSessionEndDocument

Ends a print job started by calling the function PMSessionBeginCGDocument (page 2082) or PMSessionBeginDocument (page 2083).

```
OSStatus PMSessionEndDocument (
   PMPrintSession printSession
);
```

Parameters

printSession

The current printing session. On return, the printing session is no longer valid; however, you must still call the function PMRelease to release the object.

Return Value
A result code. See Core Printing Result Codes.

Discussion
You must call this function between the creation and release of a printing session. See the function PMCreateSession. The function PMSessionEndDocument is used to end a print job, and it should be called within your application's print loop after the call to PMSessionEndPage and before releasing the printing session.

This function must be called after its corresponding Begin function (PMSessionBeginCGDocument (page 2082) or PMSessionBeginDocument (page 2083)). If the Begin function returns noErr, the function PMSessionEndDocument must be called, even if errors occur within the scope of the Begin and End functions. You should not call PMSessionEndDocument if the Begin function returns an error.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
CarbonSketch

Declared In
PMApplication.h

PMSessionEndPage

Indicates the end of drawing the current page for the specified printing session.

```
OSStatus PMSessionEndPage (
   PMPrintSession printSession
);
```

Parameters

printSession

The printing session for the print job.

Return Value

A result code. See Core Printing Result Codes.

Discussion

You must call this function between the creation and release of a printing session. See the function `PMCreateSession`. You must call the functions `PMSessionBeginPage` (page 2084) and `PMSessionEndPage` within the scope of calls to the `Begin` print job function (`PMSessionBeginCGDocument` (page 2082)) and the `End` print job function (`PMSessionEndDocument` (page 2087)).

If the function `PMSessionBeginPage` returns `noErr`, you must later call the function `PMSessionEndPage`, even if errors occur within the scope of `PMSessionBeginPage` and `PMSessionEndPage`. You should not call `PMSessionEndPage` if `PMSessionBeginPage` returns an error.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

CarbonSketch

Declared In

`PMApplication.h`

PMSessionPageSetupDialog

Displays the Page Setup dialog and records the user's selections in a page format object.

```
OSStatus PMSessionPageSetupDialog (
   PMPrintSession printSession,
   PMPageFormat pageFormat,
   Boolean *accepted
);
```

Parameters

printSession

The printing session for which you want to display the dialog.

pageFormat

The page format object in which the user's selections are recorded.

accepted

A pointer to your Boolean variable. On return, `true` if the user clicks the OK button, or `false` if the user clicks Cancel. If you are using a sheet dialog, you should ignore the value returned in this parameter.

Return Value

A result code. See Core Printing Result Codes.

Discussion
You must call this function between the creation and release of a printing session. See the function `PMCreateSession`. Before you call the function `PMSessionPageSetupDialog` you should call the function `PMSessionValidatePageFormat` to make sure page format settings are updated and valid. You should call the function `PMSessionPageSetupDialog` outside the scope of your print loop.

The Page Setup dialog shows the current settings contained in the page format object. If the user changes these settings and clicks the OK button, the page format object is updated by the printing system with the user's selections. If the user clicks the Cancel button, the page format object is returned unchanged.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
CarbonSketch

Declared In
`PMApplication.h`

PMSessionPageSetupDialogInit

Initializes a custom Page Setup dialog. (Deprecated in Mac OS X v10.4. Use a printing dialog extension instead.)

```
OSStatus PMSessionPageSetupDialogInit (
   PMPrintSession printSession,
   PMPageFormat pageFormat,
   PMDialog *newDialog
);
```

Parameters

printSession
The current printing session.

pageFormat
A page format object.

newDialog
On return, a pointer to an initialized `PMDialog` object, ready for customization by your application. Because the `PMSessionPageSetupDialogMain` (page 2090) function does not include a parameter for passing this `PMDialog` object to your dialog initialization callback function, your application should store this pointer in a global variable or as extended data in the `PMPageFormat` object.

Return Value
A result code. See Core Printing Result Codes.

Discussion
The function `PMSessionPageSetupDialogInit` is valid only between the creation and release of a printing session and after creating a page format object. You should call this function only if you create a custom dialog using the `AppendDITL` method. In Mac OS X, you should create a custom dialog by writing a printing dialog extension. If you use this function to extend a printing dialog in Mac OS X, you cannot use sheets.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
PMApplicationDeprecated.h

PMSessionPageSetupDialogMain

Displays your application's custom Page Setup dialog. (Deprecated in Mac OS X v10.4. Use a printing dialog extension instead.)

```
OSStatus PMSessionPageSetupDialogMain (
   PMPrintSession printSession,
   PMPageFormat pageFormat,
   Boolean *accepted,
   PMPageSetupDialogInitUPP myInitProc
);
```

Parameters

printSession
The current printing session.

pageFormat
A page format object.

accepted
A pointer to your Boolean variable. On return, `true` if the user clicks the OK button, or `false` if the user clicks Cancel.

myInitProc
A universal procedure pointer to your dialog initialization function. Your initialization function is defined by the callback `PMPageSetupDialogInitProcPtr`.

Return Value
A result code. See Core Printing Result Codes.

Discussion
The function `PMSessionPageSetupDialogMain` is valid only between the creation and release of a printing session and after calling the function `PMSessionPageSetupDialogInit`. You should call this function only if you create a custom dialog using the `AppendDITL` method. In Mac OS X, you should create a custom dialog by writing a printing dialog extension. If you use this function to extend a printing dialog in Mac OS X, you cannot use sheets.

Your dialog initialization function is called before your custom Page Setup dialog is displayed. Your initialization function can append items to the Page Setup dialog, and should install an item proc using the `PMSetItemProc` (page 2096) function. You must pass the same page format object to each of the functions `PMSessionPageSetupDialogMain` and `PMSessionPageSetupDialogInit` (page 2089).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
PMApplicationDeprecated.h

PMSessionPrintDialog

Displays the Print dialog and records the user's selections in a print settings object.

```
OSStatus PMSessionPrintDialog (
   PMPrintSession printSession,
   PMPrintSettings printSettings,
   PMPageFormat constPageFormat,
   Boolean *accepted
);
```

Parameters

printSession

The printing session for which you want to display the dialog.

printSettings

The print settings object in which the user's selections are recorded.

constPageFormat

The page format object for the specified printing session.

accepted

A pointer to your Boolean variable. On return, `true` if the user accepts the dialog or `false` if the user cancels the dialog. If you are using a sheet dialog, you should ignore the value of this parameter.

Return Value

A result code. See Core Printing Result Codes.

Discussion

You must call this function between the creation and release of a printing session. See the function `PMCreateSession`. Before you call the function `PMSessionPrintDialog` you should call the function `PMSessionValidatePrintSettings` to make sure print settings are updated and valid. You should call the function `PMSessionPrintDialog` outside the scope of your print loop.

The Print dialog shows the current settings in the print settings object. If the user changes these settings and accepts the dialog, the print settings object is updated by the printing system with the user's selections. If the user cancels the dialog, the print settings object is returned unchanged.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

CarbonSketch

Declared In

`PMApplication.h`

PMSessionPrintDialogInit

Initializes a custom Print dialog. (Deprecated in Mac OS X v10.4. Use a printing dialog extension instead.)

```
OSStatus PMSessionPrintDialogInit (
   PMPrintSession printSession,
   PMPrintSettings printSettings,
   PMPageFormat constPageFormat,
   PMDialog *newDialog
);
```

Parameters

printSession

The current printing session.

printSettings

A print settings object.

constPageFormat

A page format object.

newDialog

On return, a pointer to an initialized `PMDialog` object, ready for customization by your application. Because the `PMSessionPrintDialogMain` (page 2092) function does not include a parameter for passing this `PMDialog` object to your dialog initialization callback function, your application should store this pointer in a global variable or as extended data in the `PMPrintSettings` object. See the discussion of the `PMPrintDialogInitProcPtr` (page 2102) callback function for more information.

Return Value

A result code. See Core Printing Result Codes.

Discussion

The function `PMSessionPrintDialogInit` is valid only between the creation and release of a printing session and after creating a print settings object. You should call this function only if you create a custom dialog using the `AppendDITL` method. In Mac OS X, you should create a custom dialog by writing a printing dialog extension. If you use this function to extend a printing dialog in Mac OS X, you cannot use sheets.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`PMApplicationDeprecated.h`

PMSessionPrintDialogMain

Displays your application's custom Print dialog. (Deprecated in Mac OS X v10.4. Use a printing dialog extension instead.)

```
OSStatus PMSessionPrintDialogMain (
   PMPrintSession printSession,
   PMPrintSettings printSettings,
   PMPageFormat constPageFormat,
   Boolean *accepted,
   PMPrintDialogInitUPP myInitProc
);
```

Parameters

printSession

The current printing session.

printSettings

A print settings object.

constPageFormat

A page format object.

accepted

A pointer to your Boolean variable. On return, `true` if the user accepts the dialog or `false` if the user cancels the dialog.

myInitProc

A universal procedure pointer to your dialog initialization function. Your initialization function is defined by the callback `PMPrintDialogInitProcPtr`

Return Value

A result code. See Core Printing Result Codes.

Discussion

The function `PMSessionPrintDialogMain` is valid only between the creation and release of a printing session and after calling the function `PMSessionPrintDialogInit`. You should call this function only if you create a custom dialog using the `AppendDITL` method. In Mac OS X, you should create a custom dialog by writing a printing dialog extension. If you use this function to extend a printing dialog in Mac OS X, you cannot use sheets.

Your dialog initialization function is called before your custom Print dialog is displayed. Your initialization function can append items to the Print dialog, and should install an item proc using the `PMSetItemProc` (page 2096) function. You must pass the same page format and print settings objects to each of the functions `PMSessionPrintDialogMain` and `PMSessionPrintDialogInit` (page 2091).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`PMApplicationDeprecated.h`

PMSessionUseSheets

Specifies that a printing dialog should be displayed as a sheet and specifies a function to call when the user dismisses the printing dialog.

```
OSStatus PMSessionUseSheets (
   PMPrintSession printSession,
   WindowRef documentWindow,
   PMSheetDoneUPP sheetDoneProc
);
```

Parameters

printSession

The printing session for which you want to display the dialog.

documentWindow

The window to which the sheet dialog should be attached.

sheetDoneProc

A universal procedure pointer to your sheet-done function. The printing system calls this function when the user dismisses the dialog. Your sheet-done function is defined by the callback type `PMSheetDoneProcPtr` (page 2103).

Return Value

A result code. See Core Printing Result Codes.

Discussion

You must call this function between the creation and release of a printing session. See the function `PMCreateSession`.

When a printing dialog is displayed as a sheet, calls to the functions `PMSessionPageSetupDialog` (page 2088) and `PMSessionPrintDialog` (page 2091) return control immediately to your application, with the value of the `accepted` parameter undefined. The printing dialog sheet continues to be displayed until the user dismisses the dialog, at which time the printing system calls your sheet-done function. The page format and print settings objects associated with the dialog should not be used or changed until the sheet-done function is called.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`PMApplication.h`

PMSetDialogAccepted

Indicates that the user confirmed a custom dialog. (Deprecated in Mac OS X v10.4. Use a printing dialog extension instead.)

```
OSStatus PMSetDialogAccepted (
   PMDialog pmDialog,
   Boolean process
);
```

Parameters

pmDialog

A `PMDialog` object representing your customized Page Setup or Print dialog.

process

Pass `true` if the user confirms the dialog. In the case of the Page Setup dialog, this means the user clicked the OK button; in the case of the Print dialog, this means the user clicked the Print button. Pass `false` if the user clicks the Cancel button.

Return Value
A result code. See Core Printing Result Codes.

Discussion
The function `PMSetDialogAccepted` is valid between the creation and release of a printing session and while displaying your custom Page Setup or Print dialog.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`PMApplicationDeprecated.h`

PMSetDialogDone

Indicates that the user finished with a custom dialog. (Deprecated in Mac OS X v10.4. Use a printing dialog extension instead.)

```
OSStatus PMSetDialogDone (
   PMDialog pmDialog,
   Boolean done
);
```

Parameters

pmDialog

A `PMDialog` object representing your custom Page Setup or Print dialog.

done

Pass `true` when your callback function has finished with this dialog.

Return Value
A result code. See Core Printing Result Codes.

Discussion
The function `PMSetDialogDone` is valid between the creation and release of a printing session and while displaying your custom Page Setup or Print dialog.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`PMApplicationDeprecated.h`

PMSetItemProc

Installs an item proc callback function for items in a custom dialog. (Deprecated in Mac OS X v10.4. Use a printing dialog extension instead.)

```
OSStatus PMSetItemProc (
   PMDialog pmDialog,
   PMItemUPP itemProc
);
```

Parameters

pmDialog

A `PMDialog` object representing your custom Page Setup or Print dialog.

itemProc

A universal procedure pointer to your dialog item event handling function.

Return Value

A result code. See Core Printing Result Codes.

Discussion

The function `PMSetItemProc` is valid between the creation and release of a printing session and after creating your custom Page Setup or Print dialog.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`PMApplicationDeprecated.h`

PMSetModalFilterProc

Installs an event handling callback function for a modal dialog. (Deprecated in Mac OS X v10.4. Use a printing dialog extension instead.)

```
OSStatus PMSetModalFilterProc (
   PMDialog pmDialog,
   ModalFilterUPP filterProc
);
```

Parameters

pmDialog

A `PMDialog` object representing the Page Setup or Print dialog for which you want to install an event handling function.

filterProc

A universal procedure pointer to the event handling function you want called while the Page Setup or Print dialog is displayed. In Mac OS X this function is not called.

Return Value

A result code. See Core Printing Result Codes.

Discussion
The function `PMSetModalFilterProc` is valid between the creation and release of a printing session. You must install your filter procedure before calling `PMSessionPageSetupDialogMain` or `PMSessionPrintDialogMain`.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`PMApplicationDeprecated.h`

PMShowPageSetupDialogAsSheet

Displays a Page Setup dialog as a sheet attached to a window.

```
OSStatus PMShowPageSetupDialogAsSheet (
    PMPrintSession printSession,
    PMPageFormat pageFormat,
    WindowRef documentWindow,
    PMSheetDoneUPP sheetDoneProc
);
```

Parameters

printSession
The printing session for which you want to display the dialog.

pageFormat
The initial page format settings.

documentWindow
The window to which the Page Setup dialog sheet should be attached.

sheetDoneProc
A pointer to a sheet-done callback function that you provide. Your function is called when the user dismisses the dialog. See `PMSheetDoneProcPtr` (page 2103) for information about how to implement this callback.

Return Value
A result code. See Core Printing Result Codes.

Discussion
This function is asynchronous; it displays the dialog in a sheet and returns immediately. When the user dismisses the dialog, the printing system calls the function specified in the *sheetDoneProc* parameter, passing it a Boolean value that indicates whether the user pressed the OK button.

Special Considerations

In Mac OS X v10.5 and later, you should use this function if you want to display the Page Setup dialog as a document-modal sheet. This function replaces the older method of calling `PMSessionUseSheets` (page 2093) and `PMSessionPageSetupDialog` (page 2088).

Availability
Available in Mac OS X v10.5 and later.
Not available to 64-bit applications.

Declared In
PMApplication.h

PMShowPrintDialogWithOptions

Displays a Print dialog with a specified set of controls.

```
OSStatus PMShowPrintDialogWithOptions (
    PMPrintSession printSession,
    PMPrintSettings printSettings,
    PMPageFormat pageFormat,
    PMPrintDialogOptionFlags printDialogOptions,
    Boolean *accepted
);
```

Parameters

printSession
The printing session for which you want to display the dialog.

printSettings
The initial print settings for the Print dialog.

pageFormat
The initial page format settings for the Print dialog.

printDialogOptions
Flags that specify what controls are displayed in the expanded version of the Print dialog. See "Print Dialog Options" (page 2106) for a description of the flags you can use to specify the dialog options.

accepted
A pointer to a Boolean value. On return, `true` if the user accepts the dialog or `false` if the user cancels the dialog.

Return Value
A result code. See Core Printing Result Codes.

Discussion
In Mac OS X v10.5 and later, you can use this function to display the Print dialog with an optional set of controls in the expanded version of the dialog. When the Print dialog is first displayed, the user sees a simplified dialog with a disclosure button. If the button is pressed, the dialog expands and includes the controls specified in the *printDialogOptions* parameter. This function only allows you to specify controls for which flags are defined. The expanded dialog may also contain other controls that are not optional.

When you call this function, you should be prepared for the possibility that the page format settings may change. The expanded Print dialog could allow the user to modify page format settings explicitly. Even if the dialog is displayed with the default set of options, there may be other controls in the Print dialog that cause the page format settings to change.

This function is synchronous; it doesn't return until the user dismisses the dialog.

Availability
Available in Mac OS X v10.5 and later.
Not available to 64-bit applications.

Declared In
PMApplication.h

PMShowPrintDialogWithOptionsAsSheet

Displays a Print dialog with a specified set of controls as a sheet attached to a window.

```
OSStatus PMShowPrintDialogWithOptionsAsSheet (
    PMPrintSession printSession,
    PMPrintSettings printSettings,
    PMPageFormat pageFormat,
    PMPrintDialogOptionFlags printDialogOptions,
    WindowRef documentWindow,
    PMSheetDoneUPP sheetDoneProc
);
```

Parameters

printSession
The printing session for which you want to display the dialog.

printSettings
The initial print settings for the Print dialog.

pageFormat
The initial page format settings for the Print dialog.

printDialogOptions
Flags that specify what controls are displayed in the expanded version of the Print dialog. See "Print Dialog Options" (page 2106) for a description of the flags you can use to specify dialog options.

documentWindow
The window to which the Print dialog sheet should be attached.

sheetDoneProc
A pointer to a sheet-done callback function that you provide. Your function is called when the user dismisses the dialog. See `PMSheetDoneProcPtr` (page 2103) for information about how to implement this callback.

Return Value

A result code. See Core Printing Result Codes.

Discussion

In Mac OS X v10.5 and later, you can use this function to display the Print dialog with an optional set of controls in the expanded version of the dialog. When the Print dialog is first displayed, the user sees a simplified dialog with a disclosure button. If the button is pressed, the dialog expands and includes the controls specified in the *printDialogOptions* parameter. This function only allows you to specify controls for which flags are defined. The expanded dialog may also contain other controls that are not optional.

When you call this function, you should be prepared for the possibility that the page format settings may change. The expanded Print dialog could allow the user to modify page format settings explicitly. Even if the dialog is displayed with the default set of options, there may be other controls in the Print dialog that cause the page format settings to change.

This function is asynchronous; it displays the dialog in a sheet and returns immediately. When the user dismisses the dialog, the printing system calls the function specified in the *sheetDoneProc* parameter, passing it a Boolean value that indicates whether the user accepted or canceled the dialog.

Special Considerations

In Mac OS X v10.5 and later, you should use this function if you want to display the Print dialog as a document-modal sheet. This function replaces the older method of calling `PMSessionUseSheets` (page 2093) and `PMSessionPrintDialog` (page 2091).

Availability
Available in Mac OS X v10.5 and later.
Not available to 64-bit applications.

Declared In
`PMApplication.h`

Callbacks by Task

Sheet Dialog Callback

`PMSheetDoneProcPtr` (page 2103)
Defines a pointer to a sheet-done function. The function is called when the user dismisses a printing dialog presented as a sheet.

Legacy Callbacks

`PMPageSetupDialogInitProcPtr` (page 2101)
Defines a pointer to a dialog initialization function for your custom Page Setup dialog. (Deprecated. Use a printing dialog extension instead.)

`PMPrintDialogInitProcPtr` (page 2102)
Defines a pointer to a dialog initialization function for your custom Print dialog. (Deprecated. Use a printing dialog extension instead.)

`PMItemProcPtr` (page 2100)
Defines a pointer to a dialog item proc function that handles items you add to your custom Page Setup or Print dialog. (Deprecated. Use a printing dialog extension instead.)

Callbacks

PMItemProcPtr

Defines a pointer to a dialog item proc function that handles items you add to your custom Page Setup or Print dialog. (**Deprecated.** Use a printing dialog extension instead.)

```
typedef void (*PMItemProcPtr) (
    DialogRef theDialog,
    SInt16 item
);
```

You would declare your event handling function like this if you were to name it `MyPMItemCallback`:

```
void MyPMItemCallback (
    DialogRef theDialog,
    SInt16 item
);
```

Parameters

theDialog

A reference to your customized Page Setup or Print dialog (`PMDialog`). You can use the function `PMGetDialogPtr` to get a dialog reference.

item

The number of the dialog item.

Discussion

You should provide an item proc callback function if you add items to a custom dialog you create using the `AppendDITL` method. In Mac OS X, you should create a custom dialog by writing a printing dialog extension.

Your function is called by the printing system after you create a custom Page Setup or Print dialog and register your dialog item proc function by calling the function `PMSetItemProc`. In Mac OS X, your function is called for items you add to your custom Page Setup or Print dialog. Your item proc function should call the function `PMSessionSetError` if the user cancels, otherwise the printing system won't know about the user cancellation.

To provide a pointer to your item proc function, you create a universal procedure pointer (UPP) of type `PMItemUPP`, using the function `NewPMItemUPP`. You can do so with code similar to the following:

```
PMItemUPP MyPMItemCallbackUPP;
MyPMItemCallbackUPP = NewPMPItemUPP (&MyPMItemCallback);
```

After you are finished with your item proc function, you can dispose of the UPP with the function `DisposePMItemUPP`. However, if you will use the same event handling function for subsequent displays of a printing dialog, you can reuse the same UPP, rather than dispose of it and later create a new UPP.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`PMApplicationDeprecated.h`

PMPageSetupDialogInitProcPtr

Defines a pointer to a dialog initialization function for your custom Page Setup dialog. (**Deprecated.** Use a printing dialog extension instead.)

```
typedef void (*PMPageSetupDialogInitProcPtr) (
    PMPageFormat pageFormat,
    PMDialog *theDialog
);
```

You would declare your dialog initialization function like this if you were to name it `MyPageSetupDialogInitCallback`:

```
void MyPageSetupDialogInitCallback (
    PMPageFormat pageFormat,
    PMDialog *theDialog
);
```

Parameters

pageFormat

A `PMPageFormat` object.

theDialog

A pointer to the `PMDialog` object representing your custom Page Setup dialog. You obtain this object from the function `PMSessionPageSetupDialogInit`. Your application should store a pointer to the dialog object in a global variable or as extended data in the `PMPageFormat` object, where it is accessible by your initialization function.

Discussion

You should set up a dialog initialization callback only if you need to use a custom dialog you create using the `AppendDITL` method. In Mac OS X, you should create a custom dialog by writing a printing dialog extension. Your initialization function can append items to the Page Setup dialog. If you use this function to extend a printing dialog in Mac OS X, you cannot use sheets.

You pass a pointer to your dialog initialization function as a parameter to the function `PMSessionPageSetupDialogMain`. Your dialog initialization function is called before your custom Page Setup dialog is displayed. Your initialization function can append items to the Page Setup dialog, and should install an item proc using the function `PMSetitemProc`.

To provide a pointer to your dialog initialization function, you create a universal procedure pointer (UPP) of type `PMPageSetupDialogInitUPP`, using the function `NewPMPageSetupDialogInitUPP`. You can do so with code similar to the following:

```
PMPageSetupDialogInitUPP MyPageSetupDialogInitCallbackUPP;
MyPageSetupDialogInitCallbackUPP =      NewPMPageSetupDialogInitUPP
(MyPageSetupDialogInitCallback);
```

After you are finished with your dialog initialization function, you can dispose of the UPP with the function `DisposePMPageSetupDialogInitUPP`. However, if you plan to use the same dialog initialization function for subsequent display of the Page Setup dialog, you can reuse the same UPP, rather than dispose of it and later create a new UPP.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`PMApplicationDeprecated.h`

PMPrintDialogInitProcPtr

Defines a pointer to a dialog initialization function for your custom Print dialog. (**Deprecated.** Use a printing dialog extension instead.)

```
typedef void (*PMPrintDialogInitProcPtr) (
    PMPrintSettings printSettings,
    PMDialog *theDialog
);
```

You would declare your dialog initialization function like this if you were to name it `MyPrintDialogInitCallback`:

```
void MyPrintDialogInitCallback (
```

```
    PMPrintSettings printSettings,
    PMDialog *theDialog
);
```

Parameters

printSettings

A `PMPrintSettings` object.

theDialog

A pointer to the `PMDialog` object representing your custom Print dialog. You obtain this object from the function `PMSessionPrintDialogInit`. Your application should store a pointer to the dialog object in a global variable or as extended data in the `PMPrintSettings` object, where it is accessible by your initialization function.

Discussion

You should set up a dialog initialization callback if you need to use a custom dialog you create using the `AppendDITL` method. In Mac OS X, you should create a custom dialog by writing a printing dialog extension. Your initialization function can append items to the Print dialog. If you use this function to extend a printing dialog in Mac OS X, you cannot use sheets.

You pass a pointer to your dialog initialization function as a parameter to the function `PMSessionPrintDialogMain`. Your dialog initialization function is called before your custom Print dialog is displayed. Your initialization function can append items to the Print dialog, and should install an item proc using the function `PMSetitemProc`.

To provide a pointer to your dialog initialization function, you create a universal procedure pointer (UPP) of type `PMPrintDialogInitUPP`, using the function `NewPMPrintDialogInitUPP`. You can do so with code similar to the following:

```
PMPrintDialogInitUPP MyPrintDialogInitCallbackUPP;
MyPrintDialogInitCallbackUPP = NewPMPrintDialogInitUPP
(&MyPrintDialogInitCallback);
```

After you are finished with your dialog initialization function, you can dispose of the UPP with the function `DisposePMPrintDialogInitUPP`. However, if you will use the same dialog initialization function for subsequent displays of the Print dialog, you can reuse the same UPP, rather than dispose of it and later create a new UPP.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`PMApplicationDeprecated.h`

PMSheetDoneProcPtr

Defines a pointer to a sheet-done function. The function is called when the user dismisses a printing dialog presented as a sheet.

```
typedef void (*PMSheetDoneProcPtr) (
    PMPrintSession printSession,
    WindowRef documentWindow,
    Boolean accepted
);
```

You would declare your sheet-done function like this if you were to name it `MyPrintSheetDoneCallback`:

```
void MyPrintSheetDoneCallback (
    PMPrintSession printSession,
    WindowRef documentWindow,
    Boolean accepted
);
```

Parameters

printSession

The printing session object for the dialog.

documentWindow

The document window to which the sheet is attached.

accepted

A `Boolean` value indicating whether the user accepted (`true`) or canceled (`false`) the dialog.

Discussion

You pass a universal procedure pointer to your sheet-done callback function as a parameter to the functions `PMSessionUseSheets` (page 2093), `PMShowPrintDialogWithOptionsAsSheet` (page 2099), and `PMShowPageSetupDialogAsSheet` (page 2097). Your sheet-done function is called when the user dismisses the dialog. If the user accepted the dialog, your function should take appropriate action depending on whether the Page Setup or Print dialog is being shown.

To provide a pointer to your sheet-done function, you create a universal procedure pointer (UPP) of type `PMSheetDoneUPP`, using the function `NewPMSheetDoneUPP`. You can do so with code similar to the following:

```
PMSheetDoneUPP gMyPrintSheetDoneUPP;
gMyPrintSheetDoneUPP = NewPMSheetDoneUPP (&MyPrintSheetDoneProc);
```

You should declare your universal procedure pointer as a global variable to allow for multiple windows, each with a dialog available at one time.

When your print job is completed, you should use the function `DisposePMSheetDoneUPP` function to dispose of the universal procedure pointer associated with your sheet-done function. However, if you plan to use the same sheet-done function in subsequent print jobs, you can reuse the same UPP, rather than dispose of it and later create a new UPP.

The sheet-done function does not have a parameter for application-supplied data. Instead, you can attach any data your application needs to the window as a property using the Window Manager function `SetWindowProperty` and retrieve the data using the function `GetWindowProperty`.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`PMApplication.h`

Data Types

PMItemUPP

A type that defines a universal procedure pointer to an item proc callback.

```
typedef PMItemProcPtr PMItemUPP;
```

Discussion
This data type is used by functions that are deprecated. For more information, see the description of the `PMItemProcPtr` (page 2100) callback function.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`PMApplicationDeprecated.h`

PMPageSetupDialogInitUPP

A type that defines a universal procedure pointer to a Page Setup dialog initialization callback.

```
typedef PMPageSetupDialogInitProcPtr PMPageSetupDialogInitUPP;
```

Discussion
This data type is used by functions that are deprecated. For more information, see the description of the `PMPageSetupDialogInitProcPtr` (page 2101) callback function.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`PMApplicationDeprecated.h`

PMPrintDialogInitUPP

A type that defines a universal procedure pointer to a Print dialog initialization callback.

```
typedef PMPrintDialogInitProcPtr PMPrintDialogInitUPP;
```

Discussion
This data type is used by functions that are deprecated. For more information, see the description of the `PMPrintDialogInitProcPtr` (page 2102) callback function.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
PMApplicationDeprecated.h

PMSheetDoneUPP

A type that defines a universal procedure pointer to a sheet-done callback function.

```
typedef PMSheetDoneProcPtr PMSheetDoneUPP;
```

Discussion
For more information, see the description of the PMSheetDoneProcPtr (page 2103) callback function.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
PMApplication.h

Constants

Print Dialog Options

Flags that specify items to show in the expanded Print dialog in Mac OS X v10.5 and later.

```
typedef OptionBits PMPrintDialogOptionFlags;
enum {
    kPMHideInlineItems = (0L << 0),
    kPMShowDefaultInlineItems = (1L << 15),
    kPMShowInlineCopies = (1L << 0),
    kPMShowInlinePageRange = (1L << 1),
    kPMShowInlinePageRangeWithSelection = (1L << 6),
    kPMShowInlinePaperSize = (1L << 2),
    kPMShowInlineOrientation = (1L << 3),
    kPMShowInlineScale = (1L << 7),
    kPMShowPageAttributesPDE = (1L << 8),
};
```

Constants

kPMHideInlineItems
Show nothing in the inline portion of the Print dialog.
Available in Mac OS X v10.5 and later.
Not available to 64-bit applications.
Declared in PMApplication.h.

kPMShowDefaultInlineItems
Show the default set of items (copies & page range) in the inline portion of the Print dialog.
Available in Mac OS X v10.5 and later.
Not available to 64-bit applications.
Declared in PMApplication.h.

`kPMShowInlineCopies`
: Show the copies edit-text field, the collated check box, and the two-sided check box (if the printer supports it) in the top portion of the Print dialog.

  Available in Mac OS X v10.5 and later.

  Not available to 64-bit applications.

  Declared in `PMApplication.h`.

`kPMShowInlinePageRange`
: Show the page range radio buttons and the from & to page edit-text fields in the top portion of the Print dialog.

  Available in Mac OS X v10.5 and later.

  Not available to 64-bit applications.

  Declared in `PMApplication.h`.

`kPMShowInlinePageRangeWithSelection`
: Show the page range radio buttons with the addition of a selection button and the From & To Page edit-text fields in the top portion of the Print dialog.

  Available in Mac OS X v10.5 and later.

  Not available to 64-bit applications.

  Declared in `PMApplication.h`.

`kPMShowInlinePaperSize`
: Show the formatting paper size popup menu in the top portion of the Print dialog.

  Available in Mac OS X v10.5 and later.

  Not available to 64-bit applications.

  Declared in `PMApplication.h`.

`kPMShowInlineOrientation`
: Show the page orientation buttons in the top portion of the Print dialog.

  Available in Mac OS X v10.5 and later.

  Not available to 64-bit applications.

  Declared in `PMApplication.h`.

`kPMShowInlineScale`
: Show the scale edit-text field in the top portion of the Print dialog.

  Available in Mac OS X v10.5 and later.

  Not available to 64-bit applications.

  Declared in `PMApplication.h`.

`kPMShowPageAttributesPDE`
: Add a separate pane to the Print dialog that contains the Page Setup dialog information (paper size, orientation, and scale).

  Available in Mac OS X v10.5 and later.

  Not available to 64-bit applications.

  Declared in `PMApplication.h`.

Declared In
`PMApplication.h`

Data Browser Reference

Framework:	Carbon/Carbon.h
Declared in	Controls.h ControlDefinitions.h

Overview

The data browser application programming interface (API) provides a convenient way to present data for browsing and to create easily customized lists whose columns can be sorted, moved, and resized. It supports two presentation styles, each of which is derived from an abstract table-view base class:

- **List view**, which lets you present items in multiple columns with the option to create hierarchical lists whose contents can be disclosed by the user
- **Column view**, which provides in-place browsing using fixed navigation columns

The data browser programming interface has some routines that apply to both views while others are unique to one view. For functions that can be called for either, there may be differences in how the functions operate. Such differences are noted in the documentation for individual functions.

These terms are essential to understanding the reference:

- An **item** in a data browser refers to the data displayed at a particular row and column intersection. In list view, two values identify each item—an item ID and a property ID. In column view, one value—the item ID—uniquely identifies an item.
- An **item ID** is a unique 32-bit ID number that your application uses to refer to data. When you ask the data browser to display one or more items, you provide an item ID for each data item. You can store the actual data in memory, on disk, or across a network. Item IDs must be greater than `0`, which is used internally by the data browser. Item IDs can be values such as pointer values, data file offsets, and 32-bit TCP/IP host addresses.
- A **Property ID** is a non-zero, 32-bit unsigned integer value that uniquely identifies a list view column. Property IDs do not need to be ordered or sequential, but they cannot be values 0 through 1023 because those values are reserved by Apple. A property ID is typically defined as a four-character sequence. For example, a column that displays dates could be assigned the property ID `DATE`.

  Columns in column view don't use application-defined property IDs. Instead, columns in column view have the predefined property `kDataBrowserItemSelfIdentityProperty`.

After you've created, formatted, and configured a data browser, most of the work of keeping the data browser updated and responsive to user interaction happens through callbacks you provide. For example, all of the functions that get and set item data are called from within an item-data callback provided by your application. Your application has a wide latitude in what it can choose to handle through callbacks and the tasks it lets

the system perform. At the very least, your application must provide an item-data callback. Otherwise no data will ever be written to the data browser user interface. Depending on the nature of your application, you may also want to provide callbacks to handle drag-and-drop behavior, to support contextual menus, and to perform custom drawing or some other custom behavior.

The data browser is available with CarbonLib 1.1 and later and in Mac OS X.

For conceptual information and instructions on how to write code that uses a data browser to display data, see *Data Browser Programming Guide*.

Functions by Task

Creating and Configuring a Data Browser

`CreateDataBrowserControl` (page 2125)
Creates a data browser programmatically.

`SetDataBrowserViewStyle` (page 2230)
Sets the view style of the specified data browser.

`GetDataBrowserViewStyle` (page 2174)
Obtains the current view style for the specified data browser.

Manipulating Data Browser Attributes

`GetDataBrowserUserState` (page 2173)
Obtains the current view style settings for a list view.

`SetDataBrowserUserState` (page 2230)
Restores the view-style settings in list view to a previous state set by the user.

`SetDataBrowserActiveItems` (page 2199)
Sets what determines the active state of the items in a data browser.

`GetDataBrowserActiveItems` (page 2139)
Obtains what determines the active state of the items in a data browser.

`SetDataBrowserScrollBarInset` (page 2220)
Sets the inset values to use for the scroll bars of a data browser.

`GetDataBrowserScrollBarInset` (page 2162)
Obtains the inset rectangle used by a data browser to position the scroll bar.

`SetDataBrowserTarget` (page 2229)
Sets the target for a data browser.

`GetDataBrowserTarget` (page 2172)
Obtains the target for the data browser

`SetDataBrowserSortOrder` (page 2223)
Sets the sorting order for a list in list view.

`GetDataBrowserSortOrder` (page 2164)
Gets the sorting order of the list view column that's currently set for sorting.

SetDataBrowserScrollPosition (page 2221)
Scrolls a list to the specified position.

GetDataBrowserScrollPosition (page 2162)
Obtains the scrolling position of a list.

SetDataBrowserHasScrollBars (page 2206)
Sets the display state of horizontal and vertical scroll bars for a list view data browser.

GetDataBrowserHasScrollBars (page 2144)
Obtains the display state of horizontal and vertical scroll bars for a list view data browser.

SetDataBrowserSortProperty (page 2223)
Designates the list view column to use for sorting.

GetDataBrowserSortProperty (page 2165)
Obtains the property ID of the column currently used for sorting in list view.

SetDataBrowserSelectionFlags (page 2222)
Sets allowable selection behavior for a data browser.

GetDataBrowserSelectionFlags (page 2164)
Obtains the current selection behavior for a data browser.

SetDataBrowserPropertyFlags (page 2219)
Sets the appearance and behavior attributes for a column in list view.

GetDataBrowserPropertyFlags (page 2161)
Obtains the appearance and behavior attributes for a column.

SetDataBrowserEditText (page 2205)
Modifies the displayed contents of a text item while it is being edited.

CopyDataBrowserEditText (page 2125)
Copies the text being edited by the user.

GetDataBrowserEditText (page 2143)
Obtains the text being edited by the user.

SetDataBrowserEditItem (page 2204)
Programmatically starts or ends an editing session.

GetDataBrowserEditItem (page 2143)
Obtains the item ID and property ID values of the current editing session.

GetDataBrowserItemPartBounds (page 2156)
Obtains the bounds of a visual part of an item.

Setting Up and Installing Callbacks

InitDataBrowserCallbacks (page 2175)
Initializes a data browser callback structure in preparation for adding your own callbacks to the structure.

SetDataBrowserCallbacks (page 2200)
Sets the callback routines to use with a data browser, replacing any previously installed callbacks.

GetDataBrowserCallbacks (page 2140)
Obtains the callback routines installed for notifying your application of changes to a data browser and for providing the data to be displayed by the data browser.

`InitDataBrowserCustomCallbacks` (page 2176)
Initializes the data browser custom callback structure in preparation for adding your own callbacks for custom drawing or custom behavior to the structure.

`SetDataBrowserCustomCallbacks` (page 2203)
Sets the custom callback routines to use with a data browser, replacing any previously installed custom callbacks.

`GetDataBrowserCustomCallbacks` (page 2142)
Obtains the callbacks installed to implement custom drawing and behavior for the content in a data browser.

Formatting Table View

Table view is a base class from which list and column views are derived. Some functions in this group can be used with both list and column views, while others are useful only in list view.

`RemoveDataBrowserTableViewColumn` (page 2198)
Removes a column from a list view data browser.

`GetDataBrowserTableViewColumnCount` (page 2166)
Obtains the number of columns in a data browser.

`SetDataBrowserTableViewHiliteStyle` (page 2226)
Sets the highlighting style to use for a list view data browser.

`GetDataBrowserTableViewHiliteStyle` (page 2169)
Obtains the highlighting style used for a list view data browser.

`SetDataBrowserTableViewRowHeight` (page 2228)
Sets the default row height for all rows in a data browser.

`GetDataBrowserTableViewRowHeight` (page 2172)
Obtains the default row height used for all rows in a data browser.

`SetDataBrowserTableViewColumnWidth` (page 2225)
Sets the default column width for all columns in a data browser.

`GetDataBrowserTableViewColumnWidth` (page 2167)
Obtains the default column width used for all columns in a data browser.

`SetDataBrowserTableViewItemRowHeight` (page 2227)
Sets the row height for a single row in a list view data browser.

`GetDataBrowserTableViewItemRowHeight` (page 2170)
Obtains the row height for a single row in a list view data browser.

`SetDataBrowserTableViewNamedColumnWidth` (page 2228)
Sets the column width for a single column in a list view data browser.

`GetDataBrowserTableViewNamedColumnWidth` (page 2171)
Obtains the column width for a single column in a data browser.

`SetDataBrowserTableViewGeometry` (page 2225)
Sets whether columns and rows can have variable widths in list view.

`GetDataBrowserTableViewGeometry` (page 2168)
Determines whether columns and rows are set to have variable widths.

`GetDataBrowserTableViewItemID` (page 2169)
Obtains the item ID for the item displayed in the specified row.

`SetDataBrowserTableViewItemRow` (page 2226)
Changes the visual position for an item in a list view data browser.

`GetDataBrowserTableViewItemRow` (page 2170)
Obtains the visual position for the specified item in list view.

`SetDataBrowserTableViewColumnPosition` (page 2224)
Changes the visual position of a column in list view.

`GetDataBrowserTableViewColumnPosition` (page 2166)
Obtains the column position for an item in a data browser.

`GetDataBrowserTableViewColumnProperty` (page 2167)
Obtains the property ID for a column in a data browser.

Formatting List View

`AutoSizeDataBrowserListViewColumns` (page 2122)
Adjusts the size of columns displayed in list view to take best advantage of the available space.

`AddDataBrowserListViewColumn` (page 2121)
Adds a column to a data browser that uses list view.

`GetDataBrowserListViewHeaderDesc` (page 2160)
Obtains a header description for a column in list view.

`SetDataBrowserListViewHeaderDesc` (page 2218)
Provides a description for a column title in list view.

`SetDataBrowserListViewHeaderBtnHeight` (page 2217)
Sets the height of the rectangular area where the column title appears.

`GetDataBrowserListViewHeaderBtnHeight` (page 2159)
Obtains the height of the rectangular area where the column title appears.

`SetDataBrowserListViewUsePlainBackground` (page 2219)
Specifies whether list view uses a plain white background.

`GetDataBrowserListViewUsePlainBackground` (page 2160)
Determines whether list view is set to use a plain white background.

`SetDataBrowserListViewDisclosureColumn` (page 2216)
Specifies whether there is a column that has disclosure triangles and, if so, which column.

`GetDataBrowserListViewDisclosureColumn` (page 2158)
Obtains the property ID of the column whose items can display a disclosure triangle, and tells whether a disclosed item expands the row or adds rows.

Formatting Column View

`SetDataBrowserColumnViewPath` (page 2202)
Sets a path for a column view.

`GetDataBrowserColumnViewPath` (page 2141)
Obtains the current path for a selection in column view.

`GetDataBrowserColumnViewPathLength` (page 2142)
Obtains the length of the current path for a column view.

`SetDataBrowserColumnViewDisplayType` (page 2202)
Sets the display type for a data browser in column view.

`GetDataBrowserColumnViewDisplayType` (page 2140)
Obtains the display type for a column view.

Adding and Removing Data Items

`AddDataBrowserItems` (page 2120)
Adds one or more items to a data browser.

`RemoveDataBrowserItems` (page 2197)
Removes one or more items from a data browser.

`UpdateDataBrowserItems` (page 2232)
Requests a redraw of one or more items in a data browser.

Accessing and Operating on All Items

`GetDataBrowserItems` (page 2157)
Obtains a list of the items that match a specified state; operates on items in the root container or traverses items in the data hierarchy.

`GetDataBrowserItemCount` (page 2145)
Obtains the number of items whose state matches the specified state.

`ForEachDataBrowserItem` (page 2138)
Applies an item-iterator callback routine to each data item that meets the specified criteria.

Accessing and Displaying Individual Items

`IsDataBrowserItemSelected` (page 2186)
Checks to see if a data item is selected.

`GetDataBrowserItemState` (page 2158)
Obtains the state of an item.

`RevealDataBrowserItem` (page 2199)
Scrolls an item into view, optionally bringing a particular part of that item into view.

Selecting and Editing Items

`EnableDataBrowserEditCommand` (page 2137)
Determines whether the data browser is currently able to process a given editing command.

`ExecuteDataBrowserEditCommand` (page 2138)
Executes an editing command.

`GetDataBrowserSelectionAnchor` (page 2163)
Obtains the first and last items in a selection.

`MoveDataBrowserSelectionAnchor` (page 2186)
Moves or extends the current selection.

`SetDataBrowserSelectedItems` (page 2222)
Modifies the current selection by adding items, removing items, or toggling the selection state of items.

Working With Attributes

`DataBrowserGetAttributes` (page 2127)
Gets the attributes of a data browser.

`DataBrowserChangeAttributes` (page 2127)
Sets the attributes for a data browser.

Working With Containers

`OpenDataBrowserContainer` (page 2196)
Opens a data browser container.

`CloseDataBrowserContainer` (page 2124)
Closes a data browser container.

`SortDataBrowserContainer` (page 2231)
Sorts a hierarchical list of items.

Working With Metrics

`DataBrowserGetMetric` (page 2128)
Gets the value of a specified data browser metric.

`DataBrowserSetMetric` (page 2129)
Sets the value of a specified data browser metric.

Getting and Setting Item Data

The functions in this section are called from within an item-data callback routine (`DataBrowserItemDataProcPtr`) provided by your application. The data browser invokes your item-data callback each time your application needs to provide data for the display. Your callback responds by calling the appropriate function from this section.

`SetDataBrowserItemDataIcon` (page 2209)
Specifies the icon to draw.

`GetDataBrowserItemDataIcon` (page 2148)
Obtains the icon drawn for an item.

`SetDataBrowserItemDataText` (page 2214)
Specifies the text to draw.

`GetDataBrowserItemDataText` (page 2154)
Obtains the text entered by the user.

`SetDataBrowserItemDataValue` (page 2215)
Sets the value of an item; useful for such display types as sliders, progress bars, relevance indicators, and pop-up menus.

`GetDataBrowserItemDataValue` (page 2155)
Obtains the value of an item; useful for such display types as sliders, progress bars, relevance indicators, and pop-up menus.

`SetDataBrowserItemDataMinimum` (page 2213)
Specifies the minimum integer value that can be displayed for an item; useful for such display types as sliders, progress bars, relevance indicators, and pop-up menus.

`GetDataBrowserItemDataMinimum` (page 2152)
Obtains the minimum integer value that can be displayed for an item; useful for such display types as sliders, progress bars, relevance indicators, and pop-up menus.

`SetDataBrowserItemDataMaximum` (page 2212)
Specifies the maximum integer value that can be displayed for an item; useful for such display types as sliders, progress bars, relevance indicators, and pop-up menus.

`GetDataBrowserItemDataMaximum` (page 2151)
Obtains the maximum integer value that can be displayed; useful for such display types as sliders, progress bars, relevance indicators, and pop-up menus.

`SetDataBrowserItemDataBooleanValue` (page 2206)
Specifies a Boolean value for an item.

`GetDataBrowserItemDataBooleanValue` (page 2145)
Obtains the Boolean value for an item.

`SetDataBrowserItemDataMenuRef` (page 2213)
Sets the pop-up menu to display.

`GetDataBrowserItemDataMenuRef` (page 2152)
Obtains the pop-up menu displayed.

`SetDataBrowserItemDataRGBColor` (page 2214)
Specifies a color to use when drawing an item.

`GetDataBrowserItemDataRGBColor` (page 2154)
Obtains the color used to draw an item.

`SetDataBrowserItemDataDrawState` (page 2208)
Specifies whether to draw a checkbox in the active or inactive state.

`GetDataBrowserItemDataDrawState` (page 2148)
Determines whether a checkbox is in the active or inactive state.

`SetDataBrowserItemDataButtonValue` (page 2207)
Specifies a checkbox value.

`GetDataBrowserItemDataButtonValue` (page 2146)
Obtains the value for a checkbox.

`SetDataBrowserItemDataIconTransform` (page 2210)
Specifies a transformation to apply to an icon when it is drawn.

`GetDataBrowserItemDataIconTransform` (page 2149)
Obtains the transformation currently used to display an icon.

`SetDataBrowserItemDataDateTime` (page 2208)
Specifies, as a 32-bit value, a date and time value to display.

`GetDataBrowserItemDataDateTime` (page 2147)
Obtains, as a 32-bit value, the date and time value displayed.

`SetDataBrowserItemDataLongDateTime` (page 2211)
Specifies, as a 64-bit value, a date and time value to display.

`GetDataBrowserItemDataLongDateTime` (page 2150)
Obtains, as a 64-bit value, the date and time value displayed.

`SetDataBrowserItemDataItemID` (page 2210)
Communicates a property of an item when that property is another item's ID.

`GetDataBrowserItemDataItemID` (page 2150)
Obtains the item ID for an item whose property is another item's ID.

`GetDataBrowserItemDataProperty` (page 2153)
Obtains the column property ID for the column in which an item resides.

Working With Universal Procedure Pointers

The functions in this section create and dispose of universal procedure pointers (UPPs) to the callbacks you provide to the data browser. For each callback, there is a New, Dispose, and Invoke function. You don't need to use an Invoke function, because the data browser invokes callbacks for you.

The documentation for the UPP functions in this section is boilerplate text—quite repetitive and you can likely skip over it. The more interesting documentation is for the callbacks themselves, which you can find in the section "Data Browser Callbacks" (page 2233).

`NewDataBrowserItemUPP` (page 2194)
Creates a universal procedure pointer to an item-iterator callback function.

`InvokeDataBrowserItemUPP` (page 2183)
Calls an item-iterator callback function.

`DisposeDataBrowserItemUPP` (page 2135)
Disposes of a universal procedure pointer to an item-iterator callback function.

`NewDataBrowserItemDataUPP` (page 2191)
Creates a universal procedure pointer to an item-data callback function.

`InvokeDataBrowserItemDataUPP` (page 2180)
Calls an item-data callback function.

`DisposeDataBrowserItemDataUPP` (page 2133)
Disposes of a universal procedure pointer to an item-data callback function.

`NewDataBrowserItemCompareUPP` (page 2190)
Creates a universal procedure pointer to an item-comparison callback function.

`InvokeDataBrowserItemCompareUPP` (page 2180)
Calls an item-comparison callback function.

`DisposeDataBrowserItemCompareUPP` (page 2132)
Disposes of a universal procedure pointer to an item-comparison callback function.

`NewDataBrowserItemNotificationUPP` (page 2192)
Creates a universal procedure pointer to an item-notification callback function.

`InvokeDataBrowserItemNotificationUPP` (page 2182)
Calls an item-notification callback function.

DisposeDataBrowserItemNotificationUPP (page 2134)
Disposes of a universal procedure pointer to an item-notification callback function.

NewDataBrowserItemNotificationWithItemUPP (page 2193)
Creates a universal procedure pointer to an item-notification-with-data callback function.

InvokeDataBrowserItemNotificationWithItemUPP (page 2182)
Calls an item-notification-with-data callback function.

DisposeDataBrowserItemNotificationWithItemUPP (page 2134)
Disposes of a universal procedure pointer to an item-notification-with-data callback function.

NewDataBrowserAddDragItemUPP (page 2188)
Creates a universal procedure pointer to an add-drag-item callback function.

InvokeDataBrowserAddDragItemUPP (page 2177)
Calls an add-drag-item callback function.

DisposeDataBrowserAddDragItemUPP (page 2130)
Disposes of a universal procedure pointer to an add-drag-item callback function.

NewDataBrowserAcceptDragUPP (page 2187)
Creates a universal procedure pointer to an accept-drag callback function.

InvokeDataBrowserAcceptDragUPP (page 2177)
Calls an accept-drag callback function.

DisposeDataBrowserAcceptDragUPP (page 2129)
Disposes of a universal procedure pointer to an accept-drag callback function.

NewDataBrowserReceiveDragUPP (page 2195)
Creates a universal procedure pointer to a receive-drag callback function.

InvokeDataBrowserReceiveDragUPP (page 2184)
Calls a receive-drag callback function.

DisposeDataBrowserReceiveDragUPP (page 2136)
Disposes of a universal procedure pointer to a receive-drag callback function.

NewDataBrowserPostProcessDragUPP (page 2194)
Creates a universal procedure pointer to a postprocess-drag callback function.

InvokeDataBrowserPostProcessDragUPP (page 2184)
Calls a postprocess-drag callback function.

DisposeDataBrowserPostProcessDragUPP (page 2135)
Disposes of a universal procedure pointer to a postprocess-drag callback function.

NewDataBrowserGetContextualMenuUPP (page 2189)
Creates a universal procedure pointer to a get-contextual-menu callback function.

InvokeDataBrowserGetContextualMenuUPP (page 2178)
Calls a get-contextual-menu callback function.

DisposeDataBrowserGetContextualMenuUPP (page 2131)
Disposes of a universal procedure pointer to a get-contextual-menu callback function.

NewDataBrowserSelectContextualMenuUPP (page 2195)
Creates a universal procedure pointer to a select-contextual-menu callback function.

InvokeDataBrowserSelectContextualMenuUPP (page 2185)
Calls a select-contextual-menu callback function.

DisposeDataBrowserSelectContextualMenuUPP (page 2136)
Disposes of a universal procedure pointer to a select-contextual-menu callback function.

NewDataBrowserItemHelpContentUPP (page 2192)
Creates a universal procedure pointer to an item-help-content callback function.

InvokeDataBrowserItemHelpContentUPP (page 2181)
Calls an item-help-content callback function.

DisposeDataBrowserItemHelpContentUPP (page 2133)
Disposes of a universal procedure pointer to an item-help-content callback function.

NewDataBrowserDrawItemUPP (page 2188)
Creates a universal procedure pointer to a draw-item callback function.

InvokeDataBrowserDrawItemUPP (page 2177)
Calls a draw-item callback function.

DisposeDataBrowserDrawItemUPP (page 2130)
Disposes of a universal procedure pointer to a draw-item callback function.

NewDataBrowserEditItemUPP (page 2188)
Creates a universal procedure pointer to an edit-item callback function.

InvokeDataBrowserEditItemUPP (page 2178)
Calls an edit-item callback function.

DisposeDataBrowserEditItemUPP (page 2131)
Disposes of a universal procedure pointer to an edit-item callback function.

NewDataBrowserHitTestUPP (page 2189)
Creates a universal procedure pointer to a hit-test callback function.

InvokeDataBrowserHitTestUPP (page 2179)
Calls a hit-test callback function.

DisposeDataBrowserHitTestUPP (page 2131)
Disposes of a universal procedure pointer to a hit-test callback function.

NewDataBrowserTrackingUPP (page 2196)
Creates a universal procedure pointer to a tracking callback function.

InvokeDataBrowserTrackingUPP (page 2185)
Calls a tracking callback function.

DisposeDataBrowserTrackingUPP (page 2137)
Disposes of a universal procedure pointer to a tracking callback function.

NewDataBrowserItemDragRgnUPP (page 2191)
Creates a universal procedure pointer to an item-drag-region callback function.

InvokeDataBrowserItemDragRgnUPP (page 2181)
Calls an item-drag-region callback function.

DisposeDataBrowserItemDragRgnUPP (page 2133)
Disposes of a universal procedure pointer to an item-drag-region callback function.

NewDataBrowserItemAcceptDragUPP (page 2190)
Creates a universal procedure pointer to an item-accept-drag callback function.

InvokeDataBrowserItemAcceptDragUPP (page 2179)
Calls an item-accept-drag callback function.

DisposeDataBrowserItemAcceptDragUPP (page 2132)
Disposes of a universal procedure pointer to an item-accept-drag callback function.

NewDataBrowserItemReceiveDragUPP (page 2193)
Creates a universal procedure pointer to an item-receive-drag callback function.

`InvokeDataBrowserItemReceiveDragUPP` (page 2183)
Calls an item-receive-drag callback function.

`DisposeDataBrowserItemReceiveDragUPP` (page 2135)
Disposes of a universal procedure pointer to an item-receive-drag callback function.

Working With AXUIElement References

`AXUIElementCreateWithDataBrowserAndItemInfo` (page 2123)
Creates an `AXUIElementRef` that represents some part of a data browser accessibility hierarchy.

`AXUIElementGetDataBrowserItemInfo` (page 2123)
Obtains a description of the part of a data browser represented by an `AXUIElementRef`.

Functions

AddDataBrowserItems

Adds one or more items to a data browser.

```
OSStatus AddDataBrowserItems (
   ControlRef browser,
   DataBrowserItemID container,
   ItemCount numItems,
   const DataBrowserItemID *items,
   DataBrowserPropertyID preSortProperty
);
```

Parameters

browser
A data browser.

container
An item ID or the constant `kDataBrowserNoItem`. Pass the item ID that uniquely identifies the container to which you want to add items. Adding one or more items to an existing container opens the container. If you a pass `kDataBrowserNoItem`, the items are added to the root container.

numItems
The number of items in the array pointed to by the `items` parameter.

items
A pointer to an array of item ID values for the items you want to add to the data browser. You supply item ID values based on your own identification scheme. If you pass `NULL`, each time you call `AddDataBrowserItems` the data browser generates item ID values starting at `1`. Calling the function in this way clears whatever items are in the container. Because of this clearing behavior, passing `NULL` is not recommended unless your application uses a data browser to display a simple list that is populated only once with data.

preSortProperty
The property ID of the column whose sorting order matches the sorting order of the `items` array. A property ID is a four-character sequence that you assign to represent a column in list view. Pass `kDataBrowserItemNoProperty` if the `items` array is not sorted or if you don't know the sorting order of your data. You'll get the best performance from this function if you provide a sorting order.

Return Value
A result code. If the item ID specified by the `container` parameter is not classified as a container, returns `errDataBrowserItemNotAdded` if you attempt to add subitems to it. See "Data Browser Result Codes" (page 2298).

Discussion
Hierarchical lists are constructed in a top-down fashion. Your application must install all the top-level, or parent, item IDs in the data browser before it associates a list of item ID values as subitems. You can add items to a parent item only after the parent item is classified as a container. A container is an item for which the property `kDataBrowserItemIsContainerProperty` is set to `true`.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

AddDataBrowserListViewColumn

Adds a column to a data browser that uses list view.

```
OSStatus AddDataBrowserListViewColumn (
   ControlRef browser,
   DataBrowserListViewColumnDesc *columnDesc,
   DataBrowserTableViewColumnIndex position
);
```

Parameters

`browser`
A data browser.

`columnDesc`
A pointer to the list view column description data structure that you have filled out with data that specifies the column property and display information for the column heading.

`position`
The position, among the columns already installed in the data browser, to insert this column. To insert this column to the right of all other columns, pass `kDataBrowserListViewAppendColumn`. The value `0` means the leftmost column.

Return Value
A result code; `paramErr` is returned if the `columnDesc` parameter is not properly initialized. See "Data Browser Result Codes" (page 2298).

Discussion
Typically you use the function `AddDataBrowserListViewColumn` in these cases:

- When you create a data browser programmatically. If you use Interface Builder to design and lay out the data browser, you do not need to call the function `AddDataBrowserListViewColumn`. Interface Builder lets you position a column graphically and then specify the column description in the column pane of the Info window.

- When you switch from column view to list view. Regardless of how you first create a data browser, if your application allows the user to switch between views, you need to add list view columns each time the view switches from column to list view.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

AutoSizeDataBrowserListViewColumns

Adjusts the size of columns displayed in list view to take best advantage of the available space.

```
OSStatus AutoSizeDataBrowserListViewColumns (
   ControlRef browser
);
```

Parameters

browser
A data browser.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
When you call the function `AutoSizeDataBrowserListViewColumns`, it first calculates whether there is extra space or not enough space in the data browser. Then, the columns are resized using the following rules:

- If there is extra space, the data browser gives as much of the extra space as possible to the first column (that is, the leftmost column) without exceeding the maximum width for the column. If there is still space available, the data browser gives as much of the remaining space as possible to the second column without exceeding the maximum width for the column. The data browser continues to disburse space in this manner until there is no more extra space. Thus, it is possible for the first column to get all the extra space.
- If space is needed to fit all the columns, the data browser takes as much of the needed space as possible from the rightmost column (that is, the last column) without letting the column width fall below the minimum width for the column. If more space is needed, the data browser takes as much of the needed space as possible from the next-to-the-last column without letting the column width fall below the minimum width for the column. The data browser continues to adjust space in this manner until all the columns fit within the data browser.

The function `AutoSizeDataBrowserListViewColumns` resizes only if the horizontal scroll bar is turned off. Your application can call the function `SetDataBrowserHasScrollBars` (page 2206) to turn off the horizontal scroll bar.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
HIDataBrowser.h

AXUIElementCreateWithDataBrowserAndItemInfo

Creates an AXUIElementRef that represents some part of a data browser accessibility hierarchy.

```
AXUIElementRef AXUIElementCreateWithDataBrowserAndItemInfo (
   ControlRef inDataBrowser,
   const DataBrowserAccessibilityItemInfo *inInfo
);
```

Parameters

inDataBrowser
A data browser.

inInfo
A DataBrowserAccessibilityItemInfo (page 2260) structure describing the part of the data browser for which you want to create an AXUIElementRef.

Return Value
An AXUIElementRef representing the part, or NULL if an AXUIElementRef cannot be created to represent the part you specified.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
HIDataBrowser.h

AXUIElementGetDataBrowserItemInfo

Obtains a description of the part of a data browser represented by an AXUIElementRef.

```
OSStatus AXUIElementGetDataBrowserItemInfo (
   AXUIElementRef inElement,
   ControlRef inDataBrowser,
   UInt32 inDesiredInfoVersion,
   DataBrowserAccessibilityItemInfo *outInfo
);
```

Parameters

inElement
An AXUIElementRef representing part of a data browser.

inDataBrowser
A data browser.

inDesiredInfoVersion
The version of DataBrowserAccessibilityItemInfo (page 2260) structure you want to get. Currently, the only supported version is zero, so you must pass 0 or 1 as the value of this parameter.

outInfo

On input, a pointer to a `DataBrowserAccessibilityItemInfo` (page 2260) structure. On return, the structure is filled in with a description of the part of the data browser that the `AXUIElementRef` specified by `inElement` represents.

Return Value
A result code. See "Data Browser Result Codes" (page 2298). The function returns `noErr` if it was able to generate a description of the `AXUIElementRef`. If the `AXUIElementRef` does not represent the data browser you passed in, the function returns `paramErr`. If the `AXUIElementRef` represents some non-item part of the data browser, the function returns `errDataBrowserItemNotFound`.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

CloseDataBrowserContainer

Closes a data browser container.

```
OSStatus CloseDataBrowserContainer (
   ControlRef browser,
   DataBrowserItemID container
);
```

Parameters

browser

A data browser.

container

The item ID of the container you want to close.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
Normally the user navigates through a data hierarchy by clicking the disclosure triangle next to a container item in list view, or the container item (such as a folder icon) in column view. In either of these cases, the system automatically opens or closes the container. Under some circumstances your application may need to open or close a container programmatically, such as when you are restoring a display to its last known state. In such cases, you can call the function `OpenDataBrowserContainer` (page 2196) to disclose items in a container or the function `CloseDataBrowserContainer` to hide items in a container.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

CopyDataBrowserEditText

Copies the text being edited by the user.

```
OSStatus CopyDataBrowserEditText (
   ControlRef browser,
   CFStringRef *text
);
```

Parameters

browser

A data browser.

text

On input, a `CFStringRef` variable initialized to anything other than `NULL`. See the Special Considerations for details. On return, a CFString object that contains a copy of the text edited by the user. You are responsible for releasing the string when you no longer need it.

Return Value

A result code. See "Data Browser Result Codes" (page 2298).

Discussion

This function is useful only if an edit session is in progress for an item. You can check whether an edit session is open by calling the function `GetDataBrowserEditItem` (page 2143).

Special Considerations

For versions of Mac OS X prior to v10.4, the `text` parameter must be set to any value other than `NULL`. Do not allocate the CFStringRef, otherwise your application will leak memory. Instead provide code similar to the following to initialize the variable:

```
CFStringRef myText = 0XFFFFFFFF;
```

Availability

Available in CarbonLib 1.5 and later.

Available in Mac OS X version 10.0 and later.

Not available to 64-bit applications.

Declared In

`HIDataBrowser.h`

CreateDataBrowserControl

Creates a data browser programmatically.

```
OSStatus CreateDataBrowserControl (
   WindowRef window,
   const Rect *boundsRect,
   DataBrowserViewStyle style,
   ControlRef *outControl
);
```

Parameters

window

The window in which to place the data browser.

boundsRect

A pointer to a rectangle that specifies the location where you want the control to appear in the window.

style

The view style to use. Pass the constant `kDataBrowserListView` to draw the data browser using list view or `kDataBrowserColumnView` draw the data browser using column view. See "View Styles" (page 2298) for more information on these constants.

outControl

On input, a pointer to a control reference. On return, this is set to the newly created data browser. When you no longer need the data browser, call the Control Manager function `DisposeControl` to release it. When you dispose of the control, deallocate any universal procedure pointers you allocated for use with the control.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
This function creates a data browser programmatically. If you create a data browser using Interface Builder, you don't need to call `CreateDataBrowserControl`. Instead, you call the function `GetControlByID` to obtain a control reference that points to your data browser.

After you create a data browser by calling `CreateDataBrowserControl`, you can set such attributes as sorting order, scroll bars, and scroll position. See "Manipulating Data Browser Attributes" (page 2110) for the functions you can use to set data browser attributes.

You need to set up the display characteristics of the data browser by calling the appropriate functions. See "Formatting Table View" (page 2112), "Formatting List View" (page 2113), and "Formatting Column View" (page 2113) for information on the formatting functions you can call.

You need to call the functions `InitDataBrowserCallbacks` (page 2175) and `SetDataBrowserCallbacks` (page 2200) to install the callbacks needed for your data browser. At the very least, you must provide an item-data callback to add or change data items; you must do so regardless of the content your data browser displays—noncustom or custom. Otherwise, your data browser will be empty. See `DataBrowserItemDataProcPtr` (page 2245) for more information. If you present hierarchical data in list view, or use column view for browsing data, you must provide a callback to handle item notifications. See `DataBrowserItemNotificationProcPtr` (page 2250) and `DataBrowserItemNotificationWithItemProcPtr` (page 2251).

You can optionally provide callbacks to:

- Perform sorting. See `DataBrowserItemCompareProcPtr` (page 2243).
- Handle drag-and-drop behavior. See `DataBrowserAddDragItemProcPtr` (page 2234), `DataBrowserAcceptDragProcPtr` (page 2233), `DataBrowserReceiveDragProcPtr` (page 2256), and `DataBrowserPostProcessDragProcPtr` (page 2255).
- Provide contextual menus. See `DataBrowserGetContextualMenuProcPtr` (page 2238) and `DataBrowserSelectContextualMenuProcPtr` (page 2257).
- Display help tags. See `DataBrowserItemHelpContentProcPtr` (page 2248).

If your data browser uses a list whose columns require custom drawing or behavior, you must also provide callbacks to handle the custom tasks. See `InitDataBrowserCustomCallbacks` (page 2176) and `SetDataBrowserCustomCallbacks` (page 2203) for more information on initializing and installing callbacks for custom behavior. The custom tasks you can handle in list view include:

- Drawing custom content. See `DataBrowserDrawItemProcPtr` (page 2235).
- Supporting editing of custom content. See `DataBrowserEditItemProcPtr` (page 2237). Note that editing is built-in for noncustom content.
- Performing hit-testing and tracking. See `DataBrowserHitTestProcPtr` (page 2240) and `DataBrowserTrackingProcPtr` (page 2259).
- Handling drag-and-drop behavior. See `DataBrowserItemDragRgnProcPtr` (page 2246), `DataBrowserItemAcceptDragProcPtr` (page 2242), and `DataBrowserItemReceiveDragProcPtr` (page 2254).

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

DataBrowserChangeAttributes

Sets the attributes for a data browser.

```
OSStatus DataBrowserChangeAttributes (
   ControlRef inDataBrowser,
   OptionBits inAttributesToSet,
   OptionBits inAttributesToClear
);
```

Parameters

inDataBrowser
A data browser.

inAttributesToSet
The attributes to set. For possible values, see "Data Browser Attributes" (page 2273).

inAttributesToClear
The attributes to clear. For possible values, see "Data Browser Attributes" (page 2273).

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

DataBrowserGetAttributes

Gets the attributes of a data browser.

```
OSStatus DataBrowserGetAttributes (
   ControlRef inDataBrowser,
   OptionBits *outAttributes
);
```

Parameters

inDataBrowser

A data browser.

outAttributes

The attributes to get. This parameter cannot be `NULL`. For possible values, see "Data Browser Attributes" (page 2273).

Return Value

A result code. See "Data Browser Result Codes" (page 2298).

Availability

Available in Mac OS X v10.4 and later.

Not available to 64-bit applications.

Declared In

`HIDataBrowser.h`

DataBrowserGetMetric

Gets the value of a specified data browser metric.

```
OSStatus DataBrowserGetMetric (
   ControlRef inDataBrowser,
   DataBrowserMetric inMetric,
   Boolean *outUsingDefaultValue,
   CGFloat *outValue
);
```

Parameters

inDataBrowser

A data browser.

inMetric

The data browser metric value to get. For possible values, see"Data Browser Metric Values" (page 2274).

outUsingDefaultValue

On return, a Boolean whose value indicates whether the metric's value is determined by the data browser's default values. Pass `NULL` if you don't want this information.

outValue

On return, the value of the metric.

Return Value

A result code. See "Data Browser Result Codes" (page 2298). If the `inDataBrowser` is not an instance of a data browser or if the value specified by `inMetric` is not known, `DataBrowserGetMetric` returns `paramErr`.

Availability

Available in Mac OS X v10.4 and later.

Not available to 64-bit applications.

Declared In
HIDataBrowser.h

DataBrowserSetMetric

Sets the value of a specified data browser metric.

```
OSStatus DataBrowserSetMetric (
   ControlRef inDataBrowser,
   DataBrowserMetric inMetric,
   Boolean inUseDefaultValue,
   CGFloat inValue
);
```

Parameters

inDataBrowser
A data browser.

inMetric
The data browser metric whose value is to be set. For possible values, see "Data Browser Metric Values" (page 2274).

inUsingDefaultValue
A Boolean whose value indicates whether you want the data browser to revert to the default value for the metric. If you pass `true`, `inValue` is ignored and a suitable default value is used. If you pass `false`, `inValue` is set as the value of the metric.

inValue
The value to set for the metric (if the value of `inUsingDefaultValue` is `false`).

Return Value
A result code. See "Data Browser Result Codes" (page 2298). If the `inDataBrowser` is not an instance of a data browser or if the value specified by `inMetric` is not known, `DataBrowserSetMetric` returns `paramErr`.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
HIDataBrowser.h

DisposeDataBrowserAcceptDragUPP

Disposes of a universal procedure pointer to an accept-drag callback function.

```
void DisposeDataBrowserAcceptDragUPP (
   DataBrowserAcceptDragUPP userUPP
);
```

Parameters

userUPP
The universal procedure pointer to dispose of.

Discussion
See the `DataBrowserAcceptDragProcPtr` (page 2233) callback function.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.

Declared In
HIDataBrowser.h

DisposeDataBrowserAddDragItemUPP

Disposes of a universal procedure pointer to an add-drag-item callback function.

```
void DisposeDataBrowserAddDragItemUPP (
   DataBrowserAddDragItemUPP userUPP
);
```

Parameters

userUPP
The universal procedure pointer to dispose of.

Discussion
See the DataBrowserAddDragItemProcPtr (page 2234) callback function.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.

Declared In
HIDataBrowser.h

DisposeDataBrowserDrawItemUPP

Disposes of a universal procedure pointer to a draw-item callback function.

```
void DisposeDataBrowserDrawItemUPP (
   DataBrowserDrawItemUPP userUPP
);
```

Parameters

userUPP
The universal procedure pointer to dispose of.

Discussion
See the DataBrowserDrawItemProcPtr (page 2235) callback function.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.1 and later.

Declared In
HIDataBrowser.h

DisposeDataBrowserEditItemUPP

Disposes of a universal procedure pointer to an edit-item callback function.

```
void DisposeDataBrowserEditItemUPP (
   DataBrowserEditItemUPP userUPP
);
```

Parameters

userUPP

The universal procedure pointer to dispose of.

Discussion

See the `DataBrowserEditItemProcPtr` (page 2237) callback function.

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X version 10.1 and later.

Declared In

`HIDataBrowser.h`

DisposeDataBrowserGetContextualMenuUPP

Disposes of a universal procedure pointer to a get-contextual-menu callback function.

```
void DisposeDataBrowserGetContextualMenuUPP (
   DataBrowserGetContextualMenuUPP userUPP
);
```

Parameters

userUPP

The universal procedure pointer to dispose of.

Discussion

See the `DataBrowserGetContextualMenuProcPtr` (page 2238) callback function.

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X version 10.0 and later.

Declared In

`HIDataBrowser.h`

DisposeDataBrowserHitTestUPP

Disposes of a universal procedure pointer to a hit-test callback function.

```
void DisposeDataBrowserHitTestUPP (
   DataBrowserHitTestUPP userUPP
);
```

Parameters

userUPP

The universal procedure pointer to dispose of.

Discussion
See the `DataBrowserHitTestProcPtr` (page 2240) callback function.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.1 and later.

Declared In
`HIDataBrowser.h`

DisposeDataBrowserItemAcceptDragUPP

Disposes of a universal procedure pointer to an item-accept-drag callback function.

```
void DisposeDataBrowserItemAcceptDragUPP (
   DataBrowserItemAcceptDragUPP userUPP
);
```

Parameters
userUPP
The universal procedure pointer to dispose of.

Discussion
See the `DataBrowserItemAcceptDragProcPtr` (page 2242) callback function.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.1 and later.

Declared In
`HIDataBrowser.h`

DisposeDataBrowserItemCompareUPP

Disposes of a universal procedure pointer to an item-comparison callback function.

```
void DisposeDataBrowserItemCompareUPP (
   DataBrowserItemCompareUPP userUPP
);
```

Parameters
userUPP
The universal procedure pointer to dispose of.

Discussion
See the `DataBrowserItemCompareProcPtr` (page 2243) callback function.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.

Declared In
`HIDataBrowser.h`

DisposeDataBrowserItemDataUPP

Disposes of a universal procedure pointer to an item-data callback function.

```
void DisposeDataBrowserItemDataUPP (
   DataBrowserItemDataUPP userUPP
);
```

Parameters

userUPP

The universal procedure pointer to dispose of.

Discussion

See the `DataBrowserItemDataProcPtr` (page 2245) callback function.

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X version 10.0 and later.

Declared In

`HIDataBrowser.h`

DisposeDataBrowserItemDragRgnUPP

Disposes of a universal procedure pointer to an item-drag-region callback function.

```
void DisposeDataBrowserItemDragRgnUPP (
   DataBrowserItemDragRgnUPP userUPP
);
```

Parameters

userUPP

The universal procedure pointer to dispose of.

Discussion

See the `DataBrowserItemDragRgnProcPtr` (page 2246) callback function.

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X version 10.1 and later.

Declared In

`HIDataBrowser.h`

DisposeDataBrowserItemHelpContentUPP

Disposes of a universal procedure pointer to an item-help-content callback function.

```
void DisposeDataBrowserItemHelpContentUPP (
   DataBrowserItemHelpContentUPP userUPP
);
```

Parameters

userUPP

The universal procedure pointer to dispose of.

Discussion
See the `DataBrowserItemHelpContentProcPtr` (page 2248) callback function.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.

Declared In
`HIDataBrowser.h`

DisposeDataBrowserItemNotificationUPP

Disposes of a universal procedure pointer to an item-notification callback function.

```
void DisposeDataBrowserItemNotificationUPP (
   DataBrowserItemNotificationUPP userUPP
);
```

Parameters
userUPP
The universal procedure pointer to dispose of.

Discussion
See the `DataBrowserItemNotificationProcPtr` (page 2250) callback function.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.

Declared In
`HIDataBrowser.h`

DisposeDataBrowserItemNotificationWithItemUPP

Disposes of a universal procedure pointer to an item-notification-with-data callback function.

```
void DisposeDataBrowserItemNotificationWithItemUPP (
   DataBrowserItemNotificationWithItemUPP userUPP
);
```

Parameters
userUPP
The universal procedure pointer to dispose of.

Discussion
See the `DataBrowserItemNotificationWithItemProcPtr` (page 2251) callback function.

Availability
Available in CarbonLib 1.5 and later.
Available in Mac OS X version 10.1 and later.

Declared In
`HIDataBrowser.h`

DisposeDataBrowserItemReceiveDragUPP

Disposes of a universal procedure pointer to an item-receive-drag callback function.

```
void DisposeDataBrowserItemReceiveDragUPP (
   DataBrowserItemReceiveDragUPP userUPP
);
```

Parameters

userUPP

The universal procedure pointer to dispose of.

Discussion

See the `DataBrowserItemReceiveDragProcPtr` (page 2254) callback function.

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X version 10.1 and later.

Declared In

`HIDataBrowser.h`

DisposeDataBrowserItemUPP

Disposes of a universal procedure pointer to an item-iterator callback function.

```
void DisposeDataBrowserItemUPP (
   DataBrowserItemUPP userUPP
);
```

Parameters

userUPP

The universal procedure pointer to dispose of.

Discussion

See the `DataBrowserItemProcPtr` (page 2253) callback function.

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X version 10.0 and later.

Declared In

`HIDataBrowser.h`

DisposeDataBrowserPostProcessDragUPP

Disposes of a universal procedure pointer to a postprocess-drag callback function.

```
void DisposeDataBrowserPostProcessDragUPP (
   DataBrowserPostProcessDragUPP userUPP
);
```

Parameters

userUPP

The universal procedure pointer to dispose of.

Discussion
See the `DataBrowserPostProcessDragProcPtr` (page 2255) callback function.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.

Declared In
`HIDataBrowser.h`

DisposeDataBrowserReceiveDragUPP

Disposes of a universal procedure pointer to a receive-drag callback function.

```
void DisposeDataBrowserReceiveDragUPP (
   DataBrowserReceiveDragUPP userUPP
);
```

Parameters
userUPP
The universal procedure pointer to dispose of.

Discussion
See the `DataBrowserReceiveDragProcPtr` (page 2256) callback function.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.

Declared In
`HIDataBrowser.h`

DisposeDataBrowserSelectContextualMenuUPP

Disposes of a universal procedure pointer to a select-contextual-menu callback function.

```
void DisposeDataBrowserSelectContextualMenuUPP (
   DataBrowserSelectContextualMenuUPP userUPP
);
```

Parameters
userUPP
The universal procedure pointer to dispose of.

Discussion
See the `DataBrowserSelectContextualMenuProcPtr` (page 2257) callback function.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.

Declared In
`HIDataBrowser.h`

DisposeDataBrowserTrackingUPP

Disposes of a universal procedure pointer to a tracking callback function.

```
void DisposeDataBrowserTrackingUPP (
   DataBrowserTrackingUPP userUPP
);
```

Parameters

userUPP

The universal procedure pointer to dispose of.

Discussion

See the `DataBrowserTrackingProcPtr` (page 2259) callback function.

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X version 10.1 and later.

Declared In

`HIDataBrowser.h`

EnableDataBrowserEditCommand

Determines whether the data browser is currently able to process a given editing command.

```
Boolean EnableDataBrowserEditCommand (
   ControlRef browser,
   DataBrowserEditCommand command
);
```

Parameters

browser

A data browser.

command

The editing command you want to enable. You can pass any of the constants described in "Editing Commands" (page 2277).

Return Value

A value of `true` if the requested editing command can be performed by the data browser at this time.

Discussion

Editing commands (Cut, Paste, Copy, and so on) can be enabled for an editable text field that is open and selected and for which the data browser is currently able to process the given command. For example, the data browser can process a Paste command only if there is text available on the Clipboard.

Editing commands are also available for a custom display type when the callbacks you install for the custom display indicate editing is available. Your application can call the function `EnableDataBrowserEditCommand` to discover if a specific editing command can be enabled. To execute an editing command, call the function `ExecuteDataBrowserEditCommand` (page 2138).

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X version 10.0 and later.

Not available to 64-bit applications.

Declared In
HIDataBrowser.h

ExecuteDataBrowserEditCommand

Executes an editing command.

```
OSStatus ExecuteDataBrowserEditCommand (
   ControlRef browser,
   DataBrowserEditCommand command
);
```

Parameters

browser
A data browser.

command
The editing command you want to execute. You can pass any of the constants described in "Editing Commands" (page 2277).

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
Editing commands can be executed for an editable text field that is open and selected. Your application can check to see if the editing command is enabled by first calling the function `EnableDataBrowserEditCommand` (page 2137).

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
HIDataBrowser.h

ForEachDataBrowserItem

Applies an item-iterator callback routine to each data item that meets the specified criteria.

```
OSStatus ForEachDataBrowserItem (
   ControlRef browser,
   DataBrowserItemID container,
   Boolean recurse,
   DataBrowserItemState state,
   DataBrowserItemUPP callback,
   void *clientData
);
```

Parameters

browser
A data browser.

container
An item ID or the constant `kDataBrowserNoItem`. To iterate through items that are organized as subitems of a container item, pass the item ID for the container. To iterate through all items displayed at the root of the data browser, pass the constant `kDataBrowserNoItem`.

recurse
A value that indicates whether or not to traverse the entire item hierarchy when applying the callback specified by the `callback` parameter. Pass `true` to apply the callback to all items in the hierarchy. Pass `false` if you want to apply the callback only to those items at the top level of the container or data browser.

state
A value that specifies the state of the items to which to apply the callback. Pass `0` if you want to apply the callback to all items, regardless of state. Otherwise, pass one of the constants described in "Item States" (page 2280).

callback
A universal procedure pointer to your item-iterator callback routine. This routine is called for every item ID that matches the specified criteria. See `DataBrowserItemProcPtr` (page 2253) for more information on the callback routine to supply.

clientData
A pointer to a buffer, local variable, or other storage location created and disposed of by your application and needed by your callback routine.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
The function `ForEachDataBrowserItem` is useful for enumerating and performing an operation on a set of item IDs.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

GetDataBrowserActiveItems

Obtains what determines the active state of the items in a data browser.

```
OSStatus GetDataBrowserActiveItems (
   ControlRef browser,
   Boolean *active
);
```

Parameters

browser
A data browser.

active

On input, a pointer to a Boolean variable. On return, the variable is set to `true` if the active state of each item in the list is determined by the item property `kDataBrowserItemIsActiveProperty`. Otherwise, the variable is set to `false` to indicate that all items are inactive.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

GetDataBrowserCallbacks

Obtains the callback routines installed for notifying your application of changes to a data browser and for providing the data to be displayed by the data browser.

```
OSStatus GetDataBrowserCallbacks (
   ControlRef browser,
   DataBrowserCallbacks *callbacks
);
```

Parameters

browser

The data browser whose callback routines you want to obtain.

callbacks

On input, a pointer to a `DataBrowserCallbacks` structure. On return, the structure contains universal procedure pointers to the callback routines installed for the data browser.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
When `GetDataBrowserCallacks` is used in conjunction with the function `SetDataBrowserCallbacks` (page 2200), your application can override or replace one or more callbacks used by a data browser to notify your application of changes.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

GetDataBrowserColumnViewDisplayType

Obtains the display type for a column view.

```
OSStatus GetDataBrowserColumnViewDisplayType (
   ControlRef browser,
   DataBrowserPropertyType *propertyType
);
```

Parameters

browser

A data browser.

propertyType

On input, a pointer to a display type variable. On return, the variable is set to the data type or control that is displayed in the data browser. No display types other than `kDataBrowserIconAndTextType` are currently supported in column view. See "Display Types" (page 2275) for more information.

Return Value

A result code. See "Data Browser Result Codes" (page 2298).

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X version 10.0 and later.

Not available to 64-bit applications.

Declared In

`HIDataBrowser.h`

GetDataBrowserColumnViewPath

Obtains the current path for a selection in column view.

```
OSStatus GetDataBrowserColumnViewPath (
   ControlRef browser,
   Handle path
);
```

Parameters

browser

A data browser.

path

On input, a handle. On return, the handle contains an array of item ID values that specify the current path. Array element `0` is the root; array element `N-1` is the target. You must allocate the handle before calling this function, and you are responsible for disposing of it.

Return Value

A result code. See "Data Browser Result Codes" (page 2298).

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X version 10.0 and later.

Not available to 64-bit applications.

Declared In

`HIDataBrowser.h`

GetDataBrowserColumnViewPathLength

Obtains the length of the current path for a column view.

```
OSStatus GetDataBrowserColumnViewPathLength (
   ControlRef browser,
   UInt32 *pathLength
);
```

Parameters

browser

A data browser.

pathLength

On input, a pointer to an unsigned 32-bit integer. On return, this value is set to the number of levels in the path for the currently selected item.

Return Value

A result code. See "Data Browser Result Codes" (page 2298).

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X version 10.0 and later.

Not available to 64-bit applications.

Declared In

`HIDataBrowser.h`

GetDataBrowserCustomCallbacks

Obtains the callbacks installed to implement custom drawing and behavior for the content in a data browser.

```
OSStatus GetDataBrowserCustomCallbacks (
   ControlRef browser,
   DataBrowserCustomCallbacks *callbacks
);
```

Parameters

browser

A data browser.

callbacks

On input, a pointer to a `DataBrowserCustomCallbacks` structure. On return, the structure contains universal procedure pointers to the custom callback routines installed for the data browser.

Return Value

A result code. See "Data Browser Result Codes" (page 2298).

Discussion

When `GetDataBrowserCustomCallbacks` is used in conjunction with the function `SetDataBrowserCustomCallbacks` (page 2203), your application can temporarily override or replace one or more callbacks used by a data browser to support custom drawing and custom behavior.

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X version 10.0 and later.

Not available to 64-bit applications.

Declared In
HIDataBrowser.h

GetDataBrowserEditItem

Obtains the item ID and property ID values of the current editing session.

```
OSStatus GetDataBrowserEditItem (
   ControlRef browser,
   DataBrowserItemID *item,
   DataBrowserPropertyID *property
);
```

Parameters

browser
A data browser.

item
On input, a pointer to an item ID variable. On return, the variable is set to the item ID of the item that is being edited. If there is no editing session in progress, this parameter is set to `kDataBrowserNoItem`.

property
On input, a pointer to a property ID variable. On return, the variable is set to the property ID of the item that is being edited. If there is no editing session in progress, this parameter is set to `0`.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
HIDataBrowser.h

GetDataBrowserEditText

Obtains the text being edited by the user.

Not Recommended

```
OSStatus GetDataBrowserEditText (
   ControlRef browser,
   CFMutableStringRef text
);
```

Parameters

browser
A data browser.

text

On return, the CFMutableString object is set to the text being edited by the user. Your application must allocate this object and pass it to the data browser. The data browser sets its contents to the current contents of the edit session text field. You must release this object when you no longer need it.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
This function does not work. Instead use `CopyDataBrowserEditText` (page 2125).

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

GetDataBrowserHasScrollBars

Obtains the display state of horizontal and vertical scroll bars for a list view data browser.

```
OSStatus GetDataBrowserHasScrollBars (
   ControlRef browser,
   Boolean *horiz,
   Boolean *vert
);
```

Parameters

browser

A list view data browser.

horiz

On input, a pointer to a Boolean variable. On return, the variable is set to `true` if the browser control has a horizontal scroll bar.

vert

On input, a pointer to a Boolean variable. On return, the variable is set to `true` if the browser control has a vertical scroll bar.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
The function `GetDataBrowserHasScrollBars` is useful for determining if the browser control currently has scroll bars. For example, you would call the function `AutoSizeDataBrowserListViewColumns` (page 2122) only after you have determined the data browser does not have a horizontal scroll bar.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
HIDataBrowser.h

GetDataBrowserItemCount

Obtains the number of items whose state matches the specified state.

```
OSStatus GetDataBrowserItemCount (
   ControlRef browser,
   DataBrowserItemID container,
   Boolean recurse,
   DataBrowserItemState state,
   ItemCount *numItems
);
```

Parameters

browser
A data browser.

container
An item ID or the constant `kDataBrowserNoItem`. To obtain the number of items that are organized as subitems of a container item, pass the item ID for the container. To obtain the number of items displayed at the root of the data browser, provide the constant `kDataBrowserNoItem`.

recurse
A value that indicates whether or not to traverse the entire item hierarchy when counting. Pass `true` to obtain a count for all items in the hierarchy. Pass `false` if you want to count only those items at the top level of the container or data browser.

state
A value that specifies the state of the items to obtain. Only items that have this state are counted. Pass `kDataBrowserItemAnyState` if you want to count all items regardless of state. Otherwise, pass one of the constants described in "Item States" (page 2280).

numItems
On input, a pointer to an unsigned 32-bit integer. On return, this value is set to the number of items in the container that have the specified state.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
HIDataBrowser.h

GetDataBrowserItemDataBooleanValue

Obtains the Boolean value for an item.

```
OSStatus GetDataBrowserItemDataBooleanValue (
   DataBrowserItemDataRef itemData,
   Boolean *theData
);
```

Parameters

itemData

The item data reference for the item whose Boolean value you want to obtain. The item data reference is passed to the callback routine from which you are calling the function `GetDataBrowserItemDataBooleanValue`.

theData

On input, a pointer to a Boolean variable. On return, the variable is set to the Boolean value.

Return Value

A result code. See "Data Browser Result Codes" (page 2298).

Discussion

You can obtain Boolean values for the following properties:

- `kDataBrowserItemIsActiveProperty`
- `kDataBrowserItemIsSelectableProperty`
- `kDataBrowserItemIsEditableProperty`
- `kDataBrowserItemIsContainerProperty`
- `kDataBrowserItemIsOpenableProperty`
- `kDataBrowserItemIsClosableProperty`
- `kDataBrowserItemIsSortableProperty`

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X version 10.0 and later.

Not available to 64-bit applications.

Declared In

`HIDataBrowser.h`

GetDataBrowserItemDataButtonValue

Obtains the value for a checkbox.

```
OSStatus GetDataBrowserItemDataButtonValue (
   DataBrowserItemDataRef itemData,
   ThemeButtonValue *theData
);
```

Parameters

itemData

The item data reference for the item whose checkbox setting you want to obtain. The item data reference is passed to the callback routine from which you are calling the function `GetDataBrowserItemDataButtonValue`.

theData

On input, a pointer to a theme button value variable. On return, the variable is set to the checkbox setting. The value can be one of the following theme button value constants defined by the Appearance Manager,:

- `kThemeButtonOff` indicates a checkbox that is not selected.
- `kThemeButtonOn` indicates a checkbox that is selected.
- `kThemeButtonMixed` draws a checkbox that in a mixed state, indicating that a setting is on for some items in a selection and off for others.

See *Appearance Manager Reference* for more information.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
Your item-data callback calls this function in response to a set-data request for items that have the display type `kDataBrowserCheckboxType`.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

GetDataBrowserItemDataDateTime

Obtains, as a 32-bit value, the date and time value displayed.

```
OSStatus GetDataBrowserItemDataDateTime (
   DataBrowserItemDataRef itemData,
   SInt32 *theData
);
```

Parameters

itemData

The item data reference for the item whose date and time value you want to obtain. The item data reference is passed to the callback routine from which you are calling the function `GetDataBrowserItemDataDateTime`.

theData

On input, a 32-bit value. On return, the value is set to the number of elapsed seconds since midnight, January 1, 1904. For more information about date and time encodings used in the Mac OS, see *Date, Time, and Measurement Utilities Reference* in Carbon Text & International Documentation.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
This function works only with items that have the property `kDataBrowserDateTimeType`. If the column has the property `kDataBrowserRelativeDateTime`, the date is displayed relative to the current time for the computer. For example, a time 24 hours prior to the current time is displayed as "Yesterday." Other examples of relative date and time values are "Today, 1:45 PM" and "Yesterday, 7:30 AM."

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

GetDataBrowserItemDataDrawState

Determines whether a checkbox is in the active or inactive state.

```
OSStatus GetDataBrowserItemDataDrawState (
   DataBrowserItemDataRef itemData,
   ThemeDrawState *theData
);
```

Parameters

itemData

The item data reference for the checkbox whose drawing state you want to obtain. This value is passed to the callback routine from which you are calling the function `GetDataBrowserItemDataDrawState`.

theData

On input, a pointer to a theme draw state variable. On return, the variable is set to the drawing state for the item, either `kThemeStateInactive` or `kThemeStateActive`. See *Appearance Manager Reference* for more information on these constants.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
Your item-data callback calls this function in response to a get-data request for items that have display type `kDataBrowserCheckboxType`.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

GetDataBrowserItemDataIcon

Obtains the icon drawn for an item.

```
OSStatus GetDataBrowserItemDataIcon (
   DataBrowserItemDataRef itemData,
   IconRef *theData
);
```

Parameters

itemData

The item data reference for the item whose icon you want to obtain. This value is passed to the callback routine from which you are calling the function `GetDataBrowserItemDataIcon`.

theData

On input, a pointer to an `IconRef` variable. On return, the variable is set to the icon that is displayed. You are responsible for disposing of the `IconRef`.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
You call the function `GetDataBrowserItemDataIcon` from within a `DataBrowserItemDataProcPtr` (page 2245) callback routine to obtain the icon drawn in a column that has the `kDataBrowserIconType` display type or the `kDataBrowserIconAndTextType` display type.

Availability
Available in CarbonLib 1.5 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

GetDataBrowserItemDataIconTransform

Obtains the transformation currently used to display an icon.

```
OSStatus GetDataBrowserItemDataIconTransform (
   DataBrowserItemDataRef itemData,
   IconTransformType *theData
);
```

Parameters

itemData

The item data reference for the item whose icon transformation you want to obtain. This value is passed to the callback routine from which you are calling the function `GetDataBrowserItemDataIconTransform`.

theData

On input, an icon transformation type variable. On return, the variable is set to an icon transformation type. This value can be any of the icon transformation constants defined by Icon Services and Utilities. See *Icon Services and Utilities Reference* for more information.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
This function works only with items that have either the property `kDataBrowserIconAndTextType` or `kDataBrowserIconType`.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

GetDataBrowserItemDataItemID

Obtains the item ID for an item whose property is another item's ID.

```
OSStatus GetDataBrowserItemDataItemID (
   DataBrowserItemDataRef itemData,
   DataBrowserItemID *theData
);
```

Parameters

itemData

The item data reference passed to your item-data callback.

theData

On input, a pointer to an item ID variable. On return, the variable is set to the item ID associated with the `itemData` parameter.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
Typically you do not need to call this function. This function is used for item properties `kDataBrowserParentContainerProperty` or `kDataBrowserContainerAliasIDProperty`.

Availability
Available in CarbonLib 1.5 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

GetDataBrowserItemDataLongDateTime

Obtains, as a 64-bit value, the date and time value displayed.

```
OSStatus GetDataBrowserItemDataLongDateTime (
   DataBrowserItemDataRef itemData,
   LongDateTime *theData
);
```

Parameters

itemData

The item data reference for the item whose long date and time value you want to obtain. The item data reference is passed to the callback routine from which you are calling the function `GetDataBrowserItemDataLongDateTime`.

theData

On input, a 64-bit value. On return, the value is set to the number of seconds elapsed since midnight, January 1, 1904. For more information about date and time encodings used in the Mac OS, see *Date, Time, and Measurement Utilities Reference* in Carbon Text & International Documentation.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
This function works only with items that have the property `kDataBrowserDateTimeType`. If the column has the property `kDataBrowserRelativeDateTime`, the date is displayed relative to the current time for the computer. For example, a time 24 hours prior to the current time is displayed as "Yesterday." Other examples of relative date and time values are "Today, 1:45 PM" and "Yesterday, 7:30 AM."

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

GetDataBrowserItemDataMaximum

Obtains the maximum integer value that can be displayed; useful for such display types as sliders, progress bars, relevance indicators, and pop-up menus.

```
OSStatus GetDataBrowserItemDataMaximum (
   DataBrowserItemDataRef itemData,
   SInt32 *theData
);
```

Parameters

itemData

The item data reference for the item whose maximum value you want to obtain. The item data reference is passed to the callback routine from which you are calling the function `GetDataBrowserItemDataMaximum`.

theData

On input, a pointer to a signed 32-bit integer. On return, this value is set to the maximum value that can be displayed for the item.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

GetDataBrowserItemDataMenuRef

Obtains the pop-up menu displayed.

```
OSStatus GetDataBrowserItemDataMenuRef (
   DataBrowserItemDataRef itemData,
   MenuRef *theData
);
```

Parameters

itemData

The item data reference for the item whose pop-up menu you want to obtain. The item data reference is passed to the callback routine from which you are calling the function `GetDataBrowserItemDataMenuRef`.

theData

On input, a pointer to a menu reference. On return, this is set to the currently displayed pop-up menu. The system retains the menu reference that you pass; you must release it when you no longer need it.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
Your item-data callback calls this function in response to a get-data request for items that have the display type `kDataBrowserPopupMenuType`.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

GetDataBrowserItemDataMinimum

Obtains the minimum integer value that can be displayed for an item; useful for such display types as sliders, progress bars, relevance indicators, and pop-up menus.

```
OSStatus GetDataBrowserItemDataMinimum (
   DataBrowserItemDataRef itemData,
   SInt32 *theData
);
```

Parameters

itemData

The item data reference for the item whose minimum value you want to obtain. The item data reference is passed to the callback routine from which you are calling the function `GetDataBrowserItemDataMinimum`.

theData

On input, a pointer to a signed 32-bit integer. On return, this value is set to the minimum value that can be displayed for the item.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

GetDataBrowserItemDataProperty

Obtains the column property ID for the column in which an item resides.

```
OSStatus GetDataBrowserItemDataProperty (
   DataBrowserItemDataRef itemData,
   DataBrowserPropertyID *theData
);
```

Parameters

itemData

The item data reference for the item whose property ID you want to obtain. This value is passed to the callback routine from which you are calling the function `GetDataBrowserItemDataProperty`.

theData

On input, a pointer to a property ID variable. On return, the variable is set to the property ID for the item.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
When your item-data callback is invoked for an item that has the property `kDataBrowserItemIsEditableProperty`, you call the function `GetDataBrowserItemDataProperty` to obtain the column property ID. Then, your callback can use the column property ID to determine whether the item is in a column whose data can be edited.

For example, consider a list view data browser whose columns are titled "Name" and "Date Modified." Let's say Name can be modified by the user, but the Date Modified column cannot. If the user clicks an item in one of the columns, your item-data callback is called to find out whether the clicked column is editable. Your

callback needs to find out which column the "is editable" request is being made for by calling the function `GetDataBrowserItemDataProperty`. In this example, after you obtain the property ID, you would check whether the column is the Date Modified column or the Name column. You'd allow editing only if the item is in the Name column.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

GetDataBrowserItemDataRGBColor

Obtains the color used to draw an item.

```
OSStatus GetDataBrowserItemDataRGBColor (
   DataBrowserItemDataRef itemData,
   RGBColor *theData
);
```

Parameters

itemData
The item data reference for the item whose color you want to obtain. This value is passed to the callback routine from which you are calling the function `GetDataBrowserItemDataRGBColor`.

theData
On input, an RGB color variable. On return, the variable is set to the RGB values that specify the color of the item.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
Your item-data callback calls this function in response to a get-data request for items that have the display type `kDataBrowserIconType` or `kDataBrowserIconAndTextType`.

As of Mac OS X 10.3, this function does nothing.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

GetDataBrowserItemDataText

Obtains the text entered by the user.

```
OSStatus GetDataBrowserItemDataText (
   DataBrowserItemDataRef itemData,
   CFStringRef *theData
);
```

Parameters

itemData

The item data reference for the item whose text you want to obtain. This value is passed to the callback routine from which you are calling the function `GetDataBrowserItemDataText`.

theData

On input, a `CFStringRef` variable. On return, a CFString object that contains the text. Your application must release the CFString object when it is no longer needed.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
You can call the function `GetDataBrowserItemDataText` from inside an item-data callback routine when the callback's `setValue` parameter is `true`. A value of `true` indicates that the displayed text has been modified by the user. In that case, your application calls `GetDataBrowserItemDataText` to retrieve the modified text.

Note that a column is editable only if the `kDataBrowserPropertyIsEditable` flag is set for the column.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

GetDataBrowserItemDataValue

Obtains the value of an item; useful for such display types as sliders, progress bars, relevance indicators, and pop-up menus.

```
OSStatus GetDataBrowserItemDataValue (
   DataBrowserItemDataRef itemData,
   SInt32 *theData
);
```

Parameters

itemData

The item data reference for the item whose integer value you want to obtain. The item data reference is passed to the callback routine from which you are calling the function `GetDataBrowserItemDataValue`.

theData

On input, a pointer to a signed 32-bit integer. On return, it is set to the displayed value.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
Your application calls the function `GetDataBrowserItemDataValue` to obtain a new value for a display type when your item-data callback routine is called with the `setValue` parameter set to `true`.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

GetDataBrowserItemPartBounds

Obtains the bounds of a visual part of an item.

```
OSStatus GetDataBrowserItemPartBounds (
   ControlRef browser,
   DataBrowserItemID item,
   DataBrowserPropertyID property,
   DataBrowserPropertyPart part,
   Rect *bounds
);
```

Parameters

browser
A data browser.

item
The item ID that identifies the row.

property
The property ID that identifies the column.

part
The part for which you want to obtain information. The information requested depends on the type of information displayed in the column. It is up to your application to ensure it requests the appropriate information. See "Property Parts" (page 2291) for a list of the constants you can provide in this parameter.

bounds
On input, a pointer to a rectangle. On return, the rectangle contains the bounds for the specified part.

Return Value
A result code. If the item is not visible (scrolled off the screen), returns the result `ItemNotFound`. See "Data Browser Result Codes" (page 2298).

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

GetDataBrowserItems

Obtains a list of the items that match a specified state; operates on items in the root container or traverses items in the data hierarchy.

```
OSStatus GetDataBrowserItems (
   ControlRef browser,
   DataBrowserItemID container,
   Boolean recurse,
   DataBrowserItemState state,
   Handle items
);
```

Parameters

browser

A data browser.

container

An item ID or the constant `kDataBrowserNoItem`. To obtain a list of items that are organized as subitems of a container, pass the item ID of the container item. To obtain a list of items displayed in the root container, pass the constant `kDataBrowserNoItem`.

recurse

A value that indicates whether or not to traverse the entire item hierarchy when obtaining item IDs. Pass `true` to obtain item IDs for all items in the hierarchy. Pass `false` if you want to count only those item IDs at the top level of the container. If you pass `true`, you obtain a flattened list of item IDs. The list reflects the hierarchy maintained internally by the data browser and might not reflect the order of the items as they appear onscreen to the user.

state

The state of the items to obtain. Only items that have this state are returned in the `items` parameter. Pass `0` if you want to obtain all items regardless of state. Otherwise, pass one of the constants described in "Item States" (page 2280).

items

On return, the contents of the handle contain an array of item ID values for the matching items. You must allocate and dispose of the handle. To determine the number of items in the array, call the function `GetHandleSize` and divide by the size of `DataBrowserItemID`. Note that the handle contents are completely replaced by the returned array.

Return Value

A result code. See "Data Browser Result Codes" (page 2298).

Discussion

The function `GetDataBrowserItems` is a powerful routine for gathering information about the items displayed in a data browser. For example, to obtain a list of all the items the user has selected in a list, call the function with the `state` parameter set to `kDataBrowserItemIsSelected`. If your application is interested only in determining the number of items in a selection (and not the item IDs of those items), call the function `GetDataBrowserItemCount` (page 2145).

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X version 10.0 and later.

Not available to 64-bit applications.

Declared In

`HIDataBrowser.h`

GetDataBrowserItemState

Obtains the state of an item.

```
OSStatus GetDataBrowserItemState (
   ControlRef browser,
   DataBrowserItemID item,
   DataBrowserItemState *state
);
```

Parameters

browser

A data browser.

item

The item ID of the item whose state you want to check.

state

On input, a pointer to an item state variable. On return, the variable is set to a value that specifies the state of the item. See "Item States" (page 2280) for a description of the values that can be returned.

Return Value

A result code. See "Data Browser Result Codes" (page 2298).

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X version 10.0 and later.

Not available to 64-bit applications.

Declared In

`HIDataBrowser.h`

GetDataBrowserListViewDisclosureColumn

Obtains the property ID of the column whose items can display a disclosure triangle, and tells whether a disclosed item expands the row or adds rows.

```
OSStatus GetDataBrowserListViewDisclosureColumn (
   ControlRef browser,
   DataBrowserTableViewColumnID *column,
   Boolean *expandableRows
);
```

Parameters

browser

A data browser.

column

On input, a pointer to a column ID variable. On return, the variable is set to the property ID of the currently selected column. If there is no disclosure column, the variable is set to `kDataBrowserItemNoProperty`. The `DataBrowserTableViewColumnID` data type is the same as the `DataBrowserPropertyID` data type.

expandableRows

On input, a pointer to a Boolean variable. On return, the variable specifies how a disclosed row behaves. The value `true` means that a container opens as a single row with an expanded height. The value `false` means a container opens to expose individual rows. See the Discussion for more details on expandable rows.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
When the `expandableRows` variable is set to `true`:

- Disclosure triangles are drawn top-justified in the row.
- Custom row height, if any, for that row is respected only while the row is disclosed. At other times, the default row height is used.

When the `expandableRows` variable is set to `false`:

- Disclosure triangles are centered vertically in the row.
- Custom row height, if any, for that row is always respected.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

GetDataBrowserListViewHeaderBtnHeight

Obtains the height of the rectangular area where the column title appears.

```
OSStatus GetDataBrowserListViewHeaderBtnHeight (
   ControlRef browser,
   UInt16 *height
);
```

Parameters

browser

A data browser.

height

On input, a pointer to an unsigned 16-bit integer. On return, this value is set to the height of the rectangular area where the column title appears. You can save this value if you plan to call the function `SetDataBrowserListViewHeaderBtnHeight` (page 2217) to turn off header button display. You can then use the value later to turn on header button display.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Availability
Available in CarbonLib 1.1 and later.

Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
HIDataBrowser.h

GetDataBrowserListViewHeaderDesc

Obtains a header description for a column in list view.

```
OSStatus GetDataBrowserListViewHeaderDesc (
   ControlRef browser,
   DataBrowserTableViewColumnID column,
   DataBrowserListViewHeaderDesc *desc
);
```

Parameters

browser
A data browser.

column
The property ID for the column whose list view header description you want to obtain. The `DataBrowserTableViewColumnID` data type is the same as the `DataBrowserPropertyID` data type.

desc
On input, a pointer to a list view header description data structure. On return, the structure contains the header description for the specified column in list view.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
If your application allows the user to switch between column and list views, you can call this function to obtain the current header description and then save the description before you switch from list to column view. When you switch from column to list view, you can restore the list view header information by calling the function `SetDataBrowserListViewHeaderDesc` (page 2218) passing the header information you saved.

Availability
Available in CarbonLib 1.5 and later.
Available in Mac OS X version 10.2 and later.
Not available to 64-bit applications.

Declared In
HIDataBrowser.h

GetDataBrowserListViewUsePlainBackground

Determines whether list view is set to use a plain white background.

```
OSStatus GetDataBrowserListViewUsePlainBackground (
   ControlRef browser,
   Boolean *usePlainBackground
);
```

Parameters

browser

A data browser.

usePlainBackground

On input, a pointer to a Boolean variable. On return, the variable is `true` if list view is set to use a plain white background. Regardless of the value that is returned, Mac OS X supports only a plain white background. Mac OS 9 supports a plain white background as well as a shaded background.

Return Value

A result code. See "Data Browser Result Codes" (page 2298).

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X version 10.0 and later.

Not available to 64-bit applications.

Declared In

`HIDataBrowser.h`

GetDataBrowserPropertyFlags

Obtains the appearance and behavior attributes for a column.

```
OSStatus GetDataBrowserPropertyFlags (
   ControlRef browser,
   DataBrowserPropertyID property,
   DataBrowserPropertyFlags *flags
);
```

Parameters

browser

A data browser.

property

The property ID of the column whose properties you want to obtain.

flags

On input, a data browser property flags variable. On return, the variable is set to the property flags that specify the appearance and behavior attributes for a column. A `DataBrowserPropertyFlags` value is a 32-bit value that is divided into four parts as follows:

- Bits 0–7 specify properties applied to the data browser as a whole—see "Property Flags: Universal" (page 2284)
- Bits 8–15 modify display behavior—see "Property Flags: Modifiers" (page 2285)
- Bits 16–23 are properties specific to list view—see "Property Flags: Offset and Mask for List View Properties" (page 2288) and "Property Flags: List View Column Behavior" (page 2289)
- Bits 24–31 can be defined by your application—see "Property Flags: Offset and Mask for Client-Defined Properties" (page 2290)

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

GetDataBrowserScrollBarInset

Obtains the inset rectangle used by a data browser to position the scroll bar.

```
OSStatus GetDataBrowserScrollBarInset (
   ControlRef browser,
   Rect *insetRect
);
```

Parameters

browser
A data browser.

insetRect
On input, a pointer to a rectangle structure. On return, the rectangle contains the current inset settings for the data browser scroll bars. The `left` and `right` fields contain the horizontal inset values for the horizontal scroll bar, and the `top` and `bottom` fields contain the vertical inset values for the vertical scroll bar.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
Your application can call the functions `GetDataBrowserScrollBarInset` and `SetDataBrowserScrollBarInset` (page 2220) if you want to place placards or controls beside the horizontal scroll bars or above the vertical ones. To do so, first call `GetDataBrowserScrollBarInset` to obtain the current settings. After modifying the current inset settings to provide space for the placard or control, call `SetDataBrowserScrollBarInset` with the new values.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

GetDataBrowserScrollPosition

Obtains the scrolling position of a list.

```
OSStatus GetDataBrowserScrollPosition (
   ControlRef browser,
   UInt32 *top,
   UInt32 *left
);
```

Parameters

browser

A data browser.

top

On input, a pointer to an unsigned 32-bit integer. On return, this value is set to the current vertical scrolling position.

left

On input, a pointer to an unsigned 32-bit integer. On return, this value is set to the current horizontal scrolling position.

Return Value

A result code. See "Data Browser Result Codes" (page 2298).

Discussion

Normally, you use the function `GetDataBrowserScrollPosition` in conjunction with `SetDataBrowserScrollPosition` (page 2221) to save and restore the scrolling position of a list to the user's last scrolling position. These functions should not be used to scroll particular cells into the view. For that, call the function `RevealDataBrowserItem`.

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X version 10.0 and later.

Not available to 64-bit applications.

Declared In

`HIDataBrowser.h`

GetDataBrowserSelectionAnchor

Obtains the first and last items in a selection.

```
OSStatus GetDataBrowserSelectionAnchor (
   ControlRef browser,
   DataBrowserItemID *first,
   DataBrowserItemID *last
);
```

Parameters

browser

A data browser.

first

On input, a pointer to an item ID variable. On return, the variable is set to the item ID of the first item in the selection.

last

On input, a pointer to an item ID variable. On return, the variable is set to the item ID of the last item in the selection.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

GetDataBrowserSelectionFlags

Obtains the current selection behavior for a data browser.

```
OSStatus GetDataBrowserSelectionFlags (
   ControlRef browser,
   DataBrowserSelectionFlags *selectionFlags
);
```

Parameters

browser
A data browser.

selectionFlags
On input, a data browser selection flags variable. On return, the variable is set to the current selection flags. See "User Selection Flags" (page 2296) for a list of the flags that can be returned.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
Selection flags specify the selection behavior available to the user, such as whether the user can select discontinuous items.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

GetDataBrowserSortOrder

Gets the sorting order of the list view column that's currently set for sorting.

```
OSStatus GetDataBrowserSortOrder (
   ControlRef browser,
   DataBrowserSortOrder *order
);
```

Parameters

browser

A data browser.

order

On input, a pointer to a sorting order variable. On return, the variable is set to the sorting order of the current sort column in list view. See "Sorting Orders" (page 2294) for a list of the values that can be returned.

Return Value

A result code. See "Data Browser Result Codes" (page 2298).

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X version 10.0 and later.

Not available to 64-bit applications.

Declared In

`HIDataBrowser.h`

GetDataBrowserSortProperty

Obtains the property ID of the column currently used for sorting in list view.

```
OSStatus GetDataBrowserSortProperty (
   ControlRef browser,
   DataBrowserPropertyID *property
);
```

Parameters

browser

A data browser.

property

On input, a pointer to a property ID variable. On return, the variable is set to the property ID of the column used for sorting.

Return Value

A result code. See "Data Browser Result Codes" (page 2298).

Discussion

You can call the function `GetDataBrowserSortProperty` to discover the property ID of the column currently used for sorting. To designate another column for the sorting operation, call the function `SetDataBrowserSortProperty` (page 2223).

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X version 10.0 and later.

Not available to 64-bit applications.

Declared In
HIDataBrowser.h

GetDataBrowserTableViewColumnCount

Obtains the number of columns in a data browser.

```
OSStatus GetDataBrowserTableViewColumnCount (
   ControlRef browser,
   UInt32 *numColumns
);
```

Parameters

browser
A data browser.

numColumns
On input, a pointer to an unsigned 32-bit integer. On return, this value is set to the number of columns in the data browser.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
HIDataBrowser.h

GetDataBrowserTableViewColumnPosition

Obtains the column position for an item in a data browser.

```
OSStatus GetDataBrowserTableViewColumnPosition (
   ControlRef browser,
   DataBrowserTableViewColumnID column,
   DataBrowserTableViewColumnIndex *position
);
```

Parameters

browser
A data browser.

column
The property ID for the list view column for which you want to obtain the position. The `DataBrowserTableViewColumnID` data type is the same as the `DataBrowserPropertyID` data type.

position
On input, a pointer to a column index variable. On return, the variable is set to the column index for the item; `0` is the leftmost column.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

GetDataBrowserTableViewColumnProperty

Obtains the property ID for a column in a data browser.

```
OSStatus GetDataBrowserTableViewColumnProperty (
   ControlRef browser,
   DataBrowserTableViewColumnIndex column,
   DataBrowserTableViewColumnID *property
);
```

Parameters

browser
A data browser.

column
The column index of the column whose property ID you want to obtain.

property
On input, a pointer to a column ID variable. On return, the variable is set to the property ID for the column. The `DataBrowserTableViewColumnID` data type is the same as the `DataBrowserPropertyID` data type.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

GetDataBrowserTableViewColumnWidth

Obtains the default column width used for all columns in a data browser.

```
OSStatus GetDataBrowserTableViewColumnWidth (
   ControlRef browser,
   UInt16 *width
);
```

Parameters

browser

A data browser.

width

On input, a pointer to an unsigned 16-bit integer. On return, this value is set to the column width, in pixels.

Return Value

A result code. See "Data Browser Result Codes" (page 2298).

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X version 10.0 and later.

Not available to 64-bit applications.

Declared In

`HIDataBrowser.h`

GetDataBrowserTableViewGeometry

Determines whether columns and rows are set to have variable widths.

```
OSStatus GetDataBrowserTableViewGeometry (
   ControlRef browser,
   Boolean *variableWidthColumns,
   Boolean *variableHeightRows
);
```

Parameters

browser

A data browser.

variableWidthColumns

On input, a pointer to a Boolean variable. On return, the variable is set to `true` if column widths can be changed or `false` if they cannot be changed.

variableHeightRows

On input, a pointer to a Boolean variable. On return, the variable is set to `true` if row heights can be changed or `false` if they cannot be changed.

Return Value

A result code. See "Data Browser Result Codes" (page 2298).

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X version 10.0 and later.

Not available to 64-bit applications.

Declared In

`HIDataBrowser.h`

GetDataBrowserTableViewHiliteStyle

Obtains the highlighting style used for a list view data browser.

```
OSStatus GetDataBrowserTableViewHiliteStyle (
   ControlRef browser,
   DataBrowserTableViewHiliteStyle *hiliteStyle
);
```

Parameters

browser

A list view data browser.

hiliteStyle

On input, a pointer to a highlighting style variable. On return, the variable is set to the highlighting style in use. See "Table View Highlighting Styles" (page 2294) for a description of the values that can be returned.

Return Value

A result code. See "Data Browser Result Codes" (page 2298).

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X version 10.0 and later.

Not available to 64-bit applications.

Declared In

`HIDataBrowser.h`

GetDataBrowserTableViewItemID

Obtains the item ID for the item displayed in the specified row.

```
OSStatus GetDataBrowserTableViewItemID (
   ControlRef browser,
   DataBrowserTableViewRowIndex row,
   DataBrowserItemID *item
);
```

Parameters

browser

A data browser.

row

The row index for the item. The row index is the visual order of rows in the table onscreen. Rows are numbered starting at the top of the table, with the value `0`, and proceeding sequentially to the bottom of the table.

item

On input, a pointer to an item ID variable. On return, the variable is set to the item ID of the data displayed in the row.

Return Value

A result code. See "Data Browser Result Codes" (page 2298).

Discussion
This function is useful only in edge case situations for which you need to know what item ID is displayed in a particular row. For example, if you are performing some fairly involved and complex custom hit-testing, you might need to call this function.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

GetDataBrowserTableViewItemRow

Obtains the visual position for the specified item in list view.

```
OSStatus GetDataBrowserTableViewItemRow (
   ControlRef browser,
   DataBrowserItemID item,
   DataBrowserTableViewRowIndex *row
);
```

Parameters

browser
A data browser.

item
The item ID for the item whose row index you want to obtain.

row
On input, a pointer to a row index variable. On return, the variable is set to the row index for the item.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
You can use this function for items in list view. It is the inverse of the function `GetDataBrowserTableViewItemID` (page 2169).

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

GetDataBrowserTableViewItemRowHeight

Obtains the row height for a single row in a list view data browser.

```
OSStatus GetDataBrowserTableViewItemRowHeight (
   ControlRef browser,
   DataBrowserItemID item,
   UInt16 *height
);
```

Parameters

browser

A data browser.

item

The item ID of the item whose row height you want to obtain.

height

On input, a pointer to an unsigned 16-bit integer. On return, this value is set to the row height, in pixels.

Return Value

A result code. See "Data Browser Result Codes" (page 2298).

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X version 10.0 and later.

Not available to 64-bit applications.

Declared In

`HIDataBrowser.h`

GetDataBrowserTableViewNamedColumnWidth

Obtains the column width for a single column in a data browser.

```
OSStatus GetDataBrowserTableViewNamedColumnWidth (
   ControlRef browser,
   DataBrowserTableViewColumnID column,
   UInt16 *width
);
```

Parameters

browser

A data browser.

column

The property ID for the list view column whose width you want to obtain. The `DataBrowserTableViewColumnID` data type is the same as the `DataBrowserPropertyID` data type.

width

On input, a pointer to an unsigned 16-bit integer. On return, this value is set to the width of the column, in pixels.

Return Value

A result code. See "Data Browser Result Codes" (page 2298).

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X version 10.0 and later.

Not available to 64-bit applications.

Declared In
HIDataBrowser.h

GetDataBrowserTableViewRowHeight

Obtains the default row height used for all rows in a data browser.

```
OSStatus GetDataBrowserTableViewRowHeight (
   ControlRef browser,
   UInt16 *height
);
```

Parameters

browser

A data browser.

height

On input, a pointer to an unsigned 16-bit integer. On return, this value is set to the row height, in pixels.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
HIDataBrowser.h

GetDataBrowserTarget

Obtains the target for the data browser

```
OSStatus GetDataBrowserTarget (
   ControlRef browser,
   DataBrowserItemID *target
);
```

Parameters

browser

A data browser.

target

On input, a pointer to an item ID variable. On return, the variable is set to the item ID for the currently assigned target.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
In column view, the target is the rightmost column. In list view, the target can be thought of as the root container.

Your application can call the function `SetDataBrowserTarget` (page 2229) to set an item ID to use as a target if you do not want to use the default target set by the data browser.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

GetDataBrowserUserState

Obtains the current view style settings for a list view.

```
OSStatus GetDataBrowserUserState (
   ControlRef browser,
   CFDictionaryRef *stateInfo
);
```

Parameters

browser
A data browser.

stateInfo
On input, a pointer to a `CFDictionaryRef`. On return, a CFDictionary object that contains the current view style settings. You must release the object when you no longer need it by calling the function `CFRelease`. Note that although this parameter is typed as a CFData object, you must treat the result as a CFDictionary object because that is what the system fills out and returns to you.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
You typically use this function to obtain the current user settings for the data browser so that you can save the settings to a preferences file. User settings include data such as sorting order, sorting column, and column widths. Later, you can restore the settings by calling the function `SetDataBrowserUserState` (page 2230).

If you want to save the user settings to disk, you need to determine the length of the user-settings data in bytes. The following code shows how to calculate this length. First you need to convert the CFDictionary object you obtain from the function `GetDataBrowserUserState` to a property list. Then you can call the function `CFDataGetLength` to obtain the length, in bytes, of the property list.

```
CFDataRef myUserState = NULL;
OSStatus status;

ControlRef browser = GetDataBrowserFromWindow (window);
status = GetDataBrowserUserState (browser, &myUserState);

if (noErr == status)
{
```

```
  CFDataRef myTempDataRef = NULL;
  CFIndex index;

  if (myUserState != NULL)
  {
    // Convert the user state dictionary to a property list.
    myTempDataRef = CFPropertyListCreateXMLData (NULL,
                        (CFPropertyListRef) myUserState);
    if (myTempDataRef != NULL)
    {
        // Get the length, in bytes
      index = CFDataGetLength (myTempDataRef);
        // Call your function to save the data
        // You need to release the CFDataRef you created
      CFRelease (myTempDataRef);
    }
  }
 CFRelease (myUserState);
}
```

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

GetDataBrowserViewStyle

Obtains the current view style for the specified data browser.

```
OSStatus GetDataBrowserViewStyle (
   ControlRef browser,
   DataBrowserViewStyle *style
);
```

Parameters

browser
A data browser.

style
On input, a pointer to a view style variable. On return, the variable is set to the current view style for the specified data browser; can be either list view (`kDataBrowserListView`) or column view (`kDataBrowserColumnView`). See "View Styles" (page 2298) for more information on these constants.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

InitDataBrowserCallbacks

Initializes a data browser callback structure in preparation for adding your own callbacks to the structure.

```
OSStatus InitDataBrowserCallbacks (
   DataBrowserCallbacks *callbacks
);
```

Parameters

callbacks

A pointer to a `DataBrowserCallbacks` structure. Before calling the function `InitDataBrowserCallbacks`, set the `version` field of this structure to `kDataBrowserLatestCallbacks`. On return, the fields in this structure are set to `NULL`.

Return Value

A result code. See "Data Browser Result Codes" (page 2298).

Discussion

After you call this function, set the appropriate fields in the `DataBrowserCallbacks` structure to your callbacks. The `DataBrowserCallbacks` structure contains fields for the following:

- `DataBrowserItemDataProcPtr` (page 2245)
- `DataBrowserItemCompareProcPtr` (page 2243)
- `DataBrowserItemNotificationProcPtr` (page 2250) or `DataBrowserItemNotificationWithItemProcPtr` (page 2251) (Mac OS X only)
- `DataBrowserAddDragItemProcPtr` (page 2234)
- `DataBrowserAcceptDragProcPtr` (page 2233)
- `DataBrowserReceiveDragProcPtr` (page 2256)
- `DataBrowserPostProcessDragProcPtr` (page 2255)
- `DataBrowserGetContextualMenuProcPtr` (page 2238)
- `DataBrowserSelectContextualMenuProcPtr` (page 2257)
- `DataBrowserItemHelpContentProcPtr` (page 2248)

After you assign your callbacks to the appropriate field, call the function `SetDataBrowserCallbacks` (page 2200).

Note that this is a different set of callbacks from those that are assigned to fields in the `DataBrowserCustomCallbacks` data structure.

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X version 10.0 and later.

Not available to 64-bit applications.

Declared In

`HIDataBrowser.h`

InitDataBrowserCustomCallbacks

Initializes the data browser custom callback structure in preparation for adding your own callbacks for custom drawing or custom behavior to the structure.

```
OSStatus InitDataBrowserCustomCallbacks (
   DataBrowserCustomCallbacks *callbacks
);
```

Parameters

callbacks

A pointer to a `DataBrowserCustomCallbacks` structure. Before calling the function `InitDataBrowserCustomCallbacks`, set the `version` field of this structure to `kDataBrowserLatestCustomCallbacks`. On return, the fields in this structure are set to `NULL`.

Return Value

A result code. See "Data Browser Result Codes" (page 2298).

Discussion

Custom callbacks refer to those callback routines that are used to implement custom drawing or custom behavior in your data browser. The data browser API supports a limited set of built-in display types: text, icon and text, checkboxes, and so forth. If you want to display something else, you install custom callbacks to perform drawing and handle user interaction.

After you call the function `InitDataBrowserCustomCallbacks`, set the appropriate fields in the `DataBrowserCustomCallbacks` structure to your callbacks. The `DataBrowserCustomCallbacks` structure contains fields for the following:

- `DataBrowserDrawItemProcPtr` (page 2235)
- `DataBrowserEditItemProcPtr` (page 2237)
- `DataBrowserHitTestProcPtr` (page 2240)
- `DataBrowserTrackingProcPtr` (page 2259)
- `DataBrowserItemDragRgnProcPtr` (page 2246)
- `DataBrowserItemAcceptDragProcPtr` (page 2242)
- `DataBrowserItemReceiveDragProcPtr` (page 2254)

After you assign your custom callbacks to the appropriate field, call the function `SetDataBrowserCustomCallbacks` (page 2203).

Note that this is a different set of callbacks from those that are assigned to fields in the `DataBrowserCallbacks` data structure.

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X version 10.0 and later.

Not available to 64-bit applications.

Declared In

`HIDataBrowser.h`

InvokeDataBrowserAcceptDragUPP

Calls an accept-drag callback function.

```
Boolean InvokeDataBrowserAcceptDragUPP (
   ControlRef browser,
   DragReference theDrag,
   DataBrowserItemID item,
   DataBrowserAcceptDragUPP userUPP
);
```

Discussion
In most cases you do not need to use this function, because the system invokes your callback function for you. See the `DataBrowserAcceptDragProcPtr` (page 2233) callback function for more information and for a description of the parameters.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.

Declared In
`HIDataBrowser.h`

InvokeDataBrowserAddDragItemUPP

Calls an add-drag-item callback function.

```
Boolean InvokeDataBrowserAddDragItemUPP (
   ControlRef browser,
   DragReference theDrag,
   DataBrowserItemID item,
   ItemReference *itemRef,
   DataBrowserAddDragItemUPP userUPP
);
```

Discussion
In most cases you do not need to use this function, because the system invokes your callback function for you. See the `DataBrowserAddDragItemProcPtr` (page 2234) callback function for more information and for a description of the parameters.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.

Declared In
`HIDataBrowser.h`

InvokeDataBrowserDrawItemUPP

Calls a draw-item callback function.

```
void InvokeDataBrowserDrawItemUPP (
   ControlRef browser,
   DataBrowserItemID item,
   DataBrowserPropertyID property,
   DataBrowserItemState itemState,
   const Rect *theRect,
   SInt16 gdDepth,
   Boolean colorDevice,
   DataBrowserDrawItemUPP userUPP
);
```

Discussion
In most cases you do not need to use this function, because the system invokes your callback function for you. See the `DataBrowserDrawItemProcPtr` (page 2235) callback function for more information and for a description of the parameters.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.1 and later.

Declared In
`HIDataBrowser.h`

InvokeDataBrowserEditItemUPP

Calls an edit-item callback function.

```
Boolean InvokeDataBrowserEditItemUPP (
   ControlRef browser,
   DataBrowserItemID item,
   DataBrowserPropertyID property,
   CFStringRef theString,
   Rect *maxEditTextRect,
   Boolean *shrinkToFit,
   DataBrowserEditItemUPP userUPP
);
```

Discussion
In most cases you do not need to use this function, because the system invokes your callback function for you. See the `DataBrowserEditItemProcPtr` (page 2237) callback function for more information and for a description of the parameters.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.1 and later.

Declared In
`HIDataBrowser.h`

InvokeDataBrowserGetContextualMenuUPP

Calls a get-contextual-menu callback function.

```
void InvokeDataBrowserGetContextualMenuUPP (
   ControlRef browser,
   MenuRef *menu,
   UInt32 *helpType,
   CFStringRef *helpItemString,
   AEDesc *selection,
   DataBrowserGetContextualMenuUPP userUPP
);
```

Discussion
In most cases you do not need to use this function, because the system invokes your callback function for you. See the `DataBrowserGetContextualMenuProcPtr` (page 2238) callback function for more information and for a description of the parameters.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.

Declared In
`HIDataBrowser.h`

InvokeDataBrowserHitTestUPP

Calls a hit-test callback function.

```
Boolean InvokeDataBrowserHitTestUPP (
   ControlRef browser,
   DataBrowserItemID itemID,
   DataBrowserPropertyID property,
   const Rect *theRect,
   const Rect *mouseRect,
   DataBrowserHitTestUPP userUPP
);
```

Discussion
In most cases you do not need to use this function, because the system invokes your callback function for you. See the `DataBrowserHitTestProcPtr` (page 2240) callback function for more information and for a description of the parameters.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.1 and later.

Declared In
`HIDataBrowser.h`

InvokeDataBrowserItemAcceptDragUPP

Calls an item-accept-drag callback function.

```
DataBrowserDragFlags InvokeDataBrowserItemAcceptDragUPP (
   ControlRef browser,
   DataBrowserItemID itemID,
   DataBrowserPropertyID property,
   const Rect *theRect,
   DragReference theDrag,
   DataBrowserItemAcceptDragUPP userUPP
);
```

Discussion
In most cases you do not need to use this function, because the system invokes your callback function for you. See the `DataBrowserItemAcceptDragProcPtr` (page 2242) callback function for more information and for a description of the parameters.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.1 and later.

Declared In
`HIDataBrowser.h`

InvokeDataBrowserItemCompareUPP

Calls an item-comparison callback function.

```
Boolean InvokeDataBrowserItemCompareUPP (
   ControlRef browser,
   DataBrowserItemID itemOne,
   DataBrowserItemID itemTwo,
   DataBrowserPropertyID sortProperty,
   DataBrowserItemCompareUPP userUPP
);
```

Discussion
In most cases you do not need to use this function, because the system invokes your callback function for you. See the `DataBrowserItemCompareProcPtr` (page 2243) callback function for more information and for a description of the parameters.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.

Declared In
`HIDataBrowser.h`

InvokeDataBrowserItemDataUPP

Calls an item-data callback function.

```
OSStatus InvokeDataBrowserItemDataUPP (
   ControlRef browser,
   DataBrowserItemID item,
   DataBrowserPropertyID property,
   DataBrowserItemDataRef itemData,
   Boolean setValue,
   DataBrowserItemDataUPP userUPP
);
```

Discussion
In most cases you do not need to use this function, because the system invokes your callback function for you. See the `DataBrowserItemDataProcPtr` (page 2245) callback function for more information and for a description of the parameters.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.

Declared In
`HIDataBrowser.h`

InvokeDataBrowserItemDragRgnUPP

Calls an item-drag-region callback function.

```
void InvokeDataBrowserItemDragRgnUPP (
   ControlRef browser,
   DataBrowserItemID itemID,
   DataBrowserPropertyID property,
   const Rect *theRect,
   RgnHandle dragRgn,
   DataBrowserItemDragRgnUPP userUPP
);
```

Discussion
In most cases you do not need to use this function, because the system invokes your callback function for you. See the `DataBrowserItemDragRgnProcPtr` (page 2246) callback function for more information and for a description of the parameters.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.1 and later.

Declared In
`HIDataBrowser.h`

InvokeDataBrowserItemHelpContentUPP

Calls an item-help-content callback function.

```
void InvokeDataBrowserItemHelpContentUPP (
   ControlRef browser,
   DataBrowserItemID item,
   DataBrowserPropertyID property,
   HMContentRequest inRequest,
   HMContentProvidedType *outContentProvided,
   HMHelpContentRec *ioHelpContent,
   DataBrowserItemHelpContentUPP userUPP
);
```

Discussion
In most cases you do not need to use this function, because the system invokes your callback function for you. See the `DataBrowserItemHelpContentProcPtr` (page 2248) callback function for more information and for a description of the parameters.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.

Declared In
`HIDataBrowser.h`

InvokeDataBrowserItemNotificationUPP

Calls an item-notification callback function.

```
void InvokeDataBrowserItemNotificationUPP (
   ControlRef browser,
   DataBrowserItemID item,
   DataBrowserItemNotification message,
   DataBrowserItemNotificationUPP userUPP
);
```

Discussion
In most cases you don't need to use this function, because the system invokes your callback function for you. See the `DataBrowserItemNotificationProcPtr` (page 2250) callback function for more information and for a description of the parameters.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.

Declared In
`HIDataBrowser.h`

InvokeDataBrowserItemNotificationWithItemUPP

Calls an item-notification-with-data callback function.

```
void InvokeDataBrowserItemNotificationWithItemUPP (
   ControlRef browser,
   DataBrowserItemID item,
   DataBrowserItemNotification message,
   DataBrowserItemDataRef itemData,
   DataBrowserItemNotificationWithItemUPP userUPP
);
```

Discussion
In most cases you don't need to use this function, because the system invokes your callback function for you. See the `DataBrowserItemNotificationWithItemProcPtr` (page 2251) callback function for more information and for a description of the parameters.

Availability
Available in CarbonLib 1.5 and later.
Available in Mac OS X version 10.1 and later.

Declared In
`HIDataBrowser.h`

InvokeDataBrowserItemReceiveDragUPP

Calls an item-receive-drag callback function.

```
Boolean InvokeDataBrowserItemReceiveDragUPP (
   ControlRef browser,
   DataBrowserItemID itemID,
   DataBrowserPropertyID property,
   DataBrowserDragFlags dragFlags,
   DragReference theDrag,
   DataBrowserItemReceiveDragUPP userUPP
);
```

Discussion
In most cases you do not need to use this function, because the system invokes your callback function for you. See the `DataBrowserItemReceiveDragProcPtr` (page 2254) callback function for more information and for a description of the parameters.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.1 and later.

Declared In
`HIDataBrowser.h`

InvokeDataBrowserItemUPP

Calls an item-iterator callback function.

```
void InvokeDataBrowserItemUPP (
   DataBrowserItemID item,
   DataBrowserItemState state,
   void *clientData,
   DataBrowserItemUPP userUPP
);
```

Discussion
In most cases you do not need to use this function, because the system invokes your callback function for you. See the `DataBrowserItemProcPtr` (page 2253) callback function for more information and for a description of the parameters.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.

Declared In
`HIDataBrowser.h`

InvokeDataBrowserPostProcessDragUPP

Calls a postprocess-drag callback function.

```
void InvokeDataBrowserPostProcessDragUPP (
   ControlRef browser,
   DragReference theDrag,
   OSStatus trackDragResult,
   DataBrowserPostProcessDragUPP userUPP
);
```

Discussion
In most cases you do not need to use this function, because the system invokes your callback function for you. See the `DataBrowserPostProcessDragProcPtr` (page 2255) callback function for more information and for a description of the parameters.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.

Declared In
`HIDataBrowser.h`

InvokeDataBrowserReceiveDragUPP

Calls a receive-drag callback function.

```
Boolean InvokeDataBrowserReceiveDragUPP (
   ControlRef browser,
   DragReference theDrag,
   DataBrowserItemID item,
   DataBrowserReceiveDragUPP userUPP
);
```

Discussion
In most cases you do not need to use this function, because the system invokes your callback function for you. See the `DataBrowserReceiveDragProcPtr` (page 2256) callback function for more information and for a description of the parameters.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.

Declared In
`HIDataBrowser.h`

InvokeDataBrowserSelectContextualMenuUPP

Calls a select-contextual-menu callback function.

```
void InvokeDataBrowserSelectContextualMenuUPP (
   ControlRef browser,
   MenuRef menu,
   UInt32 selectionType,
   SInt16 menuID,
   MenuItemIndex menuItem,
   DataBrowserSelectContextualMenuUPP userUPP
);
```

Discussion
In most cases you do not need to use this function, because the system invokes your callback function for you. See the `DataBrowserSelectContextualMenuProcPtr` (page 2257) callback function for more information and for a description of the parameters.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.

Declared In
`HIDataBrowser.h`

InvokeDataBrowserTrackingUPP

Calls a tracking callback function.

```
DataBrowserTrackingResult InvokeDataBrowserTrackingUPP (
   ControlRef browser,
   DataBrowserItemID itemID,
   DataBrowserPropertyID property,
   const Rect *theRect,
   Point startPt,
   EventModifiers modifiers,
   DataBrowserTrackingUPP userUPP
);
```

Discussion
In most cases you do not need to use this function, because the system invokes your callback function for you. See the `DataBrowserTrackingProcPtr` (page 2259) callback function for more information and for a description of the parameters.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.1 and later.

Declared In
`HIDataBrowser.h`

IsDataBrowserItemSelected

Checks to see if a data item is selected.

```
Boolean IsDataBrowserItemSelected (
   ControlRef browser,
   DataBrowserItemID item
);
```

Parameters

browser
A data browser.

item
The item ID of the item to check.

Return Value
A value of `true` if the item is a member of the current selection.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

MoveDataBrowserSelectionAnchor

Moves or extends the current selection.

```
OSStatus MoveDataBrowserSelectionAnchor (
   ControlRef browser,
   DataBrowserSelectionAnchorDirection direction,
   Boolean extendSelection
);
```

Parameters

browser

A data browser.

direction

The direction to move or extend the current selection. You can pass any one of the constants described in "Selection Anchor Directions" (page 2293).

extendSelection

On input, a value that specifies whether to extend the current selection (`true`) or move the selection (`false`).

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

NewDataBrowserAcceptDragUPP

Creates a universal procedure pointer to an accept-drag callback function.

```
DataBrowserAcceptDragUPP NewDataBrowserAcceptDragUPP (
   DataBrowserAcceptDragProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your accept-drag callback function.

Return Value
The universal procedure pointer.

Discussion
See the `DataBrowserAcceptDragProcPtr` (page 2233) callback function.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.

Declared In
`HIDataBrowser.h`

NewDataBrowserAddDragItemUPP

Creates a universal procedure pointer to an add-drag-item callback function.

```
DataBrowserAddDragItemUPP NewDataBrowserAddDragItemUPP (
   DataBrowserAddDragItemProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your add-drag-item callback function.

Return Value

The universal procedure pointer.

Discussion

See the `DataBrowserAddDragItemProcPtr` (page 2234) callback function.

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X version 10.0 and later.

Declared In

`HIDataBrowser.h`

NewDataBrowserDrawItemUPP

Creates a universal procedure pointer to a draw-item callback function.

```
DataBrowserDrawItemUPP NewDataBrowserDrawItemUPP (
   DataBrowserDrawItemProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your draw-item callback function.

Return Value

The universal procedure pointer.

Discussion

See the `DataBrowserDrawItemProcPtr` (page 2235) callback function.

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X version 10.1 and later.

Declared In

`HIDataBrowser.h`

NewDataBrowserEditItemUPP

Creates a universal procedure pointer to an edit-item callback function.

```
DataBrowserEditItemUPP NewDataBrowserEditItemUPP (
   DataBrowserEditItemProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your edit-item callback function.

Return Value

The universal procedure pointer.

Discussion

See the `DataBrowserEditItemProcPtr` (page 2237) callback function.

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X version 10.1 and later.

Declared In

`HIDataBrowser.h`

NewDataBrowserGetContextualMenuUPP

Creates a universal procedure pointer to a get-contextual-menu callback function.

```
DataBrowserGetContextualMenuUPP NewDataBrowserGetContextualMenuUPP (
   DataBrowserGetContextualMenuProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your get-contextual-menu callback function.

Return Value

The universal procedure pointer.

Discussion

See the `DataBrowserGetContextualMenuProcPtr` (page 2238) callback function.

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X version 10.0 and later.

Declared In

`HIDataBrowser.h`

NewDataBrowserHitTestUPP

Creates a universal procedure pointer to a hit-test callback function.

```
DataBrowserHitTestUPP NewDataBrowserHitTestUPP (
   DataBrowserHitTestProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your hit-test callback function.

Return Value
The universal procedure pointer.

Discussion
See the `DataBrowserHitTestProcPtr` (page 2240) callback function.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.1 and later.

Declared In
`HIDataBrowser.h`

NewDataBrowserItemAcceptDragUPP

Creates a universal procedure pointer to an item-accept-drag callback function.

```
DataBrowserItemAcceptDragUPP NewDataBrowserItemAcceptDragUPP (
   DataBrowserItemAcceptDragProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your item-accept-drag callback function.

Return Value
The universal procedure pointer.

Discussion
See the `DataBrowserItemAcceptDragProcPtr` (page 2242) callback function.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.1 and later.

Declared In
`HIDataBrowser.h`

NewDataBrowserItemCompareUPP

Creates a universal procedure pointer to an item-comparison callback function.

```
DataBrowserItemCompareUPP NewDataBrowserItemCompareUPP (
   DataBrowserItemCompareProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your item-comparison callback function.

Return Value

The universal procedure pointer.

Discussion

See the `DataBrowserItemCompareProcPtr` (page 2243) callback function.

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X version 10.0 and later.

Declared In

`HIDataBrowser.h`

NewDataBrowserItemDataUPP

Creates a universal procedure pointer to an item-data callback function.

```
DataBrowserItemDataUPP NewDataBrowserItemDataUPP (
   DataBrowserItemDataProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your item-data callback function.

Return Value

The universal procedure pointer.

Discussion

See the `DataBrowserItemDataProcPtr` (page 2245) callback function.

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X version 10.0 and later.

Declared In

`HIDataBrowser.h`

NewDataBrowserItemDragRgnUPP

Creates a universal procedure pointer to an item-drag-region callback function.

```
DataBrowserItemDragRgnUPP NewDataBrowserItemDragRgnUPP (
   DataBrowserItemDragRgnProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your item-drag-region callback function.

Return Value

The universal procedure pointer.

Discussion

See the `DataBrowserItemDragRgnProcPtr` (page 2246) callback function.

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X version 10.1 and later.

Declared In

`HIDataBrowser.h`

NewDataBrowserItemHelpContentUPP

Creates a universal procedure pointer to an item-help-content callback function.

```
DataBrowserItemHelpContentUPP NewDataBrowserItemHelpContentUPP (
   DataBrowserItemHelpContentProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your item-help-content callback function.

Return Value

The universal procedure pointer.

Discussion

See the `DataBrowserItemHelpContentProcPtr` (page 2248) callback function.

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X version 10.0 and later.

Declared In

`HIDataBrowser.h`

NewDataBrowserItemNotificationUPP

Creates a universal procedure pointer to an item-notification callback function.

```
DataBrowserItemNotificationUPP NewDataBrowserItemNotificationUPP (
   DataBrowserItemNotificationProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your item-notification callback function.

Return Value

The universal procedure pointer.

Discussion

See the `DataBrowserItemNotificationProcPtr` (page 2250) callback function.

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X version 10.0 and later.

Declared In

`HIDataBrowser.h`

NewDataBrowserItemNotificationWithItemUPP

Creates a universal procedure pointer to an item-notification-with-data callback function.

```
DataBrowserItemNotificationWithItemUPP NewDataBrowserItemNotificationWithItemUPP
(
   DataBrowserItemNotificationWithItemProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your item-notification-with-data callback function.

Return Value

The universal procedure pointer.

Discussion

See the `DataBrowserItemNotificationWithItemProcPtr` (page 2251) callback function.

Availability

Available in CarbonLib 1.5 and later.

Available in Mac OS X version 10.1 and later.

Declared In

`HIDataBrowser.h`

NewDataBrowserItemReceiveDragUPP

Creates a universal procedure pointer to an item-receive-drag callback function.

```
DataBrowserItemReceiveDragUPP NewDataBrowserItemReceiveDragUPP (
   DataBrowserItemReceiveDragProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your item-receive-drag callback function.

Return Value
The universal procedure pointer.

Discussion
See the `DataBrowserItemReceiveDragProcPtr` (page 2254) callback function.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.1 and later.

Declared In
`HIDataBrowser.h`

NewDataBrowserItemUPP

Creates a universal procedure pointer to an item-iterator callback function.

```
DataBrowserItemUPP NewDataBrowserItemUPP (
   DataBrowserItemProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your item-iterator callback function.

Return Value
The universal procedure pointer.

Discussion
See the `DataBrowserItemProcPtr` (page 2253) callback function.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.

Declared In
`HIDataBrowser.h`

NewDataBrowserPostProcessDragUPP

Creates a universal procedure pointer to a postprocess-drag callback function.

```
DataBrowserPostProcessDragUPP NewDataBrowserPostProcessDragUPP (
   DataBrowserPostProcessDragProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your postprocess-drag callback function.

Return Value

The universal procedure pointer.

Discussion

See the `DataBrowserPostProcessDragProcPtr` (page 2255) callback function.

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X version 10.0 and later.

Declared In

`HIDataBrowser.h`

NewDataBrowserReceiveDragUPP

Creates a universal procedure pointer to a receive-drag callback function.

```
DataBrowserReceiveDragUPP NewDataBrowserReceiveDragUPP (
   DataBrowserReceiveDragProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your receive-drag callback function.

Return Value

The universal procedure pointer.

Discussion

See the `DataBrowserReceiveDragProcPtr` (page 2256) callback function.

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X version 10.0 and later.

Declared In

`HIDataBrowser.h`

NewDataBrowserSelectContextualMenuUPP

Creates a universal procedure pointer to a select-contextual-menu callback function.

```
DataBrowserSelectContextualMenuUPP NewDataBrowserSelectContextualMenuUPP (
   DataBrowserSelectContextualMenuProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your select-contextual-menu callback function.

Return Value
The universal procedure pointer.

Discussion
See the `DataBrowserSelectContextualMenuProcPtr` (page 2257) callback function.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.

Declared In
`HIDataBrowser.h`

NewDataBrowserTrackingUPP

Creates a universal procedure pointer to a tracking callback function.

```
DataBrowserTrackingUPP NewDataBrowserTrackingUPP (
   DataBrowserTrackingProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your tracking callback function.

Return Value
The universal procedure pointer.

Discussion
See the `DataBrowserTrackingProcPtr` (page 2259) callback function.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.1 and later.

Declared In
`HIDataBrowser.h`

OpenDataBrowserContainer

Opens a data browser container.

```
OSStatus OpenDataBrowserContainer (
   ControlRef browser,
   DataBrowserItemID container
);
```

Parameters

browser

A data browser.

container

The item ID of the container to open.

Return Value

A result code. See "Data Browser Result Codes" (page 2298).

Discussion

Normally the user navigates through a data hierarchy by clicking the disclosure triangle next to a container item in list view, or the container item (such as a folder icon) in column view. In either of these cases, the system automatically opens or closes the container. Under some circumstances your application may need to open or close a container programmatically, such as when you are restoring a display to its last known state. In such cases, you can call the function `OpenDataBrowserContainer` to disclose items in a container or the function `CloseDataBrowserContainer` (page 2124) to hide items in a container.

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X version 10.0 and later.

Not available to 64-bit applications.

Declared In

`HIDataBrowser.h`

RemoveDataBrowserItems

Removes one or more items from a data browser.

```
OSStatus RemoveDataBrowserItems (
   ControlRef browser,
   DataBrowserItemID container,
   ItemCount numItems,
   const DataBrowserItemID *items,
   DataBrowserPropertyID preSortProperty
);
```

Parameters

browser

A data browser.

container

An item ID or the constant `kDataBrowserNoItem`. Pass the item ID that uniquely identifies the container from which you want to remove items. Pass `kDataBrowserNoItem` to remove items from the root container.

numItems

The number of items in the array pointed to by the `items` parameter. To remove all items pass `0` and also pass `NULL` in the items parameter.

items

A pointer to an array of item ID values for the items you want to remove from the data browser. You can delete an arbitrary list of items from a container. To remove all items, pass `NULL`, and also pass `0` in the `numItems` parameter.

preSortProperty

The property ID of the column whose sorting order is the same as the sorting order of the `items` array. A property ID is a value that identifies a column independent of its position in a data browser. Pass `kDataBrowserItemNoProperty` if the `items` array is not sorted or if you don't know the sorting order. You'll get the best performance from this function if you provide a sorting order.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

RemoveDataBrowserTableViewColumn

Removes a column from a list view data browser.

```
OSStatus RemoveDataBrowserTableViewColumn (
   ControlRef browser,
   DataBrowserTableViewColumnID column
);
```

Parameters

browser

A data browser.

column

The property ID for the list view column you want to remove. The `DataBrowserTableViewColumnID` data type is the same as the `DataBrowserPropertyID` data type.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
This function works only for list view.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

RevealDataBrowserItem

Scrolls an item into view, optionally bringing a particular part of that item into view.

```
OSStatus RevealDataBrowserItem (
   ControlRef browser,
   DataBrowserItemID item,
   DataBrowserPropertyID propertyID,
   DataBrowserRevealOptions options
);
```

Parameters

browser

A data browser.

item

The item ID of the item to scroll into view.

propertyID

The property ID of the column to scroll into view. A property ID is a four-character sequence that you assign to represent a column in list view. For column view, pass `kDataBrowserNoItem`.

options

A value that specifies how to position the item in the data browser. See "Reveal Options" (page 2292) for a list of the constants you can supply.

Return Value

A result code. See "Data Browser Result Codes" (page 2298).

Discussion

In most cases the system takes care of scrolling for you. However, this function is useful if your application supports type-select and you want to scroll a matching item into view.

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X version 10.0 and later.

Not available to 64-bit applications.

Declared In

`HIDataBrowser.h`

SetDataBrowserActiveItems

Sets what determines the active state of the items in a data browser.

```
OSStatus SetDataBrowserActiveItems (
   ControlRef browser,
   Boolean active
);
```

Parameters

browser

A data browser.

active

A value that specifies the new active state for the items displayed in the list. Pass `true` to make the active state of each item determined by what your callback reports for each item's `kDataBrowserItemIsActiveProperty` property, or `false` to make all items inactive. Inactive items appear dimmed.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
Passing `true` for the `active` parameter does not make all the items active. Instead it sets the active state of each individual item according to the value associated with the `kDataBrowserItemIsActiveProperty` property for that item. This means if the active property for an item is set to `false`, and you pass `true` for the `active` parameter, then the item is inactive.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

SetDataBrowserCallbacks

Sets the callback routines to use with a data browser, replacing any previously installed callbacks.

```
OSStatus SetDataBrowserCallbacks (
   ControlRef browser,
   const DataBrowserCallbacks *callbacks
);
```

Parameters

browser

A data browser.

callbacks

A pointer to a `DataBrowserCallbacks` structure that is filled out with universal procedure pointers (UPPs) to the callback routines your application provides. At a minimum, you need to provide a UPP to an item-data callback (`DataBrowserItemDataProcPtr` (page 2245)).

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
Before calling the function `SetDataBrowserCallbacks` you must first call `InitDataBrowserCallbacks` (page 2175) to initialize the data browser callback structure. Calling `SetDataBrowserCallbacks` replaces any callback routines you installed previously by calling this function.

You can supply the following callbacks. If you don't supply callbacks in cases for which it's optional, you get the default behavior provided by the data browser API.

- DataBrowserItemDataProcPtr (page 2245). You must provide this callback because communicates the text, icons, or other data to display in list view. It also communicates the metadata that defines how data is displayed, such as whether or not an item is a container or has a parent. If you set up your data browser to allow the user to edit, this callback informs your application when the user makes a change.
- DataBrowserItemCompareProcPtr (page 2243). You must provide a sorting callback if you want users to be able to sort the items in a column. If you want containers in a hierarchical list to be sorted independently, then you must provide a sorting callback that handles the hierarchical lists appropriately.
- DataBrowserItemNotificationProcPtr (page 2250). You must provide this (or the next) callback if you have hierarchical data in a list, or if you use column view.
- DataBrowserItemNotificationWithItemProcPtr (page 2251) (Mac OS X only)
- DataBrowserAddDragItemProcPtr (page 2234). You can provide this callback to allow dragging out of your data browser.
- DataBrowserAcceptDragProcPtr (page 2233). You can provide this callback to allow dragging into your data browser; use this to accept a drag item.
- DataBrowserReceiveDragProcPtr (page 2256). You can provide this callback to allow dragging into your data browser; use this to receive a drag item.
- DataBrowserPostProcessDragProcPtr (page 2255). If you provide callbacks to allow dragging into your data browser, you can optionally provide a postprocess-drag callback to perform cleanup tasks.
- DataBrowserGetContextualMenuProcPtr (page 2238). You can optionally support a contextual menu. If so, you'll need to provide the next callback too.
- DataBrowserSelectContextualMenuProcPtr (page 2257)
- DataBrowserItemHelpContentProcPtr (page 2248). You can optionally provide help tags.

Note that this function sets a different set of callbacks from those that are set by calling the function SetDataBrowserCustomCallbacks (page 2203).

To replace a callback, you first need to get the current set of callbacks by calling the function GetDataBrowserCallbacks (page 2140). Set the appropriate fields in the DataBrowserCallbacks structure to your callback. Then you call the function SetDataBrowserCallbacks. Your application can set as many callbacks as appropriate.

The following code shows how to assign UPPs to the callbacks structure and then call the function SetDataBrowserCallbacks. The code assumes you have already called the function InitDataBrowserCallbacks (page 2175) to initialize the data browser callback structure.

```
myCallbacks.u.v1.itemNotificationCallback =
        NewDataBrowserItemNotificationUPP (MyItemNotificationCallback);

myCallbacks.u.v1.acceptDragCallback =
        NewDataBrowserAcceptDragUPP (MyAcceptDragCallback);
myCallbacks.u.v1.receiveDragCallback =
        NewDataBrowserReceiveDragUPP (MyReceiveDragCallback);
myCallbacks.u.v1.addDragItemCallback =
        NewDataBrowserAddDragItemUPP (MyAddDragItemCallback);
myCallbacks.u.v1.itemHelpContentCallback =
        NewDataBrowserItemHelpContentUPP (MyItemHelpContentCallback);
myCallbacks.u.v1.getContextualMenuCallback =
        NewDataBrowserGetContextualMenuUPP (MyGetContextualMenuCallback);
myCallbacks.u.v1.selectContextualMenuCallback =
```

```
        NewDataBrowserSelectContextualMenuUPP (
                    MySelectContextualMenuCallback);
SetDataBrowserCallbacks (browser, &myCallbacks);
```

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

SetDataBrowserColumnViewDisplayType

Sets the display type for a data browser in column view.

```
OSStatus SetDataBrowserColumnViewDisplayType (
   ControlRef browser,
   DataBrowserPropertyType propertyType
);
```

Parameters

`browser`
A data browser.

`propertyType`
The data type to be displayed in the data browser. The default is `kDataBrowserIconAndTextType`. Currently this is the only value you can supply for column view.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
This function is effectively not functional because no display types other than `kDataBrowserIconAndTextType` are currently supported. Note that the rightmost column can have the attribute `kDataBrowserColumnViewPreviewProperty` as long as you provide a callback to display the appropriate icon and text information in the preview column.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

SetDataBrowserColumnViewPath

Sets a path for a column view.

```
OSStatus SetDataBrowserColumnViewPath (
   ControlRef browser,
   UInt32 length,
   const DataBrowserItemID *path
);
```

Parameters

browser

A data browser.

length

The number of items in the array passed in the `path` parameter.

path

The address to the first item in the array of item ID values that specifies the path. Array element `0` is the root; array element `N - 1` is the target.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

SetDataBrowserCustomCallbacks

Sets the custom callback routines to use with a data browser, replacing any previously installed custom callbacks.

```
OSStatus SetDataBrowserCustomCallbacks (
   ControlRef browser,
   const DataBrowserCustomCallbacks *callbacks
);
```

Parameters

browser

A data browser.

callbacks

A pointer to a `DataBrowserCustomCallbacks` structure that is filled out with universal procedure pointers (UPPs) to the custom callback routines your application provides.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
Before calling the function `SetDataBrowserCustomCallbacks` you must first call `InitDataBrowserCustomCallbacks` (page 2176) to initialize the data browser custom callback structure. Calling `SetDataBrowserCustomCallbacks` replaces any callback routines you installed previously by calling this function.

You can supply the following custom callback routines.

- DataBrowserDrawItemProcPtr (page 2235). This callback is invoked by the data browser whenever your content needs to be drawn. You must supply this callback for data whose display type is kDataBrowserCustomType.
- DataBrowserEditItemProcPtr (page 2237). Supply this callback when you want to support editing of your content.
- DataBrowserHitTestProcPtr (page 2240). You can provide this callback to determine if the pointer is over content that can be selected or dragged.
- DataBrowserTrackingProcPtr (page 2259). This callback implements custom tracking behavior.
- DataBrowserItemDragRgnProcPtr (page 2246). You can supply this callback when you need to determine which part of an item to use to create a transparent image for a dragged item.
- DataBrowserItemAcceptDragProcPtr (page 2242). This callback determines if an item can accept a drag object.
- DataBrowserItemReceiveDragProcPtr (page 2254). This callback receives a drop over an item.

Note that this is a different set of callbacks from those that are installed by calling the function SetDataBrowserCallbacks (page 2200).

To replace a callback, you first need to set the appropriate fields in the DataBrowserCustomCallbacks structure to your callbacks. Then you call the function SetDataBrowserCustomCallbacks. The following code shows how to set custom callbacks. It assumes you have already called the function InitDataBrowserCustomCallbacks (page 2176) to initialize the data browser custom callback structure. Your application can set as many callbacks as appropriate.

```
myCustomCallbacks.u.v1.drawItemCallback =
        NewDataBrowserDrawItemUPP (MyDataBrowserDrawItemCallback);
myCustomCallbacks.u.v1.editItemCallback =
        NewDataBrowserEditItemUPP (MyDataBrowserEditItemCallback);
SetDataBrowserCustomCallbacks (browser,&myCustomCallbacks);
```

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
HIDataBrowser.h

SetDataBrowserEditItem

Programmatically starts or ends an editing session.

```
OSStatus SetDataBrowserEditItem (
   ControlRef browser,
   DataBrowserItemID item,
   DataBrowserPropertyID property
);
```

Parameters

browser
A data browser.

item
The item ID of the item to make editable.

property
The property ID of the item to make editable.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
You can call the function `SetDataBrowserEditItem` to begin or end an editing session programmatically for a text item. To begin an editing session for a text item, specify its item ID and property ID. To end an editing session, provide the constant `kDataBrowserNoItem` as the item ID number.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

SetDataBrowserEditText

Modifies the displayed contents of a text item while it is being edited.

```
OSStatus SetDataBrowserEditText (
   ControlRef browser,
   CFStringRef text
);
```

Parameters

browser
A data browser.

text
A CFString object that specifies the text to edit. The data browser makes its own copy of this object so it is safe to release your own reference after you call this function.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
You can use this function to programmatically change the text. For example, to paste data in response to a Paste command. This function is useful only if an edit session is in progress for an item. You can check whether can get session is open by calling the function `GetDataBrowserEditItem` (page 2143).

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

SetDataBrowserHasScrollBars

Sets the display state of horizontal and vertical scroll bars for a list view data browser.

```
OSStatus SetDataBrowserHasScrollBars (
   ControlRef browser,
   Boolean horiz,
   Boolean vert
);
```

Parameters

browser

A list view data browser.

horiz

A value that specifies whether to display the browser control with (`true`) or without (`false`) a horizontal scroll bar.

vert

A value that specifies whether to display the browser control with (`true`) or without (`false`) a vertical scroll bar.

Return Value

A result code. See "Data Browser Result Codes" (page 2298).

Discussion

If the list your application displays is small and its coordinates do not extend beyond the bounds of the area used to display the list, then you can call `SetDataBrowserHasScrollBars` to turn off the display of scroll bars.

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X version 10.0 and later.

Not available to 64-bit applications.

Declared In

`HIDataBrowser.h`

SetDataBrowserItemDataBooleanValue

Specifies a Boolean value for an item.

```
OSStatus SetDataBrowserItemDataBooleanValue (
   DataBrowserItemDataRef itemData,
   Boolean theData
);
```

Parameters

itemData

The item data reference for the item whose Boolean value you want to set. The item data reference is passed to the callback routine from which you are calling the function `SetDataBrowserItemDataBooleanValue`.

theData

The value to display.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
Your item-data callback calls this function in response to an inquiry for the following properties:

- `kDataBrowserItemIsActiveProperty`
- `kDataBrowserItemIsSelectableProperty`
- `kDataBrowserItemIsEditableProperty`
- `kDataBrowserItemIsContainerProperty`
- `kDataBrowserItemIsOpenableProperty`
- `kDataBrowserItemIsClosableProperty`
- `kDataBrowserItemIsSortableProperty`

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

SetDataBrowserItemDataButtonValue

Specifies a checkbox value.

```
OSStatus SetDataBrowserItemDataButtonValue (
   DataBrowserItemDataRef itemData,
   ThemeButtonValue theData
);
```

Parameters

itemData
The item data reference for the item whose checkbox value you want to set. The item data reference is passed to the callback routine from which you are calling the function `SetDataBrowserItemDataButtonValue`.

theData
The checkbox setting. You can supply any of the following theme button value constants defined by the Appearance Manager:

- `kThemeButtonOff` draws a checkbox that is not selected.
- `kThemeButtonOn` draws a checkbox that is selected.
- `kThemeButtonMixed` draws a checkbox that in a mixed state, indicating that a setting is on for some items in a selection and off for others.

See *Appearance Manager Reference* for more information.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
Your item-data callback calls this function in response to a set-data request for items that have the display type `kDataBrowserCheckboxType`.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

SetDataBrowserItemDataDateTime

Specifies, as a 32-bit value, a date and time value to display.

```
OSStatus SetDataBrowserItemDataDateTime (
   DataBrowserItemDataRef itemData,
   SInt32 theData
);
```

Parameters

itemData
The item data reference for the item whose date and time value you want to set. The item data reference is passed to the callback routine from which you are calling the function `SetDataBrowserItemDataDateTime`.

theData
A 32-bit value that represents the number of elapsed seconds since midnight, January 1, 1904. For more information about date and time encodings used in the Mac OS, see *Date, Time, and Measurement Utilities Reference*.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
This function works only with items that have the property `kDataBrowserDateTimeType`. If the column has the property `kDataBrowserRelativeDateTime`, the date is displayed relative to the current time for the computer. For example, a time 24 hours before the current time is displayed as "Yesterday." Other examples of relative date and time values are "Today, 1:45 PM" and "Yesterday, 7:30 AM."

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

SetDataBrowserItemDataDrawState

Specifies whether to draw a checkbox in the active or inactive state.

```
OSStatus SetDataBrowserItemDataDrawState (
   DataBrowserItemDataRef itemData,
   ThemeDrawState theData
);
```

Parameters

itemData

The item data reference for the item whose drawing state you want to set. This value is passed to the callback routine from which you are calling the function `SetDataBrowserItemDataDrawState`.

theData

The drawing state to use for the checkbox item. You can supply the following theme drawing state constants:

- `kThemeStateInactive` draws the item in the inactive state.
- `kThemeStateActive` draws the item in the active state.

See *Appearance Manager Reference* for more information on these constants.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
Your item-data callback calls this function in response to a set-data request for items that have the display type `kDataBrowserCheckboxType`.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

SetDataBrowserItemDataIcon

Specifies the icon to draw.

```
OSStatus SetDataBrowserItemDataIcon (
   DataBrowserItemDataRef itemData,
   IconRef theData
);
```

Parameters

itemData

The item data reference for the item whose icon you want to set. This value is passed to the callback routine from which you are calling the function `SetDataBrowserItemDataIcon`.

theData

The icon to display. The data browser retains the icon, so you may release the `IconRef` after the function returns.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
You call the function `SetDataBrowserItemDataIcon` from within a `DataBrowserItemDataProcPtr` (page 2245) callback routine to specify an icon to draw. You can specify an icon for any column that has the `kDataBrowserIconType` display type or the `kDataBrowserIconAndTextType` display type.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

SetDataBrowserItemDataIconTransform

Specifies a transformation to apply to an icon when it is drawn.

```
OSStatus SetDataBrowserItemDataIconTransform (
   DataBrowserItemDataRef itemData,
   IconTransformType theData
);
```

Parameters

itemData
The item data reference for the item whose icon transformation you want to set. This value is passed to the callback routine from which you are calling the function `SetDataBrowserItemDataIconTransform`.

theData
An icon transformation type that specifies how to modify the appearance of the icon. You can pass any of the icon transformation constants defined by Icon Services and Utilities. See *Icon Services and Utilities Reference* for more information.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
This function works only with items that either have the property `kDataBrowserIconAndTextType` or `kDataBrowserIconType`.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

SetDataBrowserItemDataItemID

Communicates a property of an item when that property is another item's ID.

```
OSStatus SetDataBrowserItemDataItemID (
   DataBrowserItemDataRef itemData,
   DataBrowserItemID theData
);
```

Parameters

itemData

The item data reference passed to your item-data callback.

theData

The item ID to set.

Return Value

A result code. See "Data Browser Result Codes" (page 2298).

Discussion

To display hierarchical data correctly the data browser needs to know whether an item is a container and whether the item is in a container (has a parent). So it sends a get-data request for the properties `kDataBrowserParentContainerProperty` and `kDataBrowserContainerAliasIDProperty` to your item-data callback.

The property `kDataBrowserContainerAliasIDProperty` is sent to your item-data callback to provide your application with a chance to follow an alias that the item might represent. If the incoming item is an alias to another item, you can call `SetDataBrowserItemDataItemID` to inform the data browser which other item the incoming item points to.

The property `kDataBrowserParentContainerProperty` is sent to your item-data callback to check whether an item has a parent. If it does, you call `SetDataBrowserItemDataItemID`, supplying the item ID of the parent in the parameter `theData`. If the item has no parent, set `theData` to `0`.

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X version 10.0 and later.

Not available to 64-bit applications.

Declared In

`HIDataBrowser.h`

SetDataBrowserItemDataLongDateTime

Specifies, as a 64-bit value, a date and time value to display.

```
OSStatus SetDataBrowserItemDataLongDateTime (
   DataBrowserItemDataRef itemData,
   const LongDateTime *theData
);
```

Parameters

itemData

The item data reference for the item whose long date and time value you want to set. The item data reference is passed to the callback routine from which you are calling the function `SetDataBrowserItemDataLongDateTime`.

theData

A pointer to a 64-bit value that represents the time as the number of elapsed seconds since midnight, January 1, 1904. For more information about date and time encodings used in the Mac OS, see *Date, Time, and Measurement Utilities Reference* in Carbon Text & International Documentation.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
This function works only with items that have the property `kDataBrowserDateTimeType`. If the column has the property `kDataBrowserRelativeDateTime`, the date is displayed relative to the current time for the computer. For example, a time 24 hours before the current time is displayed as "Yesterday." Other examples of relative date and time values are "Today, 1:45 PM" and "Yesterday, 7:30 AM."

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

SetDataBrowserItemDataMaximum

Specifies the maximum integer value that can be displayed for an item; useful for such display types as sliders, progress bars, relevance indicators, and pop-up menus.

```
OSStatus SetDataBrowserItemDataMaximum (
   DataBrowserItemDataRef itemData,
   SInt32 theData
);
```

Parameters

itemData

The item data reference for the item whose maximum value you want to set. The item data reference is passed to the callback routine from which you are calling the function `SetDataBrowserItemDataMaximum`.

theData

The maximum setting for content displayed for the item.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

SetDataBrowserItemDataMenuRef

Sets the pop-up menu to display.

```
OSStatus SetDataBrowserItemDataMenuRef (
   DataBrowserItemDataRef itemData,
   MenuRef theData
);
```

Parameters

itemData

The item data reference for the item whose pop-up menu value you want to set. The item data reference is passed to the callback routine from which you are calling the function `SetDataBrowserItemDataMenuRef`.

theData

The pop-up menu set to the value you want to display. The system retains the menu reference that you pass; you must release it when you no longer need it. Pass `NULL` if you no longer want a menu.

Return Value

A result code. See "Data Browser Result Codes" (page 2298).

Discussion

Your item-data callback calls this function in response to a set-data request for an item whose display type is `kDataBrowserPopupMenuType`.

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X version 10.0 and later.

Not available to 64-bit applications.

Declared In

`HIDataBrowser.h`

SetDataBrowserItemDataMinimum

Specifies the minimum integer value that can be displayed for an item; useful for such display types as sliders, progress bars, relevance indicators, and pop-up menus.

```
OSStatus SetDataBrowserItemDataMinimum (
   DataBrowserItemDataRef itemData,
   SInt32 theData
);
```

Parameters

itemData

The item data reference for the item whose minimum value you want to set. The item data reference is passed to the callback routine from which you are calling the function `SetDataBrowserItemDataMinimum`.

theData

The minimum setting for the displayed content.

Return Value

A result code. See "Data Browser Result Codes" (page 2298).

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

SetDataBrowserItemDataRGBColor

Specifies a color to use when drawing an item.

```
OSStatus SetDataBrowserItemDataRGBColor (
   DataBrowserItemDataRef itemData,
   const RGBColor *theData
);
```

Parameters

itemData

The item data reference for the item whose color you want to set. This value is passed to the callback routine from which you are calling the function `SetDataBrowserItemDataRGBColor`.

theData

A pointer to the RGB values that specify the color to use.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
Typically this function is used to set the color for an item that is an icon type. Your item-data callback calls this function in response to a set-data request for items that have display type `kDataBrowserIconType` or `kDataBrowserIconAndTextType`.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

SetDataBrowserItemDataText

Specifies the text to draw.

```
OSStatus SetDataBrowserItemDataText (
   DataBrowserItemDataRef itemData,
   CFStringRef theData
);
```

Parameters

itemData

The item data reference for the item whose text you want to set. This value is passed to the callback routine from which you are calling the function `SetDataBrowserItemDataText`.

theData

The CFString object that contains the text you want to draw. You are responsible for releasing the CFString object.

Return Value

A result code. See "Data Browser Result Codes" (page 2298).

Discussion

You call the function `SetDataBrowserItemDataText` from inside a data callback routine when the item being drawn is inside a column that has the `kDataBrowserTextType` display type or the `kDataBrowserIconAndTextType` display type.

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X version 10.0 and later.

Not available to 64-bit applications.

Declared In

`HIDataBrowser.h`

SetDataBrowserItemDataValue

Sets the value of an item; useful for such display types as sliders, progress bars, relevance indicators, and pop-up menus.

```
OSStatus SetDataBrowserItemDataValue (
   DataBrowserItemDataRef itemData,
   SInt32 theData
);
```

Parameters

itemData

The item data reference for the item whose integer value you want to set. The item data reference is passed to the callback routine from which you are calling the function `SetDataBrowserItemDataValue`.

theData

The value to display. The value must be between the minimum and maximum values specified by calling the functions `SetDataBrowserItemDataMinimum` (page 2213) and `SetDataBrowserItemDataMaximum` (page 2212). Values displayed by a progress bar can vary between the minimum and maximum values, inclusive.

Return Value

A result code. See "Data Browser Result Codes" (page 2298).

Discussion
Your application calls the function `SetDataBrowserItemDataValue` to set a new value for a display type when your item-data callback routine is called with the `setValue` parameter set to `false`.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

SetDataBrowserListViewDisclosureColumn

Specifies whether there is a column that has disclosure triangles and, if so, which column.

```
OSStatus SetDataBrowserListViewDisclosureColumn (
   ControlRef browser,
   DataBrowserTableViewColumnID column,
   Boolean expandableRows
);
```

Parameters

browser
A data browser.

column
The property ID for the column for which you want to set as disclosure column. Only one column in list view can be designated as a disclosure column. Pass `kDataBrowserNoItemProperty` if you do not want a disclosure column. The `DataBrowserTableViewColumnID` data type is the same as the `DataBrowserPropertyID` data type.

expandableRows
A value that specifies how a disclosed row behaves. Pass `true` to have a container open as a single row with an expanded height. Pass `false` to have a container opens to expose other rows. See the Discussion for more details on expandable rows.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
A disclosure triangle next to an item denotes the item is a container. You can use the `expandableRows` parameter to specify whether an opened container displays its items in individual rows, as shown in the top of Figure 27-1 or increases its row height to accommodate the contained information, as shown in the bottom of the figure.

Figure 27-1 A container can open to more rows or expand to show more information

▶ Paper types	

▼ Paper types	
Plain	Letter
Photo Glossy	8 X 10 Borderless

▶ Layout

▼ Layout	
Pages per sheet	2
Layout direction	Left to right
Order	Back to front

When the `expandableRows` parameter is set to `true`:

- Disclosure triangles are drawn top-justified in the row.
- Custom row height, if any, for that row is respected only while the row is disclosed. At other times, the default row height is used.

When the `expandableRows` parameter is set to `false`:

- Disclosure triangles are centered vertically in the row.
- Custom row height, if any, for that row is always respected.

When a disclosure triangle is clicked by the user, your application receives the same notifications regardless of whether `expandableRows` is set to `true` or `false`. When your application receives a notification that an expandable row is toggled to open, call the function `SetDataBrowserTableViewItemRowHeight` (page 2227) to set the row to the appropriate height.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

SetDataBrowserListViewHeaderBtnHeight

Sets the height of the rectangular area where the column title appears.

```
OSStatus SetDataBrowserListViewHeaderBtnHeight (
   ControlRef browser,
   UInt16 height
);
```

Parameters

browser

A data browser.

height

The height, in pixels, to use for the rectangular area where the column title appears. Pass `0` to turn off header button display. To turn on header button display, pass the value previously obtained from the function `GetDataBrowserListViewHeaderBtnHeight` (page 2159). The default height is currently 17 pixels.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

SetDataBrowserListViewHeaderDesc

Provides a description for a column title in list view.

```
OSStatus SetDataBrowserListViewHeaderDesc (
   ControlRef browser,
   DataBrowserTableViewColumnID column,
   DataBrowserListViewHeaderDesc *desc
);
```

Parameters

browser

A data browser.

column

The property ID for the column in list view whose title description you want to set. The `DataBrowserTableViewColumnID` data type is the same as the `DataBrowserPropertyID` data type.

desc

The list view header description structure that you have filled out with data that describes the appearance of a column title in list view.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
This functions allows you to change the behavior or appearance of a column title. Typically you call this function if your application:

- Supports switching between list and column views, and you need to restore previously saved list view title information.
- Creates a list view data browser programmatically and the columns have titles.

Availability
Available in CarbonLib 1.5 and later.
Available in Mac OS X version 10.2 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

SetDataBrowserListViewUsePlainBackground

Specifies whether list view uses a plain white background.

```
OSStatus SetDataBrowserListViewUsePlainBackground (
   ControlRef browser,
   Boolean usePlainBackground
);
```

Parameters

`browser`
A data browser.

`usePlainBackground`
A value that specifies whether to use a plain background (`true`) or not to use a plain background (`false`). A plain background is an all-white background. In Mac OS X, passing `false` currently does nothing, as Mac OS X supports only a plain white background. However, pass `true` if you want a plain white background just in case the API changes in the future. In Mac OS 9, passing `false` causes the data browser to use a shaded background.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
A list view that does not use a plain background can use colors or patterns to distinguish one column from another. For example, you could specify a color to designate a column as the sorted column.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

SetDataBrowserPropertyFlags

Sets the appearance and behavior attributes for a column in list view.

```
OSStatus SetDataBrowserPropertyFlags (
   ControlRef browser,
   DataBrowserPropertyID property,
   DataBrowserPropertyFlags flags
);
```

Parameters

browser

A data browser.

property

The property ID of the column whose appearance and behavior you want to set.

flags

The property flags to apply. A `DataBrowserPropertyFlags` value is a 32-bit value that is divided into four parts as follows:

- Bits 0–7 specify properties applied to the data browser as a whole—see "Property Flags: Universal" (page 2284)
- Bits 8–15 modify display behavior—see "Property Flags: Modifiers" (page 2285)
- Bits 16–23 are properties specific to list view—see "Property Flags: Offset and Mask for List View Properties" (page 2288) and "Property Flags: List View Column Behavior" (page 2289)
- Bits 24–31 can be defined by your application—see "Property Flags: Offset and Mask for Client-Defined Properties" (page 2290)

Return Value

A result code. See "Data Browser Result Codes" (page 2298).

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X version 10.0 and later.

Not available to 64-bit applications.

Declared In

`HIDataBrowser.h`

SetDataBrowserScrollBarInset

Sets the inset values to use for the scroll bars of a data browser.

```
OSStatus SetDataBrowserScrollBarInset (
   ControlRef browser,
   Rect *insetRect
);
```

Parameters

browser

A data browser.

insetRect

A pointer to a rectangle that specifies the inset values you want the data browser to use. The left and right fields contain the horizontal inset values for the horizontal scroll bar, and the top and bottom fields contain the vertical inset values for the vertical scroll bar.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
Your application can call the functions `GetDataBrowserScrollBarInset` (page 2162) and `SetDataBrowserScrollBarInset` if you want to place placards or controls beside the horizontal scroll bars or above the vertical ones. To do so, first call `GetDataBrowserScrollBarInset` to obtain the current settings. After modifying the current inset settings to provide space for the placard or control, call `SetDataBrowserScrollBarInset` with the new values.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

SetDataBrowserScrollPosition

Scrolls a list to the specified position.

```
OSStatus SetDataBrowserScrollPosition (
   ControlRef browser,
   UInt32 top,
   UInt32 left
);
```

Parameters

browser
A data browser.

top
The vertical scrolling position to use.

left
The horizontal scrolling position to use.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
The scrolling position `(0,0)` represents the home position, and is located at the top left of the data browser. Horizontal and vertical units are relative to the home position.

You can call this function to scroll a list to any arbitrary scrolling position. Normally, you use the function `GetDataBrowserScrollPosition` (page 2162) in conjunction with `SetDataBrowserScrollPosition` to save and restore the scrolling position of a list to the user's last scrolling position. These functions should not be used to scroll particular cells into the view. For that, call the function `RevealDataBrowserItem`.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
HIDataBrowser.h

SetDataBrowserSelectedItems

Modifies the current selection by adding items, removing items, or toggling the selection state of items.

```
OSStatus SetDataBrowserSelectedItems (
   ControlRef browser,
   ItemCount numItems,
   const DataBrowserItemID *items,
   DataBrowserSetOption operation
);
```

Parameters

browser
A data browser.

numItems
The number of item ID values stored in the array pointed to by the `items` parameter.

items
A pointer to an array of the item IDs to modify the selection with.

operation
The operation you want to perform on the current selection. You can add, assign, toggle, or remove the items specified by the `items` parameter. See "Selection State Options" (page 2293) for a list of the constants you can supply and a complete description of what each constant does.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
HIDataBrowser.h

SetDataBrowserSelectionFlags

Sets allowable selection behavior for a data browser.

```
OSStatus SetDataBrowserSelectionFlags (
   ControlRef browser,
   DataBrowserSelectionFlags selectionFlags
);
```

Parameters

browser
A data browser.

selectionFlags

Flags that specify the selection behavior you want to allow in the data browser. The flags control such things as whether discontinuous selections are allowed by the user. See "User Selection Flags" (page 2296) for detailed descriptions of these flags.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
HIDataBrowser.h

SetDataBrowserSortOrder

Sets the sorting order for a list in list view.

```
OSStatus SetDataBrowserSortOrder (
   ControlRef browser,
   DataBrowserSortOrder order
);
```

Parameters

browser

A data browser.

order

The sorting order. See "Sorting Orders" (page 2294) for a list of the constants you can supply.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
List view tracks the sorting order by column. In Mac OS X, setting the sorting order only affects the sorting order of the column currently set for sorting.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
HIDataBrowser.h

SetDataBrowserSortProperty

Designates the list view column to use for sorting.

```
OSStatus SetDataBrowserSortProperty (
   ControlRef browser,
   DataBrowserPropertyID property
);
```

Parameters

browser

A data browser.

property

The property ID of the column to use for sorting.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
If the list is not currently sorted, or if the list is currently sorted with a different column, then the list is sorted and redrawn. You can all the function `GetDataBrowserSortProperty` to obtain the property ID of the column currently used for sorting.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

SetDataBrowserTableViewColumnPosition

Changes the visual position of a column in list view.

```
OSStatus SetDataBrowserTableViewColumnPosition (
   ControlRef browser,
   DataBrowserTableViewColumnID column,
   DataBrowserTableViewColumnIndex position
);
```

Parameters

browser

A data browser.

column

The property ID for the list view column you want to move. The `DataBrowserTableViewColumnID` data type is the same as the `DataBrowserPropertyID` data type.

position

The position you want to move the column to.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
If you set your list to have the property `kDataBrowserListViewMovableColumn`, the user can rearrange columns by dragging. The function `SetDataBrowserTableViewColumnPosition` provides a way for your application to rearrange columns programmatically.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
HIDataBrowser.h

SetDataBrowserTableViewColumnWidth

Sets the default column width for all columns in a data browser.

```
OSStatus SetDataBrowserTableViewColumnWidth (
   ControlRef browser,
   UInt16 width
);
```

Parameters

browser
A data browser.

width
The column width, in pixels.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
You can override the default width for an individual list view column by calling the function SetDataBrowserTableViewNamedColumnWidth (page 2228).

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
HIDataBrowser.h

SetDataBrowserTableViewGeometry

Sets whether columns and rows can have variable widths in list view.

```
OSStatus SetDataBrowserTableViewGeometry (
   ControlRef browser,
   Boolean variableWidthColumns,
   Boolean variableHeightRows
);
```

Parameters

browser
A data browser.

variableWidthColumns
A Boolean value that specifies whether column widths can be variable (`true`) or not (`false`).

variableHeightRows
A Boolean value that specifies whether row heights can be variable (`true`) or not (`false`).

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
After you call the function `SetDataBrowserTableViewGeometry` to set up variable row heights or columns widths in list view, you can modify individual row heights or columns widths in list view by calling the appropriate function—either `SetDataBrowserTableViewItemRowHeight` (page 2227) or `SetDataBrowserTableViewNamedColumnWidth` (page 2228).

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

SetDataBrowserTableViewHiliteStyle

Sets the highlighting style to use for a list view data browser.

```
OSStatus SetDataBrowserTableViewHiliteStyle (
   ControlRef browser,
   DataBrowserTableViewHiliteStyle hiliteStyle
);
```

Parameters

browser
A list view data browser.

hiliteStyle
The highlighting style you want to use. See "Table View Highlighting Styles" (page 2294) for a description of the constants you can supply.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

SetDataBrowserTableViewItemRow

Changes the visual position for an item in a list view data browser.

```
OSStatus SetDataBrowserTableViewItemRow (
   ControlRef browser,
   DataBrowserItemID item,
   DataBrowserTableViewRowIndex row
);
```

Parameters

browser

A data browser.

item

The item ID for the item whose row you want to set.

row

The row index for the row you want to move the item to.

Return Value

A result code. See "Data Browser Result Codes" (page 2298).

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X version 10.0 and later.

Not available to 64-bit applications.

Declared In

`HIDataBrowser.h`

SetDataBrowserTableViewItemRowHeight

Sets the row height for a single row in a list view data browser.

```
OSStatus SetDataBrowserTableViewItemRowHeight (
   ControlRef browser,
   DataBrowserItemID item,
   UInt16 height
);
```

Parameters

browser

A data browser.

item

The item ID for the item whose row height you want to set.

height

The row height, in pixels.

Return Value

A result code. See "Data Browser Result Codes" (page 2298).

Discussion

Before calling this function, you must call the function `SetDataBrowserTableViewGeometry` (page 2225) to set up variable row heights.

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X version 10.0 and later.

Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

SetDataBrowserTableViewNamedColumnWidth

Sets the column width for a single column in a list view data browser.

```
OSStatus SetDataBrowserTableViewNamedColumnWidth (
   ControlRef browser,
   DataBrowserTableViewColumnID column,
   UInt16 width
);
```

Parameters

browser
A data browser.

column
The property ID for the list view column whose width you want to set. The `DataBrowserTableViewColumnID` data type is the same as the `DataBrowserPropertyID` data type.

width
The width of the column, in pixels.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
Before calling this function, you must call the function `SetDataBrowserTableViewGeometry` (page 2225) to set up variable column widths.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

SetDataBrowserTableViewRowHeight

Sets the default row height for all rows in a data browser.

```
OSStatus SetDataBrowserTableViewRowHeight (
   ControlRef browser,
   UInt16 height
);
```

Parameters

browser
A data browser.

height
The row height, in pixels.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
This function sets the default row height for all rows. You override the default row height for an individual row by calling the function `SetDataBrowserTableViewItemRowHeight` (page 2227).

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

SetDataBrowserTarget

Sets the target for a data browser.

```
OSStatus SetDataBrowserTarget (
   ControlRef browser,
   DataBrowserItemID target
);
```

Parameters

browser
A data browser.

target
The item ID to assign as the target for the browser control.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
Your application can set an item ID to use as a target if you do not want to use the default target set by the data browser. By default, the target is a container whose ID is `kDataBrowserNoItem`. For the list view, the target can be thought of as the root container. For the column view, the target is the rightmost column. When an item is dragged over a data browser but not dropped over any particular item, the target becomes the destination.

`SetDataBrowserTarget` changes the container that the data browser displays, thereby populating the data browser with items. If you use the function in column view, you must make sure your item-data callback responds to the property `kDataBrowserItemParentContainerProperty` by providing the item ID of the target's parent. This allows the function `SetDataBrowserColumnViewPath` (page 2202) to process the data properly. The target is the leaf node item whose contents you want to display. However, unlike `GetDataBrowserColumnViewPathLength` (page 2142), the function `SetDataBrowserTarget` doesn't offer a way for you to communicate the item IDs of the rest of the column containers, so `SetDataBrowserTarget` asks for them explicitly by requesting the item's parent, then the parent of the item's parent, and so on.

You can pass a noncontainer item to this function in either list or column views. If you do, you must also respond to the property `kDataBrowserItemParentContainerProperty`. The data browser requests the parent of the target so it knows which container to display the contents of in the list view.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

SetDataBrowserUserState

Restores the view-style settings in list view to a previous state set by the user.

```
OSStatus SetDataBrowserUserState (
   ControlRef browser,
   CFDictionaryRef stateInfo
);
```

Parameters

browser
A data browser.

stateInfo
A CFDictionary object that specifies the view-style settings that you want to restore. Note that although this parameter is typed as a CFData object, you must supply a CFDictionary object because that is the form of the data the system expects.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
Typically you use this function to restore the user state you previously obtained by calling the function `GetDataBrowserUserState` (page 2173).

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

SetDataBrowserViewStyle

Sets the view style of the specified data browser.

```
OSStatus SetDataBrowserViewStyle (
   ControlRef browser,
   DataBrowserViewStyle style
);
```

Parameters

browser

A data browser.

style

The view style to use. Pass the constant `kDataBrowserListView` to draw the data browser using list view or `kDataBrowserColumnView` to use column view. See "View Styles" (page 2298) for more information on these constants.

Return Value

A result code. See "Data Browser Result Codes" (page 2298).

Discussion

Although you specify a view style when you call the function `CreateDataBrowserControl` (page 2125), you can call `SetDataBrowserViewStyle` to change the style. Use `SetDataBrowserViewStyle` when you provide users the option of changing between list and column views.

After calling `SetDataBrowserViewStyle`, you need to perform the necessary tasks to configure the data browser for the view style you switched to. If you switch to list view, you need to set up list view header and column descriptions and call the function `AddDataBrowserListViewColumn` (page 2121). You might also need to call other functions such as `SetDataBrowserListViewDisclosureColumn` (page 2216) (for hierarchical lists).

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X version 10.0 and later.

Not available to 64-bit applications.

Declared In

`HIDataBrowser.h`

SortDataBrowserContainer

Sorts a hierarchical list of items.

```
OSStatus SortDataBrowserContainer (
   ControlRef browser,
   DataBrowserItemID container,
   Boolean sortChildren
);
```

Parameters

browser

A data browser.

container

An item ID or the constant `kDataBrowserNoItem`. To sort all of the items that are organized as subitems of a container item, pass the item ID for the container item. To sort all of the items displayed at the top level of the data browser, pass the constant `kDataBrowserNoItem`.

sortChildren

A value that indicates whether to sort all items in the container hierarchy. Pass `true` to sort all items in the container hierarchy and `false` to sort just the immediate children of the container.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

UpdateDataBrowserItems

Requests a redraw of one or more items in a data browser.

```
OSStatus UpdateDataBrowserItems (
   ControlRef browser,
   DataBrowserItemID container,
   ItemCount numItems,
   const DataBrowserItemID *items,
   DataBrowserPropertyID preSortProperty,
   DataBrowserPropertyID propertyID
);
```

Parameters

browser

A data browser.

container

An item ID or the constant `kDataBrowserNoItem`. Pass the item ID that uniquely identifies the container. If you are updating one or more items that are in the root container, pass `kDataBrowserNoItem`.

numItems

The number of items in the array pointed to by the `items` parameter.

items

A pointer to an array of item ID values for the items you want to update. If you pass `NULL` or if the value `kDataBrowserNoItem` is an element in this array, then all rows are updated.

preSortProperty

The property ID of the column whose sorting order is the same as the sorting order of the `items` array. A property ID is a four-character sequence that you assign to represent a column in list view. Pass `kDataBrowserItemNoProperty` if the `items` array is not sorted of if you don't know the sorting order.

propertyID

The property ID of the column that must be updated. To update all columns associated with the items in the `items` array, pass `kDataBrowserNoItem`.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
After your application makes changes to any of the data items in a data browser you must update the display by calling the function `UpdateDataBrowserItems`. Calling this function also updates any internal caches allocated by the data browser.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X version 10.0 and later.
Not available to 64-bit applications.

Declared In
`HIDataBrowser.h`

Callbacks

DataBrowserAcceptDragProcPtr

Defines a pointer to an accept-drag callback function that determines whether your application can accept a drag object in the specified location.

```
typedef Boolean (*DataBrowserAcceptDragProcPtr) (
    ControlRef browser,
    DragRef theDrag,
    DataBrowserItemID item
);
```

You would declare an accept-drag callback function named `MyDataBrowserAcceptDragCallback` like this:

```
Boolean MyDataBrowserAcceptDragCallback (
    ControlRef browser,
    DragRef theDrag,
    DataBrowserItemID item
);
```

Parameters

browser
A data browser.

theDrag
The drag reference provided by the data browser to your callback.

item
The item ID of the item the drag object is held over. If the drag object is over the data browser but not over any specific item, the `item` parameter contains the item ID that represents one of the following:

- In list view, the target. (See `SetDataBrowserTarget` (page 2229).)
- In column view, the item ID of the column the drag object is over

Return Value
Your callback returns `true` if it is capable of accepting the drag object in the location designated by the item parameter. Otherwise, your callback returns `false`.

Discussion
The accept-drag callback is called by the data browser when your application needs to determine if it can accept a drag object in a particular location.

To provide a pointer to your accept-drag callback, you create a universal procedure pointer (UPP) of type `DataBrowserAcceptDragUPP`, using the function `NewDataBrowserAcceptDragUPP` (page 2187). You can do so with code similar to the following:

```
DataBrowserAcceptDragUPP MyDataBrowserAcceptDragUPP;
MyDataBrowserAcceptDragUPP = NewDataBrowserAcceptDragUPP
                        (&MyDataBrowserAcceptDragCallback);
```

You can then assign `MyDataBrowserAcceptDragUPP` to the `acceptDragCallback` field of the structure `DataBrowserCallbacks` (page 2264). You install your data browser callbacks using the function `SetDataBrowserCallbacks` (page 2200).

When you no longer need the UPP, remove it using the `DisposeDataBrowserAcceptDragUPP` (page 2129) function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HIDataBrowser.h`

DataBrowserAddDragItemProcPtr

Defines a pointer to an add-drag-item callback function that adds an item to a drag reference.

```
typedef Boolean (*DataBrowserAddDragItemProcPtr) (
    ControlRef browser,
    DragRef theDrag,
    DataBrowserItemID item,
    ItemReference *itemRef
);
```

You would declare an add-drag-item callback function named `MyDataBrowserAddDragItemCallback` like this:

```
Boolean MyDataBrowserAddDragItemCallback (
    ControlRef browser,
    DragRef theDrag,
    DataBrowserItemID item,
    ItemRef *itemRef
);
```

Parameters
browser
A data browser.

theDrag
The drag reference provided by the data browser to your callback.

item
The item ID of the item to add to the drag object.

itemRef
A pointer to a drag item reference variable. Your callback must set this to the `DragItemRef` value that it passes to the Drag Manager function `AddDragItemFlavor`.

Return Value
Your callback returns `true` to indicate the item should be or is part of the drag object. Your callback returns `false` if the item isn't part of the drag object.

Discussion
The add-drag-item callback is called by the data browser when a drag operation needs to be started. The data browser iterates through the selected items, invoking your callback for each item. Your callback is called after the drag reference is created by the data browser but before the function `TrackDrag` is called by the system.

Your callback adds an item to the drag reference calling the function `AddDragItemFlavor`. When you call `AddDragItemFlavor`, you must provide a unique drag item reference (`DragItemRef`) for each data browser item that you add to the drag. You must also provide the data type of the added item (drag flavor type) and set the appropriate drag flavor flags.

The data browser handles imaging and adds transparency for you. As a result, you do not need to create or add your own transparency information to the drag reference.

To provide a pointer to your add-drag-item callback, you create a universal procedure pointer (UPP) of type `DataBrowserAddDragItemUPP`, using the function `NewDataBrowserAddDragItemUPP` (page 2188). You can do so with code similar to the following:

```
DataBrowserAddDragItemUPP MyDataBrowserAddDragItemUPP;
MyDataBrowserAddDragItemUPP = NewDataBrowserAddDragItemUPP
                        (&MyDataBrowserAddDragItemCallback);
```

You can then assign `MyDataBrowserAddDragItemUPP` to the `addDragItemCallback` field of the structure `DataBrowserCallbacks` (page 2264). You install your data browser callbacks using the function `SetDataBrowserCallbacks` (page 2200).

When you no longer need the UPP, remove it using the `DisposeDataBrowserAddDragItemUPP` (page 2130) function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HIDataBrowser.h`

DataBrowserDrawItemProcPtr

Defines a pointer to a draw-item callback function that draws a custom item.

```
typedef void (*DataBrowserDrawItemProcPtr) (
    ControlRef browser,
    DataBrowserItemID item,
    DataBrowserPropertyID property,
    DataBrowserItemState itemState,
    const Rect *theRect,
    SInt16 gdDepth,
    Boolean colorDevice
);
```

You would declare a draw-item callback function named `MyDataBrowserDrawItemCallback` like this:

```
void MyDataBrowserDrawItemCallback (
    ControlRef browser,
    DataBrowserItemID item,
    DataBrowserPropertyID property,
    DataBrowserItemState itemState,
    const Rect *theRect,
    SInt16 gdDepth,
    Boolean colorDevice
);
```

Parameters

browser
: A data browser.

item
: The item ID for the item to draw.

property
: The property ID for the item. In list view, this is the four-character sequence that you previously assigned to the column. In column view, this is the property `kDataBrowserItemSelfIdentityProperty`.

itemState
: The state to use when drawing the item. See "Item States" (page 2280) for a description of the constants that can be provided to your callback.

theRect
: A pointer to the bounding rectangle (in local coordinates, relative to the port) that specifies where to draw the item. This rectangle is the content rectangle, not the enclosing rectangle.

gdDepth
: The bit depth of the current QuickDraw graphics port. The data browser sets the current QuickDraw port to the port that you draw into. This may not always be the port of the data browser's own window.

colorDevice
: A value that specifies whether the current QuickDraw port is a color device (`true`) or is not (`false`).

Discussion

The draw-item callback is called by the data browser when an item whose display type is `kDataBrowserCustomType` needs to be drawn. Your application draws the item so it reflects the state specified by the `itemState` parameter.

To provide a pointer to your draw-item callback, you create a universal procedure pointer (UPP) of type `DataBrowserDrawItemUPP`, using the function `NewDataBrowserDrawItemUPP` (page 2188). You can do so with code similar to the following:

```
DataBrowserDrawItemUPP MyDataBrowserDrawItemUPP;
```

```
MyDataBrowserDrawItemUPP = NewDataBrowserDrawItemUPP
                        (&MyDataBrowserDrawItemCallback);
```

You can then assign `MyDataBrowserDrawItemUPP` to the `drawItemCallback` field of the structure `DataBrowserCustomCallbacks` (page 2265). You install your data browser custom callbacks using the function `SetDataBrowserCustomCallbacks` (page 2203).

When you no longer need the UPP, remove it using the `DisposeDataBrowserDrawItemUPP` (page 2130) function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HIDataBrowser.h`

DataBrowserEditItemProcPtr

Defines a pointer to an edit-item callback function that determines if the data browser should start an edit session for a custom item.

Not Recommended

```
typedef Boolean (*DataBrowserEditItemProcPtr) (
    ControlRef browser,
    DataBrowserItemID item,
    DataBrowserPropertyID property,
    CFStringRef theString,
    Rect *maxEditTextRect,
    Boolean *shrinkToFit
);
```

You would declare an edit-item callback function named `MyDataBrowserEditItemCallback` like this:

```
Boolean MyDataBrowserEditItemCallback (
    ControlRef browser,
    DataBrowserItemID item,
    DataBrowserPropertyID property,
    CFStringRef theString,
    Rect *maxEditTextRect,
    Boolean *shrinkToFit
);
```

Parameters

browser
A data browser.

item
The item ID number for the item.

property
The property ID for the item. In list view, this is the four-character sequence that you previously assigned to the column. In column view, this is the property `kDataBrowserItemSelfIdentityProperty`.

theString

The string to be edited. See Special Considerations for more information.

maxEditTextRect

On input, a pointer to a rectangle structure. On return, set the rectangle to the largest size the edit field can grow to. If the text grows beyond the size of the edit field, the text scrolls as the user types. This parameter is used only if the parameter `shrinkToFit` is `true`. Otherwise, the current size of the text editing field is used.

shrinkToFit

On input, a pointer to a Boolean variable. On return, set this variable to `true` if you want the data browser to expand or shrink the text editing field to match the width of the text in the edit field. Note that this parameter is currently ignored; `shrinkToFit` is always `true` by default.

Return Value

A value that indicates whether or not you want to start an edit operation for the given property of the item. If your application performs the editing operation, your callback returns `true`. Otherwise, your callback returns `false`.

Discussion

The edit-item callback is called by the data browser for an item whose property is `kDataBrowserCustomType`. Your callback must determine whether an editing session should be started and, if so, set the string to be edited, set the size of the edit rectangle, and specify whether the text editing field can adjust to match the width of the text in the edit field.

To provide a pointer to your edit-item callback, you create a universal procedure pointer (UPP) of type `DataBrowserEditItemUPP`, using the function `NewDataBrowserEditItemUPP` (page 2188). You can do so with code similar to the following:

```
DataBrowserEditItemUPP MyDataBrowserEditItemUPP;
MyDataBrowserEditItemUPP = NewDataBrowserEditItemUPP
                        (&MyDataBrowserEditItemCallback);
```

You can then assign `MyDataBrowserEditItemUPP` to the `editItemCallback` field of the structure `DataBrowserCustomCallbacks` (page 2265). You install your data browser custom callbacks using the function `SetDataBrowserCustomCallbacks` (page 2203).

When you no longer need the UPP, remove it using the `DisposeDataBrowserEditItemUPP` (page 2131) function.

Special Considerations

This callback does not work properly. The `theString` parameter is an immutable sting, which means it is not possible for the callback to set the string to the text that is to be edited.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`HIDataBrowser.h`

DataBrowserGetContextualMenuProcPtr

Defines a pointer to a get-contextual-menu callback function that obtains a menu and information about the menu.

```
typedef void (*DataBrowserGetContextualMenuProcPtr) (
    ControlRef browser,
    MenuRef *menu,
    UInt32 *helpType,
    CFStringRef *helpItemString,
    AEDesc *selection
);
```

You would declare a get-contextual-menu callback function named `MyDataBrowserGetContextualMenuCallback` like this:

```
void MyDataBrowserGetContextualMenuCallback (
    ControlRef browser,
    MenuRef *menu,
    UInt32 *helpType,
    CFStringRef *helpItemString,
    AEDesc *selection
);
```

Parameters

browser

A data browser.

menu

On input, a pointer to a menu. Your callback must set the pointer to the menu that you want to have displayed for the given item. Your application is responsible for disposing of the menu; you typically perform this task in the callback `DataBrowserSelectContextualMenuProcPtr` (page 2257).

helpType

On input, a pointer to an unsigned 32-bit integer. On return, this value specifies the type of help available for this item. This value is then passed by the data browser to the Menu Manager function `ContextualMenuSelect`. You can provide one of the following values:

- `kCMHelpItemNoHelp` if your application does not support help. The Menu Manager inserts an appropriate string into the menu and then disables the associated help item.
- `kCMHelpItemAppleGuide` if your application supports Apple Guide help. The Menu Manager inserts the name of the main Apple Guide file into the menu and enables the associated help item. You can pass this in Mac OS 9. Apple Guide is not supported in Mac OS X. In Mac OS X, this value is ignored; a generic, but inactive, help item is displayed.
- `kCMHelpItemOtherHelp` if your application supports some other form of help. In this case, your application must pass a valid string in the `helpItemString` parameter. The Menu Manager inserts the string in the menu and enables the associated help item.

See *Menu Manager Reference* for more information about these constants.

helpItemString

On input, a `CFStringRef` variable. On return, a CFString object that contains the name of the item to display in the contextual menu. This is the first item that appears in the contextual menu. If you pass `NULL`, the default string (“Help”) is displayed. Data Browser does not retain the string; it releases it.

selection

On input, a pointer to an empty `AEDesc` data structure. On return, the structure contains the data supplied by your callback. The data browser passes this structure to the function `ContextualMenuSelect`.

Discussion
The get-contextual-menu callback is called by the data browser when the user Control-clicks in the data browser. You application provides a menu and information about help that is (or is not) available for the data browser. You can determine what to provide for the content of the menu and what information to put in the `AEDesc` structure by calling the function `GetDataBrowserItems` with the `state` parameter set to `kDataBrowserItemIsSelected`. This tells you what items are selected, which you can then use to choose the appropriate information to supply.

To provide a pointer to your get-contextual-menu callback, you create a universal procedure pointer (UPP) of type `DataBrowserGetContextualMenuUPP`, using the function `NewDataBrowserGetContextualMenuUPP` (page 2189). You can do so with code similar to the following:

```
DataBrowserGetContextualMenuUPP MyDataBrowserGetContextualMenuUPP;
MyDataBrowserGetContextualMenuUPP = NewDataBrowserGetContextualMenuUPP
                        (&MyDataBrowserGetContextualMenuCallback);
```

You can then assign `MyDataBrowserGetContextualMenuUPP` to the `getContextualMenuCallback` field of the structure `DataBrowserCallbacks` (page 2264). You install your data browser callbacks using the function `SetDataBrowserCallbacks` (page 2200).

When you no longer need the UPP, remove it using the `DisposeDataBrowserGetContextualMenuUPP` (page 2131) function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HIDataBrowser.h`

DataBrowserHitTestProcPtr

Defines a pointer to a hit-test callback function that determines if the pointer is over content that can be selected or dragged.

```
typedef Boolean (*DataBrowserHitTestProcPtr) (
    ControlRef browser,
    DataBrowserItemID itemID,
    DataBrowserPropertyID property,
    const Rect *theRect,
    const Rect *mouseRect
);
```

You would declare a hit-test callback function named `MyDataBrowserHitTestCallback` like this:

```
Boolean MyDataBrowserHitTestCallback (
    ControlRef browser,
    DataBrowserItemID itemID,
    DataBrowserPropertyID property,
    const Rect *theRect,
    const Rect *mouseRect
);
```

Parameters

browser

A data browser.

itemID

The item ID number for the item over which the pointer is located.

property

The property ID for the column in which the pointer is located. In list view, this is the four-character sequence that you previously assigned to the column. In column view, this is the property `kDataBrowserItemSelfIdentityProperty`.

theRect

A pointer to the bounding rectangle, in local coordinates, of the item.

mouseRect

A pointer to a rectangle structure that contains the local coordinates of the selection. If the top-left and bottom-right coordinates of this rectangle are identical, then a single point is being tested. If they differ, then the data browser is testing to see whether your custom item is inside of the bounding rectangle of a selection.

Return Value

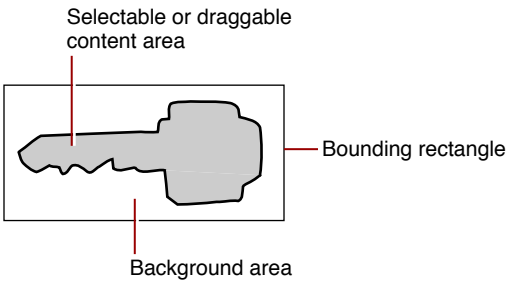
Your application returns a `true` value for either of the following conditions:

- The pointer is located over the part of the item that can be selected or dragged.
- The rectangle provided in the parameter `mouseRect` intersects with the content area of the item that can be selected or dragged.

Discussion

The hit-test callback is called by the data browser when the user hovers the pointer over, clicks the mouse within, or drags within an item whose display type is `kDataBrowserCustomType`. Your callback can use the functions `SetRect` or `SectRgn` to determine if the selectable content of the custom item is part of the selection. Figure 27-2 illustrates a situation for which the selectable or draggable content area differs from the background area in which the item is displayed.

Figure 27-2 Differentiation between the selectable content and background



To provide a pointer to your hit-test callback, you create a universal procedure pointer (UPP) of type `DataBrowserHitTestItemUPP`, using the function `NewDataBrowserHitTestUPP` (page 2189). You can do so with code similar to the following:

```
DataBrowserHitTestUPP MyDataBrowserHitTestUPP;
MyDataBrowserHitTestUPP = NewDataBrowserHitTestUPP
                        (&MyDataBrowserHitTestCallback);
```

You can then assign `MyDataBrowserHitTestUPP` to the `hitTestCallback` field of the structure `DataBrowserCustomCallbacks` (page 2265). You install your data browser custom callbacks using the function `SetDataBrowserCustomCallbacks` (page 2203).

When you no longer need the UPP, remove it using the `DisposeDataBrowserHitTestUPP` (page 2131) function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HIDataBrowser.h`

DataBrowserItemAcceptDragProcPtr

Defines a pointer to an item-accept-drag callback function that determines if a custom item can accept a drag object.

```
typedef DataBrowserDragFlags (*DataBrowserItemAcceptDragProcPtr)  (
    ControlRef browser,
    DataBrowserItemID itemID,
    DataBrowserPropertyID property,
    const Rect *theRect,
    DragRef theDrag
);
```

You would declare an item-accept-drag callback function named `MyDataBrowserItemAcceptDragCallback` like this:

```
DataBrowserDragFlags MyDataBrowserItemAcceptDragCallback (
    ControlRef browser,
    DataBrowserItemID itemID,
    DataBrowserPropertyID property,
    const Rect *theRect,
    DragRef theDrag
);
```

Parameters

browser
A data browser.

itemID
The item ID number for the item the drag object is held over. If the drag object is over the data browser but not over any specific item, the `item` parameter contains the item ID that represents one of the following:

- The target in list view. (See `SetDataBrowserTarget` (page 2229).)
- The column the drag object is over in column view

property
The property ID of the column the dragged object is over. In list view, this is the four-character sequence that you previously assigned to the column. In column view, this is the property `kDataBrowserItemSelfIdentityProperty`.

theRect
A pointer to the bounding rectangle of the item, in local coordinates relative to the current port.

theDrag
The drag reference provided by the data browser to your callback.

Return Value
If your callback determines the drag object can be accepted, return a nonzero value that has the bit `kDataBrowserItemIsDragTarget` set. Otherwise return `kDataBrowserItemNoState`. The return value is then passed to your item-receive-drag callback in the `dragFlags` parameter.

Discussion
The item-accept-drag callback is called by the data browser for an item whose display type is `kDataBrowserCustomType` when a drag object is moved over the item. Your application determines whether or not the associated item can accept the drag object. If the item cannot accept the drag object, return `0`. Otherwise, if the item is an acceptable drop location for the drag object, return a nonzero value.

If the drag object was acceptable and the drop occurs over that same item ID and property ID pair, the `DataBrowserDragFlags` values you returned from your item-accept-drag callback are passed in the `dragFlags` parameter to your item-receive-drag callback. This allows you to generate state information during drag tracking that can be communicated to you at drop time.

Do not call the function `SetOrigin` in this or any of the other drag processing callbacks.

To provide a pointer to your item-accept-drag callback, you create a universal procedure pointer (UPP) of type `DataBrowserItemAcceptDragUPP`, using the function `NewDataBrowserItemAcceptDragUPP` (page 2190). You can do so with code similar to the following:

```
DataBrowserItemAcceptDragUPP MyDataBrowserItemAcceptDragUPP;
MyDataBrowserItemAcceptDragUPP = NewDataBrowserItemAcceptDragUPP
                        (&MyDataBrowserItemAcceptDragCallback);
```

You can then assign `MyDataBrowserItemAcceptDragUPP` to the `itemAcceptDragCallback` field of the structure `DataBrowserCustomCallbacks` (page 2265). You install your data browser custom callbacks using the function `SetDataBrowserCustomCallbacks` (page 2203).

When you no longer need the UPP, remove it using the `DisposeDataBrowserItemAcceptDragUPP` (page 2132) function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HIDataBrowser.h`

DataBrowserItemCompareProcPtr

Defines a pointer to an item-comparison callback function that orders the values displayed in a column.

```
typedef Boolean (*DataBrowserItemCompareProcPtr) (
    ControlRef browser,
    DataBrowserItemID itemOne,
    DataBrowserItemID itemTwo,
    DataBrowserPropertyID sortProperty
);
```

You would declare an item-comparison callback function named `MyDataBrowserItemCompareCallback` like this:

```
Boolean MyDataBrowserItemCompareCallback (
    ControlRef browser,
    DataBrowserItemID itemOne,
    DataBrowserItemID itemTwo,
    DataBrowserPropertyID sortProperty
);
```

Parameters

browser
A data browser.

itemOne
The item ID of the first item to use in the comparison.

itemTwo
The item ID of the second item to use in the comparison.

sortProperty
The property ID for the column to sort. In list view, this is the four-character sequence that you previously assigned to the column. In column view, this is the property `kDataBrowserItemSelfIdentityProperty`.

Return Value
Your callback returns `true` if the value of the data referenced by `itemOne` is less than the value of the data referenced by `itemTwo`. It returns `false` if the value of the data referenced by `itemOne` is greater than or equal to the value of the data referenced by `itemTwo`.

Discussion
The item-comparison callback is called by the data browser when it needs to order the values displayed in a column. Your callback determines the display type of the data, then carries out the appropriate comparison for that data.

If you want your callback to use secondary and tertiary sorting, your application must keep track of previous sorting operations. Then you must make sure that each time a user clicks a column, the column is sorted, but the associated sorting orders for secondary and tertiary items in the column are preserved.

To provide a pointer to your item-comparison callback, you create a universal procedure pointer (UPP) of type `DataBrowserItemCompareUPP`, using the function `NewDataBrowserItemCompareUPP` (page 2190). You can do so with code similar to the following:

```
DataBrowserItemCompareUPP MyDataBrowserItemCompareUPP;
MyDataBrowserItemCompareUPP = NewDataBrowserItemCompareUPP
                            (&MyDataBrowserItemCompareCallback);
```

You can then assign `MyDataBrowserItemCompareUPP` to the `itemCompareCallback` field of the structure `DataBrowserCallbacks` (page 2264). You install your data browser callbacks using the function `SetDataBrowserCallbacks` (page 2200).

When you no longer need the UPP, remove it using the `DisposeDataBrowserItemCompareUPP` (page 2132) function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HIDataBrowser.h`

DataBrowserItemDataProcPtr

Defines a pointer to an item-data callback function that gets and sets properties for individual items in a data browser.

```
typedef OSStatus (*DataBrowserItemDataProcPtr) (
    ControlRef browser,
    DataBrowserItemID item,
    DataBrowserPropertyID property,
    DataBrowserItemDataRef itemData,
    Boolean setValue
);
```

You would declare an item-data callback function named `MyDataBrowserItemDataCallback` like this:

```
OSStatus MyDataBrowserItemDataCallback (
    ControlRef browser,
    DataBrowserItemID item,
    DataBrowserPropertyID property,
    DataBrowserItemDataRef itemData,
    Boolean setValue
);
```

Parameters

browser
The data browser.

item
The item ID of the item whose data is set or obtained.

property
A property ID. This value can be any of the following:

- A four-character sequence that you assign to represent a column in list view.
- Any of the API-defined properties, such as `kDataBrowserItemSelfIdentityProperty` for a column in column view or `kDataBrowserItemIsContainerProperty` for an item in list or column view that has children. See "Properties" (page 2282) for a complete list and more information on the API-defined properties.

itemData
The data buffer that either contains the data to set or receives the data to obtain.

setValue

A value that indicates whether data is to be obtained or set. This value is `false` if your application needs to set the value of the item by calling one of the set functions described in the section "Getting and Setting Item Data" (page 2115). This value is `true` if the value of the item has changed. In this case, you should call the appropriate get function, passing the item data reference provided to you in the `itemData` parameter.

Return Value
A result code. See "Data Browser Result Codes" (page 2298).

Discussion
The item-data callback communicates data between the data browser and your application. When the data browser needs to display a value for an item, it invokes your callback to request the data. If the user changes the value, the data browser invokes your callback with a new copy of the data that you can use to replace your application's internal copy. Your application must supply an item-data callback; otherwise, your data browser will not contain any data.

Your callback determines the kind of data is associated with an item and whether data needs to be obtained or set. Then, your callback takes the appropriate action by calling one of the functions listed in "Getting and Setting Item Data" (page 2115).

To provide a pointer to your item-data callback, you create a universal procedure pointer (UPP) of type `DataBrowserItemDataUPP`, using the function `NewDataBrowserItemDataUPP` (page 2191). You can do so with code similar to the following:

```
DataBrowserItemDataUPP MyDataBrowserItemDataUPP;
MyDataBrowserItemDataUPP = NewDataBrowserItemDataUPP
                           (&MyDataBrowserItemDataCallback);
```

You can then assign `MyDataBrowserItemDataUPP` to the `itemDataCallback` field of the structure `DataBrowserCallbacks` (page 2264). You install your data browser callbacks using the function `SetDataBrowserCallbacks` (page 2200).

When you no longer need the UPP, remove it using the `DisposeDataBrowserItemDataUPP` (page 2133) function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HIDataBrowser.h`

DataBrowserItemDragRgnProcPtr

Defines a pointer to an item-drag-region callback function that determines which part of the item rectangle to use when creating a transparent image for a dragged custom item.

```
typedef void (*DataBrowserItemDragRgnProcPtr) (
    ControlRef browser,
    DataBrowserItemID itemID,
    DataBrowserPropertyID property,
    const Rect *theRect,
    RgnHandle dragRgn
);
```

You would declare an item-drag-region callback function named `MyDataBrowserItemDragRgnCallback` like this:

```
void MyDataBrowserItemDragRgnCallback (
    ControlRef browser,
    DataBrowserItemID itemID,
    DataBrowserPropertyID property,
    const Rect *theRect,
    RgnHandle dragRgn
);
```

Parameters

browser
: A data browser.

itemID
: The item ID number for the row for which the drag image is generated.

property
: The property ID for the column for which the drag image is generated. In list view, this is the four-character sequence that you previously assigned to the column. In column view, this is the property `kDataBrowserItemSelfIdentityProperty`.

theRect
: A pointer to the bounding rectangle of the item, in local coordinates.

dragRgn
: On return, the drag region set to the portion of the rectangle to use for the transparent drag image. Typically this is the boundary of the content area inside your custom item. This region is used as a mask when passed to your custom draw-item callback.

Discussion

The item-drag-region callback is called by the data browser for an item whose display type is `kDataBrowserCustomType` when a drag is about to begin. Your application determines which part of the item rectangle to use when creating the transparent image that appears during a drag operation. The data browser uses this area as a clipping region when it invokes your draw-item callback.

Do not call the function `SetOrigin` in this or any of the other drag processing callbacks.

To provide a pointer to your item-drag-region callback, you create a universal procedure pointer (UPP) of type `DataBrowserItemDragRgnUPP`, using the function `NewDataBrowserItemDragRgnUPP` (page 2191). You can do so with code similar to the following:

```
DataBrowserItemDragRgnUPP MyDataBrowserItemDragRgnUPP;
MyDataBrowserItemDragRgnUPP = NewDataBrowserItemDragRgnUPP
                        (&MyDataBrowserItemDragRgnCallback);
```

You can then assign `MyDataBrowserItemDragRgnUPP` to the `itemDragRgnCallback` field of the structure `DataBrowserCustomCallbacks` (page 2265). You install your data browser custom callbacks using the function `SetDataBrowserCustomCallbacks` (page 2203).

When you no longer need the UPP, remove it using the `DisposeDataBrowserItemDragRgnUPP` (page 2133) function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HIDataBrowser.h`

DataBrowserItemHelpContentProcPtr

Defines a pointer to an item-help-content callback function that provides help tag content for an item.

```
typedef void (*DataBrowserItemHelpContentProcPtr) (
    ControlRef browser,
    DataBrowserItemID item,
    DataBrowserPropertyID property,
    HMContentRequest inRequest,
    HMContentProvidedType *outContentProvided,
    HMHelpContentPtr ioHelpContent
);
```

You would declare an item-help-content callback function named `MyDataBrowserItemHelpContentCallback` like this:

```
void DataBrowserItemHelpContentCallback (
    ControlRef browser,
    DataBrowserItemID item,
    DataBrowserPropertyID property,
    HMContentRequest inRequest,
    HMContentProvidedType *outContentProvided,
    HMHelpContentPtr ioHelpContent
);
```

Parameters

browser
A data browser.

item
The item ID of the item to provide help for.

property
The property ID of the column to provide help for. In list view, this is the four-character sequence that you previously assigned to the column. In column view, this is the property `kDataBrowserItemSelfIdentityProperty`.

inRequest

On input, a value that indicates the nature of the help tag content request. Your callback is passed one of the following constants:

- `kHMSupplyContent` indicates your callback needs to supply help content.
- `kHMDisposeContent` indicates your callback must dispose of help content.

outContentProvided

On input, a help-content-type variable. On return, the variable is set to one of the following constants that indicate whether your item-help callback was able to fulfill the request specified in the `inRequest` parameter:

- `kHMContentProvided` indicates help content is provided in the `ioHelpContent` parameter.
- `kHMContentNotProvided` indicates help content is not provided. When your callback returns this constant, the Carbon Help Manager consults other help content providers in the hierarchy until the request for help tag content is fulfilled, the top of the hierarchy is reached, or a help tag callback notifies the Carbon Help Manager to stop propagating the request.
- `kHMContentNotProvidedDontPropagate` indicates help content is not provided. When your callback returns this constant, the Carbon Help Manager assumes that there is no help content for the data browser item and does not propagate the request.

See *Carbon Help Manager Reference* for more information on these constants.

ioHelpContent

A help tag structure that describes the help tag for the item. On input, the data browser passes a value in the version field. If the value of the `inRequest` parameter is `kHMSupplyContent`, your callback must fill in the remaining fields of the structure or specify that it is unable to fulfill the help tag content request.

Discussion

The item-help-content callback is called by the data browser when the user hovers the pointer over an item in a data browser for which you've registered a help tag callback. Your application fills in the help tag structure pointed to by the `ioHelpContent` parameter.

When the help tag for the item is no longer needed, the data browser invokes your callback with a `kHMDisposeContent` request. When you receive this request, free any memory allocated for the help tag content and perform any other cleanup tasks that are necessary.

To provide a pointer to your item-help-content callback, you create a universal procedure pointer (UPP) of type `DataBrowserItemHelpContentUPP`, using the function `NewDataBrowserItemHelpContentUPP` (page 2192). You can do so with code similar to the following:

```
DataBrowserItemHelpContentUPP MyDataBrowserItemHelpContentUPP;
MyDataBrowserItemHelpContentUPP = NewDataBrowserItemHelpContentUPP
                        (&MyDataBrowserItemHelpContentCallback);
```

You can then assign `MyDataBrowserItemHelpContentUPP` to the `itemHelpContentCallback` field of the structure `DataBrowserCallbacks` (page 2264). You install your data browser callbacks using the function `SetDataBrowserCallbacks` (page 2200).

When you no longer need the UPP, remove it using the `DisposeDataBrowserItemHelpContentUPP` (page 2133) function.

Availability

Available in Mac OS X v10.0 and later.

Declared In
HIDataBrowser.h

DataBrowserItemNotificationProcPtr

Defines a pointer to an item-notification callback function that notifies your application of changes in the data browser.

```
typedef void (*DataBrowserItemNotificationProcPtr) (
    ControlRef browser,
    DataBrowserItemID item,
    DataBrowserItemNotification message
);
```

You would declare an item-notification callback function named `MyDataBrowserItemNotificationCallback` like this:

```
void MyDataBrowserItemNotificationCallback (
    ControlRef browser,
    DataBrowserItemID item,
    DataBrowserItemNotification message
);
```

Parameters

browser
A data browser.

item
The item ID of the item that generated the notification.

message
A notification. See "Item Notifications" (page 2278) for a description of the values that can be provided to your callback.

Discussion

The item-notification callback is called by the data browser to notify your application of actions taken by the user (such as editing started, container opened, container closed) or any other condition that your application might choose to respond to. Your item-notification callback can evaluate the notification and take appropriate action.

To provide a pointer to your item-notification callback, you create a universal procedure pointer (UPP) of type `DataBrowserItemNotificationUPP`, using the function `NewDataBrowserItemNotificationUPP` (page 2192). You can do so with code similar to the following:

```
DataBrowserItemNotificationUPP MyDataBrowserItemNotificationUPP;
MyDataBrowserItemNotificationUPP = NewDataBrowserItemNotificationUPP
                            (&MyDataBrowserItemNotificationCallback);
```

You can then assign `MyDataBrowserItemNotificationUPP` to the `itemNotificationCallback` field of the structure `DataBrowserCallbacks` (page 2264). You install your data browser callbacks using the function `SetDataBrowserCallbacks` (page 2200).

When you no longer need the UPP, remove it using the `DisposeDataBrowserItemNotificationUPP` (page 2134) function.

Special Considerations

In CarbonLib the item-notification callback is invoked with the three parameters shown in `DataBrowserItemNotificationProcPtr` (page 2250). The four-parameter version—`DataBrowserItemNotificationWithItemProcPtr` (page 2251)—does not provide valid data in the fourth parameter. Any attempt to use the invalid data in a CarbonLib application may result in a crash.

In Mac OS X, the item-notification callback is invoked with the four parameters shown in `DataBrowserItemNotificationWithItemProcPtr` (page 2251). In Mac OS X you have the option of using the `DataBrowserItemNotificationProcPtr` (page 2250) or the `DataBrowserItemNotificationWithItemProcPtr` (page 2251). Which one you choose depends on whether your application needs to use the data passed to your callback in the fourth parameter (the `itemData` parameter).

Availability

Available in Mac OS X v10.1 and later.

Declared In

`HIDataBrowser.h`

DataBrowserItemNotificationWithItemProcPtr

Defines a pointer to an item-notification-with-data callback function that notifies your application of changes in the data browser and supplies any data associated with the changes.

```
typedef void (*DataBrowserItemNotificationWithItemProcPtr) (
    ControlRef browser,
    DataBrowserItemID item,
    DataBrowserItemNotification message,
    DataBrowserItemDataRef itemData
);
```

You would declare an item-notification-with-data callback function named `MyDataBrowserItemNotificationWithItemCallback` like this:

```
void MyDataBrowserItemNotificationWithItemCallback (
    ControlRef browser,
    DataBrowserItemID item,
    DataBrowserItemNotification message,
    DataBrowserItemDataRef itemData
);
```

Parameters

browser

A data browser.

item

The item ID of the item that generated the notification.

message

A notification. See "Item Notifications" (page 2278) for a description of the values that can be provided to your callback.

itemData

The data associated with the changes in the data browser that caused this callback to be invoked. You pass this data to functions that get and set item data (such as `SetDataBrowserItemDataIcon` (page 2209), `SetDataBrowserItemDataText` (page 2214), `GetDataBrowserItemDataIcon` (page 2148), and `GetDataBrowserItemDataText` (page 2154)). See "Getting and Setting Item Data" (page 2115) for a list of all the function that can use item data.

Discussion

The item-notification-with-data callback is called by the data browser to notify your application of actions taken by the user (such as editing started, container opened, container closed) or any other condition that your application might choose to respond to. Your item-notification-with-data callback can evaluate the notification and take appropriate action. Unlike the callback `DataBrowserItemNotificationProcPtr` (page 2250), the callback `DataBrowserItemNotificationWithItemProcPtr` (page 2251) provides a fourth parameter that contains any data associated with the item from which the notification is generated. This data is available only to Mac OS X applications. See Special Considerations for more details.

To provide a pointer to your item-notification-with-data callback, you create a universal procedure pointer (UPP) of type `DataBrowserItemNotificationWithItemUPP`, using the function `NewDataBrowserItemNotificationWithItemUPP` (page 2193). You can do so with code similar to the following:

```
DataBrowserItemNotificationWithItemUPP
                    MyDataBrowserItemNotificationWithItemUPP;
MyDataBrowserItemNotificationWithItemUPP =
                NewDataBrowserItemNotificationWithItemUPP
                (&MyDataBrowserItemNotificationWithItemCallback);
```

You can then assign `MyDataBrowserItemNotificationWithItemUPP` to the `itemNotificationCallback` field of the structure `DataBrowserCallbacks` (page 2264). You install your data browser callbacks using the function `SetDataBrowserCallbacks` (page 2200).

When you no longer need the UPP, remove it using the `DisposeDataBrowserItemNotificationWithItemUPP` (page 2134) function.

Special Considerations

In CarbonLib the item-notification callback is invoked with the three parameters shown in `DataBrowserItemNotificationProcPtr` (page 2250). The four-parameter version (`DataBrowserItemNotificationWithItemProcPtr` (page 2251)) does not provide valid data in the fourth parameter. Any attempt to use the invalid data in a CarbonLib application may result in a crash.

In Mac OS X, the item-notification callback is invoked with the four parameters shown in `DataBrowserItemNotificationWithItemProcPtr` (page 2251). In Mac OS X you have the option of using the `DataBrowserItemNotificationProcPtr` (page 2250) or the `DataBrowserItemNotificationWithItemProcPtr` (page 2251). Which one you choose depends on whether your application needs to use the data passed to your callback in the fourth parameter (`itemData`).

Availability

Available in Mac OS X v10.1 and later.

Declared In

`HIDataBrowser.h`

DataBrowserItemProcPtr

Defines a pointer to an item-iterator callback function that is applied by the function `ForEachDataBrowserItem` to each item in a data browser.

```
typedef void (*DataBrowserItemProcPtr) (
    DataBrowserItemID item,
    DataBrowserItemState state,
    void *clientData
);
```

You would declare an item-iterator callback function named `MyDataBrowserItemCallback` like this:

```
void MyDataBrowserItemCallback (
    DataBrowserItemID item,
    DataBrowserItemState state,
    void *clientData
);
```

Parameters

item

The item ID of the item to operate on.

state

The state of the item. See "Item States" (page 2280) for a description of possible states. If the function `ForEachDataBrowserItem` (page 2138) is set up to operate on items of a specified state, then the state passed to your callback includes the state specified as a parameter to `ForEachDataBrowserItem`.

clientData

A pointer to a buffer, local variable, or other storage location created and disposed of by your application, and supplied to the data browser with a previous call to `ForEachDataBrowserItem`.

Discussion

An item-iterator callback is supplied as a parameter to the function `ForEachDataBrowserItem`. The function applies your callback to each data item that meets the criteria specified by the function `ForEachDataBrowserItem`.

To provide a pointer to your item-iterator callback, you create a universal procedure pointer (UPP) of type `DataBrowserItemUPP`, using the function `NewDataBrowserItemUPP` (page 2194). You can do so with code similar to the following:

```
DataBrowserItemUPP MyDataBrowserItemUPP;
MyDataBrowserItemUPP = NewDataBrowserItemUPP
                            (&MyDataBrowserItemCallback);
```

You can then pass `MyDataBrowserItemUPP` in the `callback` parameter of the function `ForEachDataBrowserItem`. When you no longer need the UPP, remove it using the `DisposeDataBrowserItemUPP` (page 2135) function.

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X version 10.0 and later.

Declared In

`HIDataBrowser.h`

DataBrowserItemReceiveDragProcPtr

Defines a pointer to an item-receive-drag callback function that receives a drop over a custom item.

```
typedef Boolean (*DataBrowserItemReceiveDragProcPtr) (
    ControlRef browser,
    DataBrowserItemID itemID,
    DataBrowserPropertyID property,
    DataBrowserDragFlags dragFlags,
    DragRef theDrag
);
```

You would declare an item-receive-drag callback function named `MyDataBrowserItemReceiveDragCallback` like this:

```
Boolean MyDataBrowserItemReceiveDragCallback (
    ControlRef browser,
    DataBrowserItemID itemID,
    DataBrowserPropertyID property,
    DataBrowserDragFlags dragFlags,
    DragRef theDrag
);
```

Parameters

browser
: A data browser.

itemID
: The item ID number for the item over which the drop occurred.

property
: The property ID for the column in which the drop occurred. In list view, this is the four-character sequence that you previously assigned to the column. In column view, this is the property `kDataBrowserItemSelfIdentityProperty`.

dragFlags
: A drag flag. This value is `kDataBrowserItemIsDragTarget` if your item-accept-drag callback determined the drag object can be accepted.

theDrag
: The drag reference provided by the data browser to your callback.

Return Value

A value that indicates whether the drop is received. Your callback returns `true` if it successfully receives the drag object. If it returns `false`, zoom-back animation occurs.

Discussion

After your item-accept-drag callback has determined that a location can accept a drag object and after a drop operation occurs, the data browser calls your item-receive-drag callback. Your application takes whatever actions necessary to add the dropped data to the data browser.

Do not call the function `SetOrigin` in this or any of the drag processing callbacks.

To provide a pointer to your item-receive-drag callback, you create a universal procedure pointer (UPP) of type `DataBrowserItemReceiveDragUPP`, using the function `NewDataBrowserItemReceiveDragUPP` (page 2193). You can do so with code similar to the following:

```
DataBrowserItemReceiveDragUPP MyDataBrowserItemReceiveDragUPP;
```

```
MyDataBrowserItemReceiveDragUPP = NewDataBrowserItemReceiveDragUPP
                        (&MyDataBrowserItemReceiveDragCallback);
```

You can then assign `MyDataBrowserItemReceiveDragUPP` to the `itemReceiveDragCallback` field of the structure `DataBrowserCustomCallbacks` (page 2265). You install your data browser custom callbacks using the function `SetDataBrowserCustomCallbacks` (page 2203).

When you no longer need the UPP, remove it using the `DisposeDataBrowserItemReceiveDragUPP` (page 2135) function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HIDataBrowser.h`

DataBrowserPostProcessDragProcPtr

Defines a pointer to a postprocess-drag callback function that performs necessary cleanup tasks, such as deallocating resources that were allocated by your other drag processing callbacks.

```
typedef void (*DataBrowserPostProcessDragProcPtr) (
    ControlRef browser,
    DragRef theDrag,
    OSStatus trackDragResult
);
```

You would declare a postprocess-drag callback function named `MyDataBrowserPostProcessDragCallback` like this:

```
void MyDataBrowserPostProcessDragCallback (
    ControlRef browser,
    DragRef theDrag,
    OSStatus trackDragResult
);
```

Parameters

browser
A data browser.

theDrag
The drag reference provided by the data browser to your callback.

trackDragResult
The result returned by the function `TrackDrag` and passed to your callback by the data browser.

Discussion
This callback is called after starting a drag from within the data browser. It is not called if the drag originated from somewhere else.

The postprocess-drag callback is called by the data browser after a drag process is complete and any drag processing callback routines you installed (add drag, accept drag, or receive drag callbacks) were called during the drag operation. Your postprocess-drag callback deallocates any resources that were allocated by

your other drag-processing callbacks. Your postprocess-drag callback is called immediately before the drag reference is deallocated by the data browser so your application should not assume the drag reference exists after your callback completes.

To provide a pointer to your postprocess-drag callback, you create a universal procedure pointer (UPP) of type `DataBrowserPostProcessDragUPP`, using the function `NewDataBrowserPostProcessDragUPP` (page 2194). You can do so with code similar to the following:

```
DataBrowserPostProcessDragUPP MyDataBrowserPostProcessDragUPP;
MyDataBrowserPostProcessDragUPP = NewDataBrowserPostProcessDragUPP
                        (&MyDataBrowserPostProcessDragCallback);
```

You can then assign `MyDataBrowserPostProcessDragUPP` to the `postProcessDragCallback` field of the structure `DataBrowserCallbacks` (page 2264). You install your data browser callbacks using the function `SetDataBrowserCallbacks` (page 2200).

When you no longer need the UPP, remove it using the `DisposeDataBrowserPostProcessDragUPP` (page 2135) function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HIDataBrowser.h`

DataBrowserReceiveDragProcPtr

Defines a pointer to a receive-drag callback function that extract items from a drag object and handles the drag item appropriately.

```
typedef Boolean (*DataBrowserReceiveDragProcPtr) (
    ControlRef browser,
    DragRef theDrag,
    DataBrowserItemID item
);
```

You would declare a receive-drag callback function named `MyDataBrowserReceiveDragCallback` like this:

```
Boolean MyDataBrowserReceiveDragCallback (
    ControlRef browser,
    DragRef theDrag,
    DataBrowserItemID item
);
```

Parameters

browser
A data browser.

theDrag
The drag reference provided by the data browser to your callback.

item

The item ID of the item over which the drop operation occurred. If the drag object is over the data browser, but not over any specific item, the `item` parameter contains the item ID that represents one of the following:

- In list view, the target. (See `SetDataBrowserTarget` (page 2229).)
- In column view, the item ID that represents the column the drag object is over

Return Value
Your callback returns `true` if it successfully processes the information in the drag object. Otherwise, your callback returns `false` to have zoom-back animation occur for the drag object, thereby indicating to the user that the drag operation was not successful.

Discussion
The receive-drag callback is called by the data browser when your application needs to receive a drag object. Your application extracts the items it needs from the drag object and processes them appropriately.

To provide a pointer to your receive-drag callback, you create a universal procedure pointer (UPP) of type `DataBrowserReceiveDragUPP`, using the function `NewDataBrowserReceiveDragUPP` (page 2195). You can do so with code similar to the following:

```
DataBrowserReceiveDragUPP MyDataBrowserReceiveDragUPP;
MyDataBrowserReceiveDragUPP = NewDataBrowserReceiveDragUPP
                        (&MyDataBrowserReceiveDragCallback);
```

You can then assign `MyDataBrowserReceiveDragUPP` to the `receiveDragCallback` field of the structure `DataBrowserCallbacks` (page 2264). You install your data browser callbacks using the function `SetDataBrowserCallbacks` (page 2200).

When you no longer need the UPP, remove it using the `DisposeDataBrowserReceiveDragUPP` (page 2136) function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HIDataBrowser.h`

DataBrowserSelectContextualMenuProcPtr

Defines a pointer to a select-contextual-menu callback function that processes a contextual menu selection.

```
typedef void (*DataBrowserSelectContextualMenuProcPtr) (
    ControlRef browser,
    MenuRef menu,
    UInt32 selectionType,
    SInt16 menuID,
    MenuItemIndex menuItem
);
```

You would declare a select-contextual-menu callback function named `MyDataBrowserSelectContextualMenuCallback` like this:

```
void MyDataBrowserSelectContextualMenuCallback (
    ControlRef browser,
```

```
    MenuRef menu,
    UInt32 selectionType,
    SInt16 menuID,
    MenuItemIndex menuItem
);
```

Parameters

browser

A data browser.

menu

On input, the menu reference your application provided to the data browser in the callback `DataBrowserGetContextualMenuProcPtr` (page 2238).

selectionType

On input, the selection type provided to the data browser from the Menu Manager function `ContextualMenuSelect`.

menuID

On input, the menu ID of the menu selected. This value is `0` if no selection was made.

menuItem

The menu item index of the item selected.

Discussion

The select-contextual-menu callback is called by the data browser when the user finishes interacting with a contextual menu. Your callback needs to:

- Check whether the user chose an item from the menu. If so, process the selection appropriately. Note that your callback is invoked even if the user does not choose an item from the menu.
- Optionally dispose of the menu you allocated in your get-contextual-menu callback, and that is passed to your select-contextual-menu callback in the `menu` parameter.

To provide a pointer to your select-contextual-menu callback, you create a universal procedure pointer (UPP) of type `DataBrowserSelectContextualMenuUPP`, using the function `NewDataBrowserSelectContextualMenuUPP` (page 2195). You can do so with code similar to the following:

```
DataBrowserSelectContextualMenuUPP MyDataBrowserSelectContextualMenuUPP;
MyDataBrowserSelectContextualMenuUPP =
                 NewDataBrowserSelectContextualMenuUPP
                (&MyDataBrowserSelectContextualMenuCallback);
```

You can then assign `MyDataBrowserSelectContextualMenuUPP` to the `selectContextualMenuCallback` field of the structure `DataBrowserCallbacks` (page 2264). You install your data browser callbacks using the function `SetDataBrowserCallbacks` (page 2200).

When you no longer need the UPP, remove it using the `DisposeDataBrowserSelectContextualMenuUPP` (page 2136) function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`HIDataBrowser.h`

DataBrowserTrackingProcPtr

Defines a pointer to a tracking callback function that implements tracking behavior for such tasks as highlighting a button or providing animation when the user clicks a custom item.

```
typedef DataBrowserTrackingResult (*DataBrowserTrackingProcPtr)  (
    ControlRef browser,
    DataBrowserItemID itemID,
    DataBrowserPropertyID property,
    const Rect *theRect,
    Point startPt,
    EventModifiers modifiers
);
```

You would declare a tracking callback function named `MyDataBrowserTrackingCallback` like this:

```
DataBrowserTrackingResult MyDataBrowserTrackingCallback (
    ControlRef browser,
    DataBrowserItemID itemID,
    DataBrowserPropertyID property,
    const Rect *theRect,
    Point startPt,
    EventModifiers modifiers
);
```

Parameters

browser
: A data browser.

itemID
: The item ID number for the item the user clicked.

property
: The property ID for the column in which the pointer is located. In list view, this is the four-character sequence that you previously assigned to the column. In column view, this is the property `kDataBrowserItemSelfIdentityProperty`.

theRect
: A pointer to the bounding rectangle of the item, in local coordinates relative to the current port.

startPt
: The location of the pointer at the start of the click.

modifiers
: The state of the modifier keys. See *Carbon Event Manager Reference* in Carbon Events & Other Input Documentation for a list of the constants that can be passed to your callback.

Return Value

A tracking result that indicates whether further processing is required by the data browser. Your callback returns `kDataBrowserStopTracking`, `kDataBrowserContentHit`, or `kDataBrowserNothingHit`. See the Discussion for more details.

Discussion

The tracking callback is called by the data browser for an item whose display type is `kDataBrowserCustomType` when a mouse click is inside the content area of the item. Your tracking callback is called only for mouse-down events and only after your `DataBrowserHitTestProcPtr` (page 2240) callback returns `true`. Your tracking callback performs one of following tasks:

- Provides its own custom tracking behavior and animation and returns the result `kDataBrowserStopTracking`. This result informs the data browser that your application handled the click and no further processing is required. The data browser does not attempt to display a contextual menu, start a drag operation, process a double-click, or draw a selection rectangle. You are responsible for all facets of click handling if you return `kDataBrowserStopTracking`.
- Returns the value `kDataBrowserNothingHit` to indicate a negative hit and no further processing needs to take place. This result indicates that a nonselectable portion—whitespace—was hit. The data browser won't select the item, but it could, for example, start a selection rectangle.
- Returns the value `kDataBrowserContentHit` to request that the data browser continues to process the click. This result indicates that a selectable portion of the item is hit. The data browser selects the item and takes other appropriate actions.

To provide a pointer to your tracking callback, you create a universal procedure pointer (UPP) of type `DataBrowserTrackingUPP`, using the function `NewDataBrowserTrackingUPP` (page 2196). You can do so with code similar to the following:

```
DataBrowserTrackingUPP MyDataBrowserTrackingUPP;
MyDataBrowserTrackingUPP = NewDataBrowserTrackingUPP
                        (&MyDataBrowserTrackingCallback);
```

You can then assign `MyDataBrowserTrackingUPP` to the `trackingCallback` field of the structure `DataBrowserCustomCallbacks` (page 2265). You install your data browser custom callbacks using the function `SetDataBrowserCustomCallbacks` (page 2203).

When you no longer need the UPP, remove it using the `DisposeDataBrowserTrackingUPP` (page 2137) function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HIDataBrowser.h`

Data Types

DataBrowserAccessibilityItemInfo

Contains a structure that describes data browser accessibility item information.

```
struct DataBrowserAccessibilityItemInfo {
    UInt32 version;
    union {
        DataBrowserAccessibilityItemInfoV0 v0;
        DataBrowserAccessibilityItemInfoV1 v1;
    } u;
typedef struct DataBrowserAccessibilityItemInfoV0 v0;
typedef struct DataBrowserAccessibilityItemInfoV1 v1;
```

Fields

`version`

Identifies how to interpret the following union. Set this field to `0` if you fill out the union's data in the form of a `DataBrowserAccessibilityItemInfoV0` structure. Set this field to `1` if you fill out the union's data in the form of a `DataBrowserAccessibilityItemInfoV1` structure.

`u.v0`

A `DataBrowserAccessibilityItemInfoV0` (page 2261) structure.

`u.v1`

A `DataBrowserAccessibilityItemInfoV1` (page 2262) structure.

DataBrowserAccessibilityItemInfoV0

Contains a description of data browser accessibility item information.

```
struct DataBrowserAccessibilityItemInfoV0 {
    DataBrowserItemID container;
    DataBrowserItemID item;
    DataBrowserPropertyID columnProperty;
    DataBrowserPropertyPart propertyPart;
    }
typedef struct DataBrowserAccessibilityItemInfoV0  DataBrowserAccessibilityInfoV0;
```

Fields

`container`

The `DataBrowserItemID` of the container the `AXUIElementRef` represents or lives within. Even `kDataBrowserNoItem` might be meaningful, since it is the root container ID if you haven't overridden it via `SetDataBrowserTarget` (page 2229). In list view, the container helps narrow down the `AXUIElementRef` to either a disclosed child of another row, or the list as a whole. In column view, the container helps narrow down the `AXUIElementRef` to a column. See also the description of the `columnProperty` field.

`item`

The `DataBrowserItemID` of the item the `AXUIElementRef` represents or lives within. If `item` is `kDataBrowserNoItem`, the `AXUIElementRef` represents just the container. In list view, `item` helps narrow down the `AXUIElementRef` to a row or the root container as a whole. In column view, `item` helps narrow down the `AXUIElementRef` to a cell or a column as a whole. See also the description of the `columnProperty` field.

`columnProperty`
: The `DataBrowserPropertyID` of the column the `AXUIElementRef` represents or lives within. If `columnProperty` is `kDataBrowserItemNoProperty` and `item` is not `kDataBrowserNoItem`, the `AXUIElementRef` represents a whole row. In list view, this field helps narrow down the `AXUIElementRef` to a cell or a row as a whole. In column view, `columnProperty` must always be set to `kDataBrowserItemNoProperty` unless the `AXUIElementRef` represents the preview column. When the `AXUIElementRef` represents the preview column, `columnProperty` must always be set to `kDataBrowserColumnViewPreviewProperty` and the other fields of this structure must be set to `0` or the equivalent constant.

`propertyPart`
: The `DataBrowserPropertyPart` of the sub-cell part the `AXUIElementRef` represents. Examples include the disclosure triangle in a cell, the text in a cell, and the check box in a cell. If `propertyPart` is `kDataBrowserPropertyEnclosingPart` and `columnProperty` is not `kDataBrowserItemNoProperty`, the `AXUIElementRef` represents the cell as a whole. In both list view and column view, this field helps narrow down the `AXUIElementRef` to either a sub-cell part or a cell as a whole. For column view, see also the description of the `columnProperty` field.

DataBrowserAccessibilityItemInfoV1

Contains a description of data browser accessibility item information that includes a row and a column index.

```
struct DataBrowserAccessibilityItemInfoV1 {
    DataBrowserItemID container;
    DataBrowserItemID item;
    DataBrowserPropertyID columnProperty;
    DataBrowserPropertyPart propertyPart;
    DataBrowserTableViewRowIndex rowIndex;
    DataBrowserTableViewColumnIndex columnIndex;
    }
typedef struct DataBrowserAccessibilityItemInfoV1  DataBrowserAccessibilityInfoV1;
```

Fields

`container`
: The `DataBrowserItemID` of the container the `AXUIElementRef` represents or lives within. Even `kDataBrowserNoItem` might be meaningful, since it is the root container ID if you haven't overridden it via `SetDataBrowserTarget` (page 2229). In list view, the container helps narrow down the `AXUIElementRef` to either a disclosed child of another row, or the list as a whole. In column view, the container helps narrow down the `AXUIElementRef` to a column. See also the description of the `columnProperty` field.

`item`
: The `DataBrowserItemID` of the item the `AXUIElementRef` represents or lives within. If `item` is `kDataBrowserNoItem`, the `AXUIElementRef` represents just the container. In list view, `item` helps narrow down the `AXUIElementRef` to a row or the root container as a whole. In column view, `item` helps narrow down the `AXUIElementRef` to a cell or a column as a whole. See also the description of the `columnProperty` field.

`columnProperty`
: The `DataBrowserPropertyID` of the column the `AXUIElementRef` represents or lives within. If `columnProperty` is `kDataBrowserItemNoProperty` and `item` is not `kDataBrowserNoItem`, the `AXUIElementRef` represents a whole row. In list view, this field helps narrow down the `AXUIElementRef` to a cell or a row as a whole. In column view, `columnProperty` must always be set to `kDataBrowserItemNoProperty` unless the `AXUIElementRef` represents the preview column. When the `AXUIElementRef` represents the preview column, `columnProperty` must always be set to `kDataBrowserColumnViewPreviewProperty` and the other fields of this structure must be set to `0` or the equivalent constant.

`propertyPart`
: The `DataBrowserPropertyPart` of the sub-cell part the `AXUIElementRef` represents. Examples include the disclosure triangle in a cell, the text in a cell, and the check box in a cell. If `propertyPart` is `kDataBrowserPropertyEnclosingPart` and `columnProperty` is not `kDataBrowserItemNoProperty`, the `AXUIElementRef` represents the cell as a whole. In both list view and column view, this field helps narrow down the `AXUIElementRef` to either a sub-cell part or a cell as a whole. For column view, see also the description of the `columnProperty` field.

`rowIndex`
: The zero-based `DataBrowserTableViewRowIndex` of the row specified by the other parts of this structure. If the other parts of this structure do not specify a row or a part thereof, this field must be set to 0. Because this field is zero based, you must test the other parts of this structure to see whether this field is meaningful. In list view, when the other parts of this structure specify an item or part thereof, this field must be set to the row index at which the specified item can be found. In column view, when the other parts of this structure specify a cell or part thereof, this field must be set to the row index at which the specified cell can be found.

`propertyPart`
: The zero-based `DataBrowserTableViewColumnIndex` of the column specified by the other parts of this structure. If the other parts of this structure do not specify a column or a part thereof, this field must be set to zero. Because this field is zero based, you must test the other parts this structure to see whether this field is meaningful. In list view, when the other parts of this structure specify a cell or part thereof, this field must be set to the column index at which the specified cell can be found. In column view, when the other parts of this structure specify a column or part thereof, this field must be set to the column index at which the specified cell can be found.

DataBrowserPropertyDesc

Contains property and display information for a list view column.

```
struct DataBrowserPropertyDesc {
    DataBrowserPropertyID propertyID;
    DataBrowserPropertyType propertyType;
    DataBrowserPropertyFlags propertyFlags;
};
typedef struct DataBrowserPropertyDesc DataBrowserPropertyDesc;
```

Fields

`propertyID`
: A four-character sequence that uniquely identifies the column. If you use Interface Builder to design the data browser, this is the unique value you enter in the Property ID field in the column paned of the Info window for a list view column. (For example, `mTxt` or `BLUE`). The four-character sequence must have at least one uppercase letter in it because sequences that are all lowercase are reserved for Apple.

`propertyType`
: The data type or control type to be displayed in the column. See "Display Types" (page 2275) for a list of the possible values for this field.

`propertyFlags`
: A value that contains property flags that control the display or interaction provided by the column. This is a 32-bit value that is divided into four parts as follows:

- Bits 0–7 specify properties applied to the data browser as a whole—see "Property Flags: Universal" (page 2284)
- Bits 8–15 modify display behavior—see "Property Flags: Modifiers" (page 2285)
- Bits 16–23 are properties specific to list view—see "Property Flags: Offset and Mask for List View Properties" (page 2288) and "Property Flags: List View Column Behavior" (page 2289)
- Bits 24–31 can be defined by your application—see "Property Flags: Offset and Mask for Client-Defined Properties" (page 2290)

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HIDataBrowser.h`

DataBrowserCallbacks

Contains universal procedure pointers (UPPs) to callback functions used to obtain information from your application or notify your application of changes to the data browser.

```
struct DataBrowserCallbacks {
    UInt32 version
    union {
        struct {
            DataBrowserItemDataUPP itemDataCallback;
            DataBrowserItemCompareUPP itemCompareCallback;
            DataBrowserItemNotificationUPP itemNotificationCallback;
            DataBrowserAddDragItemUPP addDragItemCallback;
            DataBrowserAcceptDragUPP acceptDragCallback;
            DataBrowserReceiveDragUPP receiveDragCallback;
            DataBrowserPostProcessDragUPP postProcessDragCallback;
            DataBrowserItemHelpContentUPP itemHelpContentCallback;
            DataBrowserGetContextualMenuUPP getContextualMenuCallback;
            DataBrowserSelectContextualMenuUPP selectContextualMenuCallback;
        } v1;
    } u;
};
typedef struct DataBrowserCallbacks DataBrowserCallbacks;
```

Fields

`version`
: The version of the custom callbacks structure. Set this field to the constant `kDataBrowserLatestCallbacks`.

`u.v1.itemDataCallback`
: A universal procedure pointer to an item-data callback.

`u.v1.itemCompareCallback`
A universal procedure pointer to an item-compare callback.

`u.v1.itemNotificationCallback`
A universal procedure pointer to an item-notification callback or item-notification-with-data callback.

`u.v1.addDragItemCallback`
A universal procedure pointer to an add-drag-item callback.

`u.v1.acceptDragCallback`
A universal procedure pointer to an accept-drag callback.

`u.v1.receiveDragCallback`
A universal procedure pointer to a receive-drag callback.

`u.v1.postProcessDragCallback`
A universal procedure pointer to a postprocess-drag callback.

`u.v1.itemHelpContentCallback`
A universal procedure pointer to an item-help-content callback.

`u.v1.getContextualMenuCallback`
A universal procedure pointer to a get-contextual-menu callback.

`u.v1.selectContextualMenuCallback`
A universal procedure pointer to a select-contextual-menu callback.

Discussion
Your application does not need to fill out the entire data structure. It must provide a UPP to an item-data callback. You provide UPPs only for the other tasks your application wants to handle. For more information on installing callbacks, see the functions `InitDataBrowserCallbacks` (page 2175) and `SetDataBrowserCallbacks` (page 2200).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HIDataBrowser.h`

DataBrowserCustomCallbacks

Contains universal procedure pointers (UPPs) to callback functions that implement custom drawing and user interaction for columns that have the display type `kDataBrowserCustomType`.

```
struct DataBrowserCustomCallbacks {
    UInt32 version
    union {
        struct {
            DataBrowserDrawItemUPP drawItemCallback;
            DataBrowserEditItemUPP editTextCallback;
            DataBrowserHitTestUPP hitTestCallback;
            DataBrowserTrackingUPP trackingCallback;
            DataBrowserItemDragRgnUPP dragRegionCallback;
            DataBrowserItemAcceptDragUPP acceptDragCallback;
            DataBrowserItemReceiveDragUPP receiveDragCallback;
        } v1;
    } u;
};
typedef struct DataBrowserCustomCallbacks DataBrowserCustomCallbacks;
```

Fields

`version`
The version of the custom callbacks structure. Use `kDataBrowserLatestCustomCallbacks`.

`u.v1.drawItemCallback`
A universal procedure pointer to a draw-item callback.

`u.v1.editTextCallback`
A universal procedure pointer to an edit-text callback.

`u.v1.hitTestCallback`
A universal procedure pointer to a hit-test callback

`u.v1.trackingCallback`
A universal procedure pointer to a tracking callback.

`u.v1.dragRegionCallback`
A universal procedure pointer to an item-drag-region callback.

`u.v1.acceptDragCallback`
A universal procedure pointer to an item-accept-drag callback.

`u.v1.receiveDragCallback`
A universal procedure pointer to an item-receive-drag callback.

Discussion

Your application can use the `DataBrowserCustomCallbacks` structure to provide callbacks that control the presentation of user interface elements displayed inside a data browser. Your application does not need to fill out the entire data structure. You need to provide UPPs only for tasks your application wants to handle. For more information on installing callbacks, see the functions `InitDataBrowserCustomCallbacks` (page 2176) and `SetDataBrowserCustomCallbacks` (page 2203).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`HIDataBrowser.h`

DataBrowserDragFlags

Defines a data type for values used in drag callbacks.

```
typedef unsigned long DataBrowserDragFlags;
```

Discussion
This data type is used as a return value for the item-accept-drag callback (`DataBrowserItemAcceptDragProcPtr` (page 2242)) and as a parameter to the item-receive-drag callback (`DataBrowserItemReceiveDragProcPtr` (page 2254)). The values associated with this data type are `kDataBrowserItemIsDragTarget` and `kDataBrowserItemNoState`. See "Item States" (page 2280) for more information on these constants.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HIDataBrowser.h`

DataBrowserItemDataRef

Defines a data type for a pointer that specifies an item.

```
typedef void *DataBrowserItemDataRef;
```

Discussion
Functions listed in "Getting and Setting Item Data" (page 2115) are called from within a data browser item-data callback routine (`DataBrowserItemDataProcPtr` (page 2245)). Each of these functions use a `DataBrowserItemDataRef` data type to specify the item to get or set data for.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HIDataBrowser.h`

DataBrowserItemID

Defines a data type for a value that identifies an item independent of its position in a data browser.

```
typedef UInt32 DataBrowserItemID;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HIDataBrowser.h`

DataBrowserPropertyFlags

Defines a data type for a value that specifies the behavior and look of a data browser.

```
typedef unsigned long DataBrowserPropertyFlags;
```

Discussion
These constants in the following sections are `DataBrowserPropertyFlags` data types:

- "Property Flags: Universal" (page 2284). Bits 0–7 modify the appearance or behavior of display properties.
- "Property Flags: Modifiers" (page 2285). Bits 8–15 specify how the data associated with a display type is displayed.
- "Property Flags: Offset and Mask for List View Properties" (page 2288). Specify an offset and mask for bits 16–23.
- "Property Flags: List View Column Behavior" (page 2289). Bits 16–23 specify behaviors for columns in list view.
- "Property Flags: Offset and Mask for Client-Defined Properties" (page 2290). Specify an offset and mask for bits 24–31.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HIDataBrowser.h`

DataBrowserPropertyID

Defines a data type for a value that identifies a column independent of its position in a data browser.

```
typedef UInt32 DataBrowserPropertyID;
```

Discussion
Typically, this is a four-character sequence that you assign to represent a column in list view. For example, a column that displays song titles could be assigned a property ID of `SONG` and you'd assign it to a constant in your application using code similar to the following:

```
kSongColumn = SONG;
```

Then, each time call a function that requires a parameter of type `DataBrowserPropertyID`, supply the appropriate application-defined constant.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HIDataBrowser.h`

DataBrowserTableViewRowIndex

Defines a data type for a value that specifies a row position in a table view.

```
typedef UInt32 DataBrowserTableViewRowIndex;
```

Discussion
Row indices are zero based.

Availability
Available in Mac OS X v10.0 and later.

Declared In
HIDataBrowser.h

DataBrowserTableViewColumnIndex

Defines a data type for a value that specifies column position in a table view.

```
typedef UInt32 DataBrowserTableViewColumnIndex;
```

Discussion
This data type is typically used for table-view formatting. Column indices are zero based.

Availability
Available in Mac OS X v10.0 and later.

Declared In
HIDataBrowser.h

DataBrowserTableViewColumnID

Defines a data type for a value that identifies a column independent of its position in a data browser.

```
typedef DataBrowserPropertyID DataBrowserTableViewColumnID;
```

Discussion
For details on using this data type, see DataBrowserPropertyID (page 2268).

Availability
Available in Mac OS X v10.0 and later.

Declared In
HIDataBrowser.h

DataBrowserTableViewColumnDesc

Defines a data type for a table view column description.

```
typedef DataBrowserPropertyDesc DataBrowserTableViewColumnDesc;
```

Discussion
See the DataBrowserPropertyDesc (page 2263) data structure for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
HIDataBrowser.h

DataBrowserListViewHeaderDesc

Describes the appearance of a column title in list view.

```
struct DataBrowserListViewHeaderDesc {
    UInt32 version;
    UInt16 minimumWidth;
    UInt16 maximumWidth;
    SInt16 titleOffset;
    CFStringRef titleString;
    DataBrowserSortOrder initialOrder;
    ControlFontStyleRec btnFontStyle;
    ControlButtonContentInfo btnContentInfo;
};
typedef struct DataBrowserListViewHeaderDesc DataBrowserListViewHeaderDesc;
```

Fields

`version`
The format of the structure. Set this field to the value `kDataBrowserListViewLatestHeaderDesc`.

`minimumWidth`
For resizable columns, the smallest width to which the column can be resized. If the column is not resizable, set this to the same value as the `maximumWidth` field.

`maximumWidth`
For resizable columns, the largest width to which the column can be resized. If the column is not resizable, set this to the same value as the `minimumWidth` field.

`titleOffset`
An offset, in pixels, from the left side of the column that specifies where the title text will be drawn. The title alignment (set in the `just` field of the `btnFontStyle` parameter) and the `titleOffset` values dictate the alignment and offset (inset by default) of the content of the column when displaying one of he predefined display types. Typically the title offset is set to `0`.

`titleString`
The text to use for the column title. Set the string to `NULL` if you do not want to display a title.

`initialOrder`
The initial sorting order to use for the column when the column is the current sort column. After the data browser is visible, the user can change the sorting order. You can assign one of the following values:

- `kDataBrowserOrderIncreasing` means this column sorts in ascending order.
- `kDataBrowserOrderDecreasing` means this column sorts in descending order.

`btnFontStyle`
A structure that describes the contents of the column heading and how to draw them. This allows you to customize the font that the column title is drawn with, which is independent of the font used to draw the data in the column.

`btnContentInfo`
A structure that defines the icon, if any, to use for the column heading.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HIDataBrowser.h`

DataBrowserListViewColumnDesc

Contains property information for a list view column and specifies display information for the column title.

```
struct DataBrowserListViewColumnDesc {
    DataBrowserTableViewColumnDesc propertyDesc;
    DataBrowserListViewHeaderDesc headerBtnDesc;
};
typedef struct DataBrowserListViewColumnDesc DataBrowserListViewColumnDesc;
```

Fields

`propertyDesc`

A structure that contains property and display information for a column. See `DataBrowserTableViewColumnDesc` (page 2269) for more information.

`headerBtnDesc`

A structure that contains display information for the column title. See `DataBrowserListViewHeaderDesc` (page 2269) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HIDataBrowser.h`

kHIDataBrowserClassID

Defines the HIObject class ID for the HIDataBrowser class.

```
#define kHIDataBrowserClassID CFSTR("com.apple.HIDataBrowser");
```

Availability
Available in Mac OS X v10.4 and later.

Constants

Callback Data Structure Version

Specifies the version of the latest standard callback data structure.

```
enum {
    kDataBrowserLatestCallbacks = 0
};
```

Constants

`kDataBrowserLatestCallbacks`

A convenience constant used in the `DataBrowserCallbacks` (page 2264) data structure to specify the version of the structure.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

Control Data Tags

Define data-browser-specific tags for use with the Control Manager functions `GetControlData` and `SetControlData`.

```
enum {
  kControlDataBrowserIncludesFrameAndFocusTag = 'brdr',
  kControlDataBrowserKeyFilterTag = kControlEditTextKeyFilterTag,
  kControlDataBrowserEditTextKeyFilterTag =
                kControlDataBrowserKeyFilterTag,
  kControlDataBrowserEditTextValidationProcTag =
                kControlEditTextValidationProcTag
};
```

Constants

`kControlDataBrowserIncludesFrameAndFocusTag`
Include the frame and user focus. The associated data is of type `Boolean`.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kControlDataBrowserKeyFilterTag`
Use a filter. The associated data is a universal procedure pointer to a `ControlKeyFilterProcPtr` callback. This callback is invoked when the user edits an item.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kControlDataBrowserEditTextKeyFilterTag`
Use a text filter. This is a duplicate of `kControlDataBrowserKeyFilterTag`.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kControlDataBrowserEditTextValidationProcTag`
Use a callback to validate the text. The associated data is a universal procedure pointer to a `ControlEditTextValidataionProcPtr` callback. This callback is invoked when the user finishes editing an item.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

Discussion

You can use these options with the Control Manger functions `GetControlData` and `SetControlData`.

Custom Callback Data Structure Version

Specifies the version of the latest custom callback data structure.

```
enum {
    kDataBrowserLatestCustomCallbacks = 0
};
```

Constants

`kDataBrowserLatestCustomCallbacks`
A convenience constant used in the `DataBrowserCustomCallbacks` (page 2265) data structure to specify the version of the structure.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

Data Browser Attributes

Specifies data browser attribute constants.

```
enum {
kDataBrowserAttributeNone = 0,
kDataBrowserAttributeColumnViewResizeWindow = (1 << 0),
kDataBrowserAttributeListViewAlternatingRowColors = (1 <<
1)
kDataBrowserAttributeListViewDrawColumnDividers = (1 << 2)
};
```

Constants

`kDataBrowserAttributeNone`
The data browser has no attributes.

Available in Mac OS X v10.4 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserAttributeColumnViewResizeWindow`
In column view, this data browser is allowed to resize the owning window whenever necessary. This includes, but is not necessarily limited to, situations where column resize operations need more visible space in the window. If you turn this attribute on, your window must tolerate being resized behind your application's back. If your window needs to react to bounds changes, use a `kEventWindowBoundsChanged` event handler. If you need to constrain your window's minimum and maximum bounds, use the `kEventWindowGetMinimumSize` and `kEventWindowGetMaximumSize` handlers, the `SetWindowResizeLimits` function, or something similar.

Available in Mac OS X v10.4 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserAttributeListViewAlternatingRowColors`
In list view, this data browser should draw alternating row background colors. However, note that this attribute does not work with variable row heights as of Mac OS X v10.4.

Available in Mac OS X v10.4 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserAttributeListViewDrawColumnDividers`
In list view, this data browser should draw a vertical line between the columns.

Available in Mac OS X v10.4 and later.

Declared in `HIDataBrowser.h`.

Discussion
Use these constants in conjunction with `DataBrowserGetAttributes` (page 2127) and `DataBrowserChangeAttributes` (page 2127).

Availability
Available in Mac OS X v10.4 and later.

Data Browser Control Kind Tag

Specifies the control type is a data browser.

```
enum {
kControlKindDataBrowser       = 'datb'
};
```

Constants
`kControlKindDataBrowser`
The data browser control type.
Available in Mac OS X v10.0 and later.
Declared in `HIDataBrowser.h`.

Discussion
When you call the function `GetControlKind` with a control reference that represents a data browser, you obtain the data browser tag `kControlKindDataBrowser` as the control type.

Data Browser Metric Values

Specifies constants used by `DataBrowserSetMetric`.

```
enum {
kDataBrowserMetricCellContentInset = 1,
kDataBrowserMetricIconAndTextGap = 2,
kDataBrowserMetricDisclosureColumnEdgeInset = 3,
kDataBrowserMetricDisclosureTriangleAndContentGap = 4,
kDataBrowserMetricDisclosureColumnPerDepthGap = 5,
kDataBrowserMetricLast = kDataBrowserMetricDisclosureColumnPerDepthGap
};
typedef UInt32 DataBrowserMetric;
```

Constants
`kDataBrowserMetricCellContentInset`
The content (icon, text, etc.) within a cell is drawn a certain amount in from the left and right edges of the cell. This metric governs the amount of inset.
Available in Mac OS X v10.4 and later.
Declared in `HIDataBrowser.h`.

`kDataBrowserMetricIconAndTextGap`
This metric controls the space between the icon and text within a column of type `kDataBrowserIconAndTextType`.
Available in Mac OS X v10.4 and later.
Declared in `HIDataBrowser.h`.

`kDataBrowserMetricDisclosureColumnEdgeInset`
In list view only, this metric is used instead of (not in addition to) `DataBrowserMetricCellContentInset` for the side of the cell in the disclosure column that displays the disclosure triangle.

Available in Mac OS X v10.4 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserMetricDisclosureTriangleAndContentGap`
In list view only, this metric controls the amount of space between the disclosure triangle and the cell's content.

Available in Mac OS X v10.4 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserMetricDisclosureColumnPerDepthGap`
In list view only, this metric controls the amount of space in the disclosure column for each level of indentation in progressively deeper hierarchies of disclosed items.

Available in Mac OS X v10.4 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserMetricLast`
Same as `kDataBrowserMetricDisclosureColumnPerDepthGap`.

Available in Mac OS X v10.4 and later.

Declared in `HIDataBrowser.h`.

Display Types

Specify a data type or control to display in a column.

```
typedef OSType DataBrowserPropertyType;
enum {
    kDataBrowserCustomType = 0x3F3F3F3F,
    kDataBrowserIconType = 'icnr',
    kDataBrowserTextType = 'text',
    kDataBrowserDateTimeType = 'date',
    kDataBrowserSliderType = 'sldr',
    kDataBrowserCheckboxType = 'chbx',
    kDataBrowserProgressBarType = 'prog',
    kDataBrowserRelevanceRankType = 'rank',
    kDataBrowserPopupMenuType = 'menu',
    kDataBrowserIconAndTextType = 'ticn'
};
```

Constants

`kDataBrowserCustomType`
Displays custom data defined by your application. You must install callbacks to handle items of this type. Use custom types with caution. In some cases custom types do not display properly or exhibit the appropriate behavior.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserIconType`
Displays icons. The associated data for the icon can be of type `IconRef`, `IconTransformType`, and `RGBColor`.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserTextType`
Displays text. The associated data is a CFString object.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserDateTimeType`
Displays date and time information. The associated data can be of type `DateTime` or `LongDateTime`.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserSliderType`
Displays slider controls. The associated data are values that define the minimum and maximum values and the current value for the slider. Avoid using slider controls in a data browser because, in some cases, they do not display properly onscreen.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserCheckboxType`
Displays checkbox controls. The associated data is the current value of the checkbox and a `ThemeButtonValue` value. Avoid using checkbox controls in a data browser because, in some cases, they do not display properly onscreen.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserProgressBarType`
Displays progress bar controls. The associated data are values that define the minimum and maximum values and the current setting for the progress bar.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserRelevanceRankType`
Displays relevance indicators. The associated data are values that define the minimum and maximum values and the current setting for the relevance indicator. Avoid using relevance indicators in a data browser because, in some cases, they do not display properly onscreen.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserPopupMenuType`
Displays pop-up menus. The associated data is a `MenuRef` data type, a menu item ID, and the value of the pop-up menu, which indicates the item of the menu to draw in the pop-up menu. Avoid using pop-up menu controls in a data browser because, in some cases, they do not display properly onscreen.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserIconAndTextType`
Displays icon and text data. The associated data can be of any data type used for icons or text, such as an `IconRef` data type and a CFString object.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

Discussion
You can use these constants to define what is displayed in a column. Each constant defines a presentation style and implies a specific set of primitive types or data structures.

Editing Commands

Specifies editing actions to apply to a data browser item.

```
typedef UInt32 DataBrowserEditCommand;
enum {
    kDataBrowserEditMsgUndo = kHICommandUndo,
    kDataBrowserEditMsgRedo = kHICommandRedo,
    kDataBrowserEditMsgCut = kHICommandCut,
    kDataBrowserEditMsgCopy = kHICommandCopy,
    kDataBrowserEditMsgPaste = kHICommandPaste,
    kDataBrowserEditMsgClear = kHICommandClear,
    kDataBrowserEditMsgSelectAll = kHICommandSelectAll
};
```

Constants
`kDataBrowserEditMsgUndo`
Undo the last editing operation.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserEditMsgRedo`
Redo the last editing operation that was undone.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserEditMsgCut`
Cut the contents of the selection to the Clipboard.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserEditMsgCopy`
Copy the contents of the selection o the Clipboard.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserEditMsgPaste`
Replace the contents of the selection with the contents of the Clipboard.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserEditMsgClear`
Remove the contents of the selection.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserEditMsgSelectAll`
Select all of the text inside of the current edit session.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

Discussion
These options are for use with the functions `EnableDataBrowserEditCommand` (page 2137) and `ExecuteDataBrowserEditCommand` (page 2138).

Item Notifications

Specify notifications provided by the data browser to your application.

```
typedef UInt32 DataBrowserItemNotification;
enum {
    kDataBrowserItemAdded = 1,
    kDataBrowserItemRemoved = 2,
    kDataBrowserEditStarted = 3,
    kDataBrowserEditStopped = 4,
    kDataBrowserItemSelected = 5,
    kDataBrowserItemDeselected = 6,
    kDataBrowserItemDoubleClicked = 7,
    kDataBrowserContainerOpened = 8,
    kDataBrowserContainerClosing = 9,
    kDataBrowserContainerClosed = 10,
    kDataBrowserContainerSorting = 11,
    kDataBrowserContainerSorted = 12,
    kDataBrowserUserStateChanged = 13,
    kDataBrowserSelectionSetChanged = 14,
    kDataBrowserTargetChanged = 15,
    kDataBrowserUserToggledContainer = 16
};
```

Constants
`kDataBrowserItemAdded`
The specified item has been added to a container in the data browser.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserItemRemoved`
The specified item has been removed from a container in the data browser.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserEditStarted`
An text editing session has started for the specified item.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserEditStopped`
An text editing session has stopped for the specified item.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserItemSelected`
An item has been added to the selection set.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserItemDeselected`
An item has been removed from the selection set.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserItemDoubleClicked`
The user double-clicked an item.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserContainerOpened`
A container has been opened.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserContainerClosing`
A container is about to close. This notification is sent at the start of the close operation. The container still contains its child items and it is still valid to pass them to data browser functions. The data browser handles closing automatically, so typically applications do not look for this notification. You'd use this only if you are interested in fetching information on the items before the close actually happens.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserContainerClosed`
A container has been closed. This notification is sent after the close operation, so it is no longer valid to pass child items to data browser functions.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserContainerSorting`
A container is about to be sorted.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserContainerSorted`
A container has been sorted.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserUserStateChanged`
The user has reformatted the view for the target. For example, the user changed a sorting order or a column width.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserSelectionSetChanged`
The selection set has been modified.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserTargetChanged`
The target has changed to the specified item.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserUserToggledContainer`
The user has toggled opened or closed a container by clicking a disclosure triangle.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

Discussion
These notifications are used with the callbacks `DataBrowserItemNotificationProcPtr` (page 2250) and `DataBrowserItemNotificationWithItemProcPtr` (page 2251). Notifications sent for containers are the same regardless of whether the container opens to an expandable row or to individual rows for each item in the container.

Item States

Indicate the current state of an item in the data browser.

```
typedef UInt32 DataBrowserItemState;
enum {
    kDataBrowserItemNoState = 0,
    kDataBrowserItemAnyState = (unsigned long) (-1),
    kDataBrowserItemIsSelected = 1 << 0,
    kDataBrowserContainerIsOpen = 1 << 1,
    kDataBrowserItemIsDragTarget = 1 << 2
};
```

Constants
`kDataBrowserItemNoState`
The state is undefined.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserItemAnyState`
Any state is acceptable.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserItemIsSelected`
The item is selected.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserContainerIsOpen`
The item container is open. This state applies to:

- A parent item in a hierarchical list in list view
- An item in column view if the item's contents are displayed in the next column

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserItemIsDragTarget`
The item is a drag target.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

List View Header Description Version

Defines the version of the list view header description data structure.

```
enum {
    kDataBrowserListViewLatestHeaderDesc = 0
};
```

Constants

`kDataBrowserListViewLatestHeaderDesc`
A convenience constant that specifies the version of the list view header description data structure.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

List View Append Column

Defines the last column as the column to use for an append operation.

```
enum {
    kDataBrowserListViewAppendColumn = kDataBrowserTableViewLastColumn
};
```

Constants

`kDataBrowserListViewAppendColumn`
The column to use for an append operation.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

No Item Constant

Specifies that there is no item to provide or obtain.

```
enum {
    kDataBrowserNoItem = 0L
};
```

Constants

`kDataBrowserNoItem`
A convenience constant used when there is no item to provide or obtain. This value is of type `DataBrowserItemID`.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

Properties

Specify attributes for items, containers, and columns.

```
enum {
    kDataBrowserItemNoProperty = 0L,
    kDataBrowserItemIsActiveProperty = 1L,
    kDataBrowserItemIsSelectableProperty = 2L,
    kDataBrowserItemIsEditableProperty = 3L,
    kDataBrowserItemIsContainerProperty = 4L,
    kDataBrowserContainerIsOpenableProperty = 5L,
    kDataBrowserContainerIsClosableProperty = 6L,
    kDataBrowserContainerIsSortableProperty = 7L,
    kDataBrowserItemSelfIdentityProperty = 8L,
    kDataBrowserContainerAliasIDProperty = 9L,
    kDataBrowserColumnViewPreviewProperty = 10L,
    kDataBrowserItemParentContainerProperty = 11L
};
```

Constants

`kDataBrowserItemNoProperty`
No property; no associated data.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserItemIsActiveProperty`
The active state of the item. The associated data is of type `Boolean`. The default value `true` indicates the item is active.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserItemIsSelectableProperty`
The selection capability of the item. The associated data is of type `Boolean`. The default value `true` indicates the item can be selected.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserItemIsEditableProperty`
The editing capability of the item. The associated data is of type `Boolean`. The default value `false` indicates the item cannot be edited.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserItemIsContainerProperty`
The container attribute for an item. The associated data is of type `Boolean`. The default value `false` indicates the item cannot contain other items.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserContainerIsOpenableProperty`
The opening capability of a container item. The associated data is of type `Boolean`. The default value `true` indicates the container can be opened.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserContainerIsClosableProperty`
The closing capability of a container item. The associated data is of type `Boolean`. The default value `true` indicates the container can be closed.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserContainerIsSortableProperty`
The sorting capability of container item. The associated data is of type `Boolean`. The default value `true` indicates the items in the container can be sorted.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserItemSelfIdentityProperty`
This property is used only in column view for the item-data and item-compare callbacks. You should not use this property for anything else. It is passed to your item-data callback when the data browser needs to know the data to draw to represent the item. It is your responsibility to provide the data to display for that item in whatever format would be appropriate for the column view display type, which is `kDataBrowserIconAndTextType`.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserContainerAliasIDProperty`
An alias or symbolic link from an item to a container item. The associated data is of type `DataBrowserItemID`. This property is sent to your item-data callback to provide your application with a chance to follow an alias that the item might represent. If the incoming item is an alias to another item, you can call the function `SetDataBrowserItemDataItemID` to inform the data browser which other item the incoming item points to.

This property is sent only from column view. Your support for it is optional. Your response allows the data browser to be a bit more memory efficient with its internal storage. If a given item is an alias to an item whose contents are already displayed in one column of the column view, the contents can be shared between those two columns.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserColumnViewPreviewProperty`

The column displays a preview. There is no associated data. Available only in column view. This property is sent to your draw-item callback to indicate the need for you to draw a preview of the given item. It can be sent to other callbacks to provide an opportunity for your application to draw or track in the preview column.

You can also pass this in the `propertyID` parameter of the function `RevealDataBrowserItem` (along with the appropriate item ID of the item whose preview is displayed) to make sure the preview column is visible to the user.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserItemParentContainerProperty`

Designates the parent of the specified item. The associated data is of type `DataBrowserItemID`. This is sent to your item-data callback when the data browser needs to know the parent container item for a given item.

In column view, this allows the function `SetDataBrowserTarget` (page 2229) to process the data properly. The target is the leaf node item whose contents you want to display, which is the rightmost column in column view. However, unlike `SetDataBrowserColumnViewPath`, the function `SetDataBrowserTarget` doesn't offer a way for you to communicate the item IDs of the rest of the column containers, so `SetDataBrowserTarget` asks for them explicitly by requesting the item's parent, then the parent of the item's parent, and so on. (Your item-data callback might be called with the parent container property at times other than an explicit call to `SetDataBrowserTarget`, so your item-data callback should support this property.)

In list view, this property allows you to pass a non-container to `SetDataBrowserTarget`. In this case, the data browser requests the parent of the target so it knows which container to display the contents of in the list view. (Again, your item-data callback might be called with the parent container property at times other than an explicit call to `SetDataBrowserTarget`, so your item-data callback should be sure to support this property.)

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

Discussion
You can use these constants along with the appropriate value to manage the attributes of items in a data browser. Typically your item-data callback responds to inquires about these properties by calling the appropriate accessor function from within the callback. See `DataBrowserItemDataProcPtr` (page 2245) for more information on the item-data callback. "Getting and Setting Item Data" (page 2115) describes the accessor functions.

Property Flags: Universal

Modify the appearance or behavior of display properties.

```
enum {
    kDataBrowserUniversalPropertyFlagsMask = 0xFF,
    kDataBrowserPropertyIsMutable = 1 << 0,
    kDataBrowserDefaultPropertyFlags = 0 << 0,
    kDataBrowserUniversalPropertyFlags =  kDataBrowserUniversalPropertyFlagsMask,
    kDataBrowserPropertyIsEditable = kDataBrowserPropertyIsMutable
};
```

Constants

`kDataBrowserUniversalPropertyFlagsMask`
Test for universal property flags. This constant is used by the data browser; your application doesn't need to use it.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserPropertyIsMutable`
The property is mutable. You can assign this flag to the `propertyFlags` field of the `DataBrowserPropertyDesc` structure. You must set this flag is you want to allow editing of the text part of the property.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserDefaultPropertyFlags`
The default properties. You can assign this flag to the `propertyFlags` field of the `DataBrowserPropertyDesc` structure if all you need is the default behavior. The default is for all flags to be off.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserUniversalPropertyFlags`
Universal property flags. This constant is used by the data browser; your application doesn't need to use it.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserPropertyIsEditable`
The data can be edited. You must set this flag if you want to allow editing of the text part of the property. This flag can be set if the values displayed in the column can be changed. If your application specifies this flag, then it must also provide a callback that allows the data browser to retrieve and store data values displayed in this column. You can assign this flag to the `propertyFlags` field of the `DataBrowserPropertyDesc` structure.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

Discussion

These constants reside in bits 0–7 of the `DataBrowserPropertyFlags` data type.

Property Flags: Modifiers

Specify how to display the data associated with a display type.

```
enum {
    kDataBrowserPropertyFlagsOffset = 8,
    kDataBrowserPropertyFlagsMask =
                0xFF << kDataBrowserPropertyFlagsOffset,
    kDataBrowserCheckboxTriState  =
                1 << kDataBrowserPropertyFlagsOffset,
    kDataBrowserDateTimeRelative  =
                1 << (kDataBrowserPropertyFlagsOffset),
    kDataBrowserDateTimeDateOnly  =
                1 << (kDataBrowserPropertyFlagsOffset + 1),
    kDataBrowserDateTimeTimeOnly  =
                1 << (kDataBrowserPropertyFlagsOffset + 2),
    kDataBrowserDateTimeSecondsToo =
                1 << (kDataBrowserPropertyFlagsOffset + 3),
    kDataBrowserSliderPlainThumb  =
            kThemeThumbPlain << kDataBrowserPropertyFlagsOffset,
    kDataBrowserSliderUpwardThumb =
            kThemeThumbUpward << kDataBrowserPropertyFlagsOffset,
    kDataBrowserSliderDownwardThumb =
            kThemeThumbDownward << kDataBrowserPropertyFlagsOffset,
    kDataBrowserDoNotTruncateText =
                3 << kDataBrowserPropertyFlagsOffset,
    kDataBrowserTruncateTextAtEnd =
                2 << kDataBrowserPropertyFlagsOffset,
    kDataBrowserTruncateTextMiddle =
                0 << kDataBrowserPropertyFlagsOffset,
    kDataBrowserTruncateTextAtStart =
                1 << kDataBrowserPropertyFlagsOffset,
    kDataBrowserPopupMenuButtonless =
                1 << kDataBrowserPropertyFlagsOffset,
    kDataBrowserPropertyModificationFlags =
                 kDataBrowserPropertyFlagsMask,
    kDataBrowserRelativeDateTime  = kDataBrowserDateTimeRelative
};
```

Constants

`kDataBrowserPropertyFlagsOffset`

The offset value for this set of property flags.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserPropertyFlagsMask`

Use to set or test for property modifier flags.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserCheckboxTriState`

Modifies the `kDataBrowserCheckboxType` display type to display a checkbox that can have on, off, and mixed-mode states instead of just on and off states.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserDateTimeRelative`

Modifies the `kDataBrowserDateTimeType` display type to display relative date and time.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserDateTimeDateOnly`
Modifies the `kDataBrowserDateTimeType` display type to display only the date.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserDateTimeTimeOnly`
Modifies the `kDataBrowserDateTimeType` display type to display only the time.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserDateTimeSecondsToo`
Modifies the `kDataBrowserDateTimeType` display type to display the time with seconds.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserSliderPlainThumb`
Modifies the `kDataBrowserSliderType` display type to display a round thumb.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserSliderUpwardThumb`
Modifies the `kDataBrowserSliderType` display type to display a directional thumb that points up.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserSliderDownwardThumb`
Modifies the `kDataBrowserSliderType` display type to display a directional thumb that points down.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserDoNotTruncateText`
Modifies the `kDataBrowserTextType` and the `kDataBrowserIconAndTextType` display types so they do not truncate text.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserTruncateTextAtEnd`
Modifies the `kDataBrowserTextType` and the `kDataBrowserIconAndTextType` display types so they truncate text, if needed, at the end of the text string.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserTruncateTextMiddle`
Modifies the `kDataBrowserTextType` and the `kDataBrowserIconAndTextType` display types so they truncate text, if needed, in the middle of the text string.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserTruncateTextAtStart`
Modifies the `kDataBrowserTextType` and the `kDataBrowserIconAndTextType` display type so they truncate text, if needed, at the beginning of the text string.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserPopupMenuButtonless`
This flag is only for use with columns of type `kDataBrowserPopupMenuType` and indicates that the popup is to be drawn in a sleek buttonless fashion. The text is drawn next to a popup glyph, and the whole cell is clickable.

Available on Mac OS X v10.4 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserPropertyModificationFlags`
Old name. Instead use `kDataBrowserPropertyFlagsMask`.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserRelativeDateTime`
Old name. Instead use `kDataBrowserDateTimeRelative`.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

Discussion
These constants reside in bits 8–15 of the `DataBrowserPropertyFlags` data type.

Property Flags: Offset and Mask for List View Properties

Specify an offset and mask for bits 16–23 of the `DataBrowserPropertyFlag` data type.

```
enum {
    kDataBrowserViewSpecificFlagsOffset = 16,
    kDataBrowserViewSpecificFlagsMask =
            0xFF << kDataBrowserViewSpecificFlagsOffset,
    kDataBrowserViewSpecificPropertyFlags =
            kDataBrowserViewSpecificFlagsMask
};
```

Constants
`kDataBrowserViewSpecificFlagsOffset`
The offset value for this set of property flags.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserViewSpecificFlagsMask`
Use to set or test for view-specific property flags.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserViewSpecificPropertyFlags`
Old name. Instead use `kDataBrowserViewSpecificFlagsMask`.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

Discussion
See also "Property Flags: List View Column Behavior" (page 2289).

Property Flags: List View Column Behavior

Specify behaviors for columns in list view.

```
typedef DataBrowserPropertyFlags DataBrowserListViewPropertyFlags;
enum {
    kDataBrowserListViewSelectionColumn =
                kDataBrowserTableViewSelectionColumn,
    kDataBrowserListViewMovableColumn =
                1 << (kDataBrowserViewSpecificFlagsOffset  + 1),
    kDataBrowserListViewSortableColumn =
                1 << (kDataBrowserViewSpecificFlagsOffset  + 2),
    kDataBrowserListViewTypeSelectColumn =
                1 << (kDataBrowserViewSpecificFlagsOffset  + 3),
    kDataBrowserListViewNoGapForIconInHeaderButton =
                1 << (kDataBrowserViewSpecificFlagsOffset  + 4),
    kDataBrowserListViewDefaultColumnFlags =
              kDataBrowserListViewMovableColumn +
              kDataBrowserListViewSortableColumn
};
```

Constants
`kDataBrowserListViewSelectionColumn`
If you are using a minimally highlighted list, this indicates to draw the contents of this column as highlighted when the item is selected. (Minimal highlighting is the highlighting used by the Finder for list view prior to Mac OS X version 10.3.)

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserListViewMovableColumn`
The column is movable.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserListViewSortableColumn`
The column can be sorted.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserListViewTypeSelectColumn`
: The column is capable of being selected and having text entered. If one or more of your list view columns are marked as type-selectable, the data browser handles type-selection for you automatically. You can use this flag with columns whose display type is `kDataBrowserTextType`, `kDataBrowserIconAndTextType`, or `kDataBrowserDateTimeType`. If you set this flag for a column of another type, the type-select behavior is undefined. Turning on this flag causes the data browser to obtain keyboard input using a Carbon event handler instead of relying on calls to the function `HandleControlKey`.

Declared in `HIDataBrowser.h`.

Available in Mac OS X 10.3 and later.

`kDataBrowserListViewNoGapForIconInHeaderButton`
: Normally the text in a header button for a column of type `kDataBrowserIconAndTextType` is aligned as though it has an icon next to it even if no icon is specified for the header button. This flag indicates that space should not be reserved for an icon if no icon is provided for the header button. This flag allows a client to justify the left edge of the text in a header button to the left edge of the icon in the cells beneath it.

Declared in `HIDataBrowser.h`.

Available in Mac OS X v10.4 and later.

`kDataBrowserListViewDefaultColumnFlags`
: The default properties. You can assign this flag to the `propertyFlags` field of the `DataBrowserPropertyDesc` structure if all you need is the default behavior for list view.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

Discussion
These constants reside in bits 16–23 of the `DataBrowserPropertyFlags` data type and are specific to a list view.

Property Flags: Offset and Mask for Client-Defined Properties

Specify an offset and mask for the high 8 bits of the `DataBrowserPropertyFlag` data type.

```
enum {
    kDataBrowserClientPropertyFlagsOffset = 24,
    kDataBrowserClientPropertyFlagsMask = (unsigned long)
            (0xFF << kDataBrowserClientPropertyFlagsOffset)
};
```

Constants
`kDataBrowserClientPropertyFlagsOffset`
: The offset value for this set of property flags.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserClientPropertyFlagsMask`
: The mask value for this set of property flags.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

Discussion
Bits 24–31 of the `DataBrowserPropertyFlags` data type are reserved for use by your application. You can set and query them for your own purposes.

Property Parts

Specify the visual components of a data type or control displayed in a column.

```
typedef OSType DataBrowserPropertyPart;
enum {
    kDataBrowserPropertyEnclosingPart = 0,
    kDataBrowserPropertyContentPart = '----',
    kDataBrowserPropertyDisclosurePart = 'disc',
    kDataBrowserPropertyTextPart = kDataBrowserTextType,
    kDataBrowserPropertyIconPart = kDataBrowserIconType,
    kDataBrowserPropertySliderPart = kDataBrowserSliderType,
    kDataBrowserPropertyCheckboxPart = kDataBrowserCheckboxType,
    kDataBrowserPropertyProgressBarPart = kDataBrowserProgressBarType,
    kDataBrowserPropertyRelevanceRankPart =
                kDataBrowserRelevanceRankType
};
```

Constants

`kDataBrowserPropertyEnclosingPart`
The outer boundary of an item.
Available in Mac OS X v10.0 and later.
Declared in `HIDataBrowser.h`.

`kDataBrowserPropertyContentPart`
The content of an item.
Available in Mac OS X v10.0 and later.
Declared in `HIDataBrowser.h`.

`kDataBrowserPropertyDisclosurePart`
The location of a disclosure rectangle.
Available in Mac OS X v10.0 and later.
Declared in `HIDataBrowser.h`.

`kDataBrowserPropertyTextPart`
The location where text is drawn.
Available in Mac OS X v10.0 and later.
Declared in `HIDataBrowser.h`.

`kDataBrowserPropertyIconPart`
The location where an icon is displayed.
Available in Mac OS X v10.0 and later.
Declared in `HIDataBrowser.h`.

`kDataBrowserPropertySliderPart`
The location of a slider.
Available in Mac OS X v10.0 and later.
Declared in `HIDataBrowser.h`.

`kDataBrowserPropertyCheckboxPart`
The location of a checkbox.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserPropertyProgressBarPart`
The location of a progress bar.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserPropertyRelevanceRankPart`
The location of a relevance indicator.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

Discussion
You can pass these constants as a parameter to the function `GetDataBrowserItemPartBounds` (page 2156) to obtain a rectangle that contains the bounds for the specified part.

Reveal Options

Specify how to position an item in a data browser.

```
typedef UInt8 DataBrowserRevealOptions;
enum {
    kDataBrowserRevealOnly = 0,
    kDataBrowserRevealAndCenterInView = 1 << 0,
    kDataBrowserRevealWithoutSelecting = 1 << 1
};
```

Constants
`kDataBrowserRevealOnly`
Move the content of the data browser as little as possible to make the item visible, and show the item in a selected state.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserRevealAndCenterInView`
Reveal the item so that, if possible, the item is centered in the data browser.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserRevealWithoutSelecting`
Reveal the item but do not select it.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

Discussion
You can pass reveal options as a parameter to the function `RevealDataBrowserItem` (page 2199). You can pass more than one at a time for combined functionality.

Selection Anchor Directions

Specify an anchor direction to apply when the user drags a data selection.

```
typedef UInt32 DataBrowserSelectionAnchorDirection;
enum {
    kDataBrowserSelectionAnchorUp = 0,
    kDataBrowserSelectionAnchorDown = 1,
    kDataBrowserSelectionAnchorLeft = 2,
    kDataBrowserSelectionAnchorRight = 3
};
```

Constants

`kDataBrowserSelectionAnchorUp`
Apply the anchor direction at the top of the selection set.
Available in Mac OS X v10.0 and later.
Declared in `HIDataBrowser.h`.

`kDataBrowserSelectionAnchorDown`
Apply the anchor direction at the bottom of the selection set.
Available in Mac OS X v10.0 and later.
Declared in `HIDataBrowser.h`.

`kDataBrowserSelectionAnchorLeft`
Apply the anchor direction at the left of the selection set.
Available in Mac OS X v10.0 and later.
Declared in `HIDataBrowser.h`.

`kDataBrowserSelectionAnchorRight`
Apply the anchor direction at the right of the selection set.
Available in Mac OS X v10.0 and later.
Declared in `HIDataBrowser.h`.

Discussion

These options are for use with the function `MoveDataBrowserSelectionAnchor` (page 2186).

Selection State Options

Specify how the selection state should be affected by a given list of items.

```
typedef UInt32 DataBrowserSetOption;
enum {
    kDataBrowserItemsAdd = 0,
    kDataBrowserItemsAssign = 1,
    kDataBrowserItemsToggle = 2,
    kDataBrowserItemsRemove = 3
};
```

Constants

`kDataBrowserItemsAdd`
Add specified items to the existing set or selection.
Available in Mac OS X v10.0 and later.
Declared in `HIDataBrowser.h`.

`kDataBrowserItemsAssign`
Assign the destination set to the specified item and redraw the list appropriately.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserItemsToggle`
Toggle the membership state of the specified items. Any of the items in the current selection are removed from the selection, and those items that are not in the selection are added to the selection.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserItemsRemove`
Remove specified items from the existing set and redraw the items so they are not highlighted.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

Discussion
These options are for use with the function `SetDataBrowserSelectedItems` (page 2222).

Sorting Orders

Specify the order in which to sort data.

```
typedef UInt16 DataBrowserSortOrder;
enum {
    kDataBrowserOrderUndefined = 0,
    kDataBrowserOrderIncreasing = 1,
    kDataBrowserOrderDecreasing = 2
};
```

Constants
`kDataBrowserOrderUndefined`
This constant has no meaning in the context of your application; don't use it.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserOrderIncreasing`
Sort in increasing order.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserOrderDecreasing`
Sort in decreasing order.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

Table View Highlighting Styles

Specify a highlighting style to use in list view.

```
typedef UInt32 DataBrowserTableViewHiliteStyle;
enum {
    kDataBrowserTableViewMinimalHilite = 0,
    kDataBrowserTableViewFillHilite = 1
};
```

Constants

`kDataBrowserTableViewMinimalHilite`

Use minimal highlighting. This is the highlighting used by the Finder for list view prior to Mac OS X v. 10.3. For this style, the highlight color for active items is `kThemeBrushPrimaryHighlightColor` and for inactive items the color is `kThemeBrushSecondaryHighlightColor`.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserTableViewFillHilite`

Use full-row highlighting. This is the highlighting used by the Finder for list view in Mac OS X v. 10.3. For this style, the highlight color in Mac OS X v. 10.3 for active items is `kThemeBrushAlternatePrimaryHighlightColor` and for inactive items the color is `kThemeBrushSecondaryHighlightColor`. Prior to Mac OS X v. 10.3 the highlight color for active and inactive items is `kThemeBrushPrimaryHighlightColor`.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

Table View Last Column Value

Specifies the last column position.

```
enum {
    kDataBrowserTableViewLastColumn = -1
};
```

Constants

`kDataBrowserTableViewLastColumn`

The last column position.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

Table View Property Flag

Specifies a table view property.

```
typedef UInt32 DataBrowserTableViewPropertyFlags;
enum {
   kDataBrowserTableViewSelectionColumn = 1 <<  kDataBrowserViewSpecificFlagsOffset
};
```

Constants

`kDataBrowserTableViewSelectionColumn`

The column can be selected.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

Tracking Results

Specify the outcome of tracking a drag operation.

```
typedef SInt16 DataBrowserTrackingResult;
enum {
    kDataBrowserContentHit = 1,
    kDataBrowserNothingHit = 0,
    kDataBrowserStopTracking = -1
};
```

Constants

`kDataBrowserContentHit`
Indicates that a selectable portion of the item was hit. The data browser will select the item and do other relevant actions as appropriate.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserNothingHit`
Indicates that a nonselectable portion—whitespace—was hit. The data browser won't select the item, but it could, for example, start a selection rectangle.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

`kDataBrowserStopTracking`
Indicates the callback handled the click completely. The data browser will not attempt to display a contextual menu, start a drag, process a double-click, or draw a selection rectangle. The callback is responsible for all facets of click handling if it returns this value.

Available in Mac OS X v10.0 and later.

Declared in `HIDataBrowser.h`.

Discussion

These constants are returned from the custom tracking callback, `DataBrowserTrackingProcPtr` (page 2259).

User Selection Flags

Specify allowable selection behavior.

```
typedef UInt32 DataBrowserSelectionFlags;
enum {
    kDataBrowserDragSelect = 1 << 0,
    kDataBrowserSelectOnlyOne = 1 << 1,
    kDataBrowserResetSelection = 1 << 2,
    kDataBrowserCmdTogglesSelection = 1 << 3,
    kDataBrowserNoDisjointSelection = 1 << 4,
    kDataBrowserAlwaysExtendSelection = 1 << 5,
    kDataBrowserNeverEmptySelectionSet = 1 << 6
};
```

Constants

`kDataBrowserDragSelect`
: Allows items to be selected by dragging. If the user to clicks the mouse to select a nondraggable item and drags the mouse, any item the pointer moves over is selected. This is on by default.

  Available in Mac OS X v10.0 and later.

  Declared in `HIDataBrowser.h`.

`kDataBrowserSelectOnlyOne`
: Only one item can be selected at a time. If you use this flag in conjunction with the flag `kDataBrowserDragSelect`, then as the user drags, previous items are deselected as each new item is dragged over.

  Available in Mac OS X v10.0 and later.

  Declared in `HIDataBrowser.h`.

`kDataBrowserResetSelection`
: Clears all selected items before processing the next selection. This is off by default.

  Available in Mac OS X v10.0 and later.

  Declared in `HIDataBrowser.h`.

`kDataBrowserCmdTogglesSelection`
: Enables use of a command to toggle items in and out of a selection. This is on by default.

  Available in Mac OS X v10.0 and later.

  Declared in `HIDataBrowser.h`.

`kDataBrowserNoDisjointSelection`
: Prevents a discontinuous selection.

  Available in Mac OS X v10.0 and later.

  Declared in `HIDataBrowser.h`.

`kDataBrowserAlwaysExtendSelection`
: Multiple items can be selected without holding down a modifier key.

  Available in Mac OS X v10.0 and later.

  Declared in `HIDataBrowser.h`.

`kDataBrowserNeverEmptySelectionSet`
: There must always be at least one selected item; the user cannot deselect the last selected item.

  Available in Mac OS X v10.0 and later.

  Declared in `HIDataBrowser.h`.

Discussion

You can use one or more of these flags together.

View Styles

Specify a style to use when displaying data in a data browser.

```
typedef OSType DataBrowserViewStyle;
enum {
    kDataBrowserNoView = '????',
    kDataBrowserListView = 'lstv',
    kDataBrowserColumnView = 'clmv'
};
```

Constants

`kDataBrowserNoView`
There is no view.
Available in Mac OS X v10.0 and later.
Declared in `HIDataBrowser.h`.

`kDataBrowserListView`
Display data in a list. Lists can be hierarchical.
Available in Mac OS X v10.0 and later.
Declared in `HIDataBrowser.h`.

`kDataBrowserColumnView`
Display data in columns that can be browsed.
Available in Mac OS X v10.0 and later.
Declared in `HIDataBrowser.h`.

Result Codes

The most common result codes returned by the data browser are listed here.

Result Code	Value	Description
`errDataBrowserNotConfigured`	-4970	The data browser is not properly configured. Available in Mac OS X v10.0 and later.
`errDataBrowserItemNotFound`	-4971	The item is not in the data browser. Available in Mac OS X v10.0 and later.
`errDataBrowserPropertyNotFound`	-4972	The property is not in the data browser. Available in Mac OS X v10.0 and later.
`errDataBrowserInvalidPropertyPart`	-4973	The property part is not valid. Available in Mac OS X v10.0 and later.
`errDataBrowserInvalidPropertyData`	-4974	The property data is not valid. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
`errDataBrowserItemNotAdded`	-4975	The item was not added to the data browser. Available in Mac OS X v10.0 and later.
`errDataBrowserPropertyNotSupported`	-4979	The property is not supported in the data browser. Available in Mac OS X v10.0 and later.

Fonts Panel Reference

Framework: CommonPanels.framework
Declared in FontPanel.h

Overview

The Fonts Panel programming interface is a collection of functions and data types that support the use of a Fonts panel in a Carbon application. The Fonts panel is the preferred user interface for fonts in Mac OS X.

Functions

FPIsFontPanelVisible

Checks whether the Fonts panel is visible.

```
Boolean FPIsFontPanelVisible (
   void
);
```

Return Value
Returns `true` if the Fonts panel is visible; `false` otherwise. See the Mac Types documentation for a description of the `Boolean` data type.

Availability
Not available in CarbonLib 1.x.
Available in Mac OS X version 10.2.

Declared In
`FontPanel.h`

FPShowHideFontPanel

Shows or hides the Fonts panel.

```
OSStatus FPShowHideFontPanel (
   void
);
```

Return Value
A result code. See "Fonts Panel Results Codes" (page 2308).

Discussion
The function `FPShowHideFontPanel` shows the Fonts panel if the panel is not visible and hides the Fonts panel if the panel is visible.

Availability
Not available in CarbonLib 1.x.
Available in Mac OS X version 10.2.

Declared In
`FontPanel.h`

SetFontInfoForSelection

Sets the selections that appear in the Fonts panel.

```
OSStatus SetFontInfoForSelection (
   OSType iStyleType,
   UInt32 iNumStyles,
   void *iStyles,
   EventTargetRef iFPEventTarget
);
```

Parameters

`iStyleType`
The type of style data you are passing to the Fonts panel. Pass `kFontSelectionQDType` if you supply a `FontSelectionQDStyle` data structure in the `iStyles` parameter. Pass `kFontSelectionATSUIType` if you supply `ATSUStyle` data in the `iStyles` parameter.

`iNumStyles`
The number of styles passed in the `iStyles` parameter.

`iStyles`
A pointer to an array of data structures that correspond to the style type specified by the `iStyleType` parameter.

`iFPEventTarget`
The event target you want to receive the Carbon event `kEventFontSelection` that is associated with the Fonts panel. You typically specify the event target for the window that is sending the style selection to the Fonts panel.

Return Value
A result code. See "Fonts Panel Results Codes" (page 2308).

Discussion
You can programmatically set a selection in the Fonts panel by calling the function `SetFontInfoForSelection`. You can call this function even when the Fonts panel is not open or visible. When the Fonts panel becomes visible later, the style information specified in the most recent call to `SetFontInfoForSelection` is selected in the panel.

Availability
Not available in CarbonLib 1.x.
Available in Mac OS X version 10.2.

Declared In
`FontPanel.h`

Data Types

FontSelectionQDStyle

Contains QuickDraw style font information.

```
struct FontSelectionQDStyle {
    UInt32 version;
    FMFontFamilyInstance instance;
    FMFontSize size;
    Boolean hasColor;
    UInt8 reserved;
    RGBColor color;
};
typedef struct FontSelectionQDStyle FontSelectionQDStyle;
typedef FontSelectionQDStyle * FontSelectionQDStylePtr;
```

Fields

`version`
: Specifies the version number of the data structure.

`instance`
: Specifies a font family instance.

`size`
: Specifies the font size in points.

`hasColor`
: Specifies whether the font has a color other than black.

`reserved`
: Reserved for future use.

`color`
: Specifies the font color.

Discussion

You should use this data structure to specify QuickDraw-style font information when you call the function `SetFontInfoForSelection` (page 2302). When you use this data structure, you must pass the constant `kFontSelectionQDType` in the `iStyleType` parameter of the function.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`FontPanel.h`

FontSelectionQDStylePtr

A pointer to a data structure that contains QuickDraw style font information.

```
typedef FontSelectionQDStyle * FontSelectionQDStylePtr;
```

Discussion

See `FontSelectionQDStyle` (page 2303) for more information.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`FontPanel.h`

Constants

Font Event Class

Specifies a Carbon event related to font selection or handling.

```
enum {
    kEventClassFont = 'font'
};
```

Constants

`kEventClassFont`
Specifies a Carbon event related to font selection or handling.
Available in Mac OS X v10.2 and later.
Declared in `FontPanel.h`.

Declared In
`FontPanel.h`

Fonts Panel Events

Specify a Fonts panel Carbon event.

```
enum {
    kEventFontPanelClosed = 1,
    kEventFontSelection = 2
};
```

Constants

`kEventFontPanelClosed`
Specifies a close event in the Fonts panel. Your application should update the user interface as necessary. For example, you may need to change a Hide Fonts Panel menu command to Show Fonts Panel.
Available in Mac OS X v10.2 and later.
Declared in `FontPanel.h`.

`kEventFontSelection`
Specifies a selection event in the Fonts panel. Your application can obtain the selections from the event parameters associated with this event. Your application must check for all those parameters it handles and apply the selections to the text.
Available in Mac OS X v10.2 and later.
Declared in `FontPanel.h`.

Discussion
When the user closes the Fonts panel, the action generates the Carbon event `kEventWindowClosed`. You can then update any menu items or other controls whose state may need to change. The event `kEventWindowClosed` has no parameters. However, when the user selects an item in the Fonts panel, the system sends the Carbon event `kEventFontSelection` to the event target you specified when your application called the Fonts Panel function `SetFontInfoForSelection` (page 2302). The event `kEventFontSelection` has parameters that reflect the current Fonts panel selection.

Declared In
`FontPanel.h`

Font Information Types

Specify the type of font information.

```
enum {
    kFontSelectionATSUIType = 'astl',
    kFontSelectionQDType = 'qstl'
};
```

Constants
`kFontSelectionATSUIType`
Specifies ATSUI data (`ATSUStyle`).
Available in Mac OS X v10.2 and later.
Declared in `FontPanel.h`.

`kFontSelectionQDType`
Specifies QuickDraw data (`FontSelectionQDStyle`).
Available in Mac OS X v10.2 and later.
Declared in `FontPanel.h`.

Discussion
These constants specify the type of font information passed to the function `SetFontInfoForSelection` (page 2302).

Declared In
`FontPanel.h`

Font Information Versions

Specify the supported versions of the `FontSelectionQDStyle` data structure.

```
enum {
    kFontSelectionQDStyleVersionZero = 0
};
```

Constants
`kFontSelectionQDStyleVersionZero`
Specifies version 0 of a QuickDraw font selection.
Available in Mac OS X v10.2 and later.
Declared in `FontPanel.h`.

Discussion
You should always set the `version` field in the `FontSelectionQDStyle` data structure to this constant

Declared In
`FontPanel.h`

Fonts Panel Command

Specifies the four-character code for the show/hide Fonts panel command.

```
enum {
    kHICommandShowHideFontPanel = 'shfp'
};
```

Constants
`kHICommandShowHideFontPanel`
Specifies the four-character code for the show/hide Fonts panel command.
Available in Mac OS X v10.2 and later.
Declared in `FontPanel.h`.

Discussion
If the user closes the Fonts panel directly, your application receives a `kEventFontPanelClosed` Carbon event. You can then call the Carbon Event Manager function `GetEventParameter` to extract the HI command from the event.

Declared In
`FontPanel.h`

Font Parameters and Data Types

Specify font event parameters and their associated data types.

```
enum {
    typeATSUFontID = typeUInt32,
    typeATSUSize = typeFixed,
    typeFMFontFamily = typeSInt16,
    typeFMFontStyle = typeSInt16,
    typeFMFontSize = typeSInt16,
    typeFontColor = typeRGBColor,
    kEventParamATSUFontID = 'auid',
    kEventParamATSUFontSize = 'ausz',
    kEventParamFMFontFamily = 'fmfm',
    kEventParamFMFontStyle = 'fmst',
    kEventParamFMFontSize = 'fmsz',
    kEventParamFontColor = 'fclr'
};
```

Constants
`typeATSUFontID`
Specifies that an `ATSUFontID` is of type `UInt32`.
Available in Mac OS X v10.2 and later.
Declared in `FontPanel.h`.

`typeATSUSize`
Specifies that an `ATSUSize` is of type `Fixed`.

Available in Mac OS X v10.2 and later.

Declared in `FontPanel.h`.

`typeFMFontFamily`
Specifies that an `FMFontFamily` is of type `SInt16`.

Available in Mac OS X v10.2 and later.

Declared in `FontPanel.h`.

`typeFMFontStyle`
Specifies that an `FMFontStyle` is of type `SInt16`.

Available in Mac OS X v10.2 and later.

Declared in `FontPanel.h`.

`typeFMFontSize`
Specifies that an `FMFontSize` is of type `SInt16`.

Available in Mac OS X v10.2 and later.

Declared in `FontPanel.h`.

`typeFontColor`
Specifies that a font color is of type `RGBColor`.

Available in Mac OS X v10.2 and later.

Declared in `FontPanel.h`.

`kEventParamATSUFontID`
Specifies the four-character code associated with an `ATSUFontID` selection in the Fonts panel.

Available in Mac OS X v10.2 and later.

Declared in `FontPanel.h`.

`kEventParamATSUFontSize`
Specifies the four-character code associated with an `ATSUFontSize` selection in the Fonts panel.

Available in Mac OS X v10.2 and later.

Declared in `FontPanel.h`.

`kEventParamFMFontFamily`
Specifies the four-character code associated with an `FMFontFamily` selection in the Fonts panel.

Available in Mac OS X v10.2 and later.

Declared in `FontPanel.h`.

`kEventParamFMFontStyle`
Specifies the four-character code associated with an `FMFontStyle` selection in the Fonts panel.

Available in Mac OS X v10.2 and later.

Declared in `FontPanel.h`.

`kEventParamFMFontSize`
Specifies the four-character code associated with an `FMFontSize` selection in the Fonts panel.

Available in Mac OS X v10.2 and later.

Declared in `FontPanel.h`.

`kEventParamFontColor`

Specifies the four-character code associated with a font color selection in the Fonts panel.

Available in Mac OS X v10.2 and later.

Declared in `FontPanel.h`.

Discussion

Table 28-1 (page 2308) lists the parameters and data types associated with the Carbon event `kEventFontSelection`.

Table 28-1 Parameters and parameter data types for a font selection event.

Parameter	Data type
`kEventParamATSUFontID`	`typeATSUFontID`
`kEventParamATSUFontSize`	`typeATSUFontSize`
`kEventParamFMFontFamily`	`typeFMFontFamily`
`kEventParamFMFontStyle`	`typeFMFontStyle`
`kEventParamFMFontSize`	`typeFMFontSize`
`kEventParamFontColor`	`typeFontColor`

Declared In

`FontPanel.h`

Result Codes

The most common result codes returned by the Fonts Panel programming interface are listed below.

Result Code	Value	Description
`fontPanelShowErr`	-8880	The Fonts panel could not be opened. Available in Mac OS X v10.2 and later.
`fontPanelSelectionStyleErr`	-8881	Your application specified an invalid style type in the `iStyleType` parameter of the `SetFontInfoForSelection` (page 2302) function. Available in Mac OS X v10.2 and later.
`fontPanelFontSelectionQDStyleVersionErr`	-8882	Your application specified an invalid version in the `version` field of the `FontSelectionQDStyle` data structure. Available in Mac OS X v10.2 and later.

HIArchive Reference

Framework: Carbon/Carbon.h

Declared in HIArchive.h

Overview

HIArchive provides a convenient and standardized mechanism for flattening data objects so they can be stored in memory or on disk. Applications can use these archives whenever they need to package complex data. For example, you can use archives to:

- Store document data
- Transfer data using pasteboards, drag-and-drop, streams, or Apple events
- Store localization strings and user interface elements in the same package

HIArchive encodes archives in the binary property list format. You can convert archives to a text XML format using the `plutil` property list tool accessible from Terminal. HIArchive is comparable to (and uses the same underlying mechanism as) the Cocoa NSKeyedArchiver/Unarchiver classes.

For details about using HIArchive, see *HIArchive Programming Guide*.

HIArchive is available in Mac OS X version 10.4 and later.

Functions by Task

Storing Objects in an Archive

`HIArchiveCreateForEncoding` (page 2312)
Creates an HIArchive object to store objects.

`HIArchiveEncodeBoolean` (page 2314)
Stores a Boolean value in an archive.

`HIArchiveEncodeNumber` (page 2315)
Stores a number in an archive.

`HIArchiveEncodeCFType` (page 2314)
Stores a CFType object in an archive.

`HIArchiveCopyEncodedData` (page 2311)
Compresses an archive for storage.

Retrieving Objects from an Archive

`HIArchiveCreateForDecoding` (page 2311)
Creates an HIArchive object to retrieve objects.

`HIArchiveDecodeBoolean` (page 2312)
Retrieves a Boolean value from an archive.

`HIArchiveDecodeNumber` (page 2313)
Retrieves a number from an archive.

`HIArchiveCopyDecodedCFType` (page 2310)
Retrieves a CFType object from an archive.

Miscellaneous Function

`HIArchiveGetTypeID` (page 2316)
Obtains the CFType ID for HIArchive objects.

Functions

HIArchiveCopyDecodedCFType

Retrieves a CFType object from an archive.

```
OSStatus HIArchiveCopyDecodedCFType (
   HIArchiveRef inDecoder,
   CFStringRef inKey,
   CFTypeRef *outCFType
);
```

Parameters

inDecoder
The archive holding the CFType object to retrieve.

inKey
A Core Foundation string key identifying the CFType object to retrieve.

outCFType
On return, `outCFType` points to the retrieved CFType object.

Return Value
A result code.

Discussion
You also use this function for retrieving HIObjects and objects subclassed from HIObject.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`HIArchive.h`

HIArchiveCopyEncodedData

Compresses an archive for storage.

```
OSStatus HIArchiveCopyEncodedData (
   HIArchiveRef inEncoder,
   CFDataRef *outData
);
```

Parameters

inEncoder

The archive to compress.

outData

On return, `outData` points to the compressed archive.

Return Value

A result code.

Discussion

When you have finished adding data to an archive, calling `HIArchiveCopyEncodedData` compresses the data and returns it to you as a CFData object. You can use the returned data reference to store or transfer the data as you choose, for example writing it to a file or copying it to a pasteboard.

After compression, you can release the original HIArchive reference by calling `CFRelease`.

Availability

Available in Mac OS X v10.4 and later.

Not available to 64-bit applications.

Declared In

`HIArchive.h`

HIArchiveCreateForDecoding

Creates an HIArchive object to retrieve objects.

```
OSStatus HIArchiveCreateForDecoding (
   CFDataRef inData,
   OptionBits inOptions,
   HIArchiveRef *outDecoder
);
```

Parameters

inData

A CFData reference pointing to archived data. This archive was originally written to a data stream using `HIArchiveCopyEncodedData` (page 2311). This data reference does not have to be the one originally returned by `HIArchiveCopyEncodedData` (page 2311), but it must contain a copy of the same data.

inOptions

Any decoding options. Currently the only option is `kHIArchiveDecodeSuperclassForUnregisteredObjects`.

outDecoder

On return, `outDecoder` points to the newly created HIArchive object.

Return Value
A result code.

Discussion
You use this function when you want to retrieve data from an existing HIArchive.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`HIArchive.h`

HIArchiveCreateForEncoding

Creates an HIArchive object to store objects.

```
OSStatus HIArchiveCreateForEncoding (
   HIArchiveRef *outEncoder
);
```

Parameters

outEncoder
On return, `outEncoder` points to the newly created HIArchive object.

Return Value
A result code.

Discussion
Before you can archive any objects, you must create an HIArchive object in which to store them.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`HIArchive.h`

HIArchiveDecodeBoolean

Retrieves a Boolean value from an archive.

```
OSStatus HIArchiveDecodeBoolean (
   HIArchiveRef inDecoder,
   CFStringRef inKey,
   Boolean *outBoolean
);
```

Parameters

inDecoder
The archive holding the Boolean value.

inKey
A Core Foundation string key identifying the Boolean to retrieve.

outBoolean
On return, outBoolean points to the retrieved Boolean value.

Return Value
A result code.

Discussion
This function is a convenience wrapper that calls HIArchiveCopyDecodedCFType (page 2310) to obtain a CFBoolean value.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
HIArchive.h

HIArchiveDecodeNumber

Retrieves a number from an archive.

```
OSStatus HIArchiveDecodeNumber (
   HIArchiveRef inDecoder,
   CFStringRef inKey,
   CFNumberType inNumberType,
   void *outNumberValue
);
```

Parameters
inDecoder
The archive holding the number to retrieve.

inKey
A Core Foundation string key identifying the number to retrieve.

inNumberType
A CFNumber type identifying the type of number value to be retrieved. For example, kCFNumberSInt32Type. See CFNumber Reference in Core Foundation Reference Documentation for additional possible values.

outNumberValue
Before calling, outNumberValue must point to a number variable of the type and size you specified in inNumberType. On return, outNumberValue points to the retrieved number.

Return Value
A result code.

Discussion
This function is a convenience wrapper that calls HIArchiveCopyDecodedCFType (page 2310) to obtain a CFNumber value.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
HIArchive.h

HIArchiveEncodeBoolean

Stores a Boolean value in an archive.

```
OSStatus HIArchiveEncodeBoolean (
   HIArchiveRef inEncoder,
   CFStringRef inKey,
   Boolean inBoolean
);
```

Parameters

inEncoder
The archive to store the Boolean value.

inKey
A Core Foundation string key identifying the Boolean value.

inBoolean
The Boolean value.

Return Value
A result code.

Discussion
This function is a convenience wrapper that calls `HIArchiveEncodeCFType` (page 2314) with a CFBoolean value.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`HIArchive.h`

HIArchiveEncodeCFType

Stores a CFType object in an archive.

```
OSStatus HIArchiveEncodeCFType (
   HIArchiveRef inEncoder,
   CFStringRef inKey,
   CFTypeRef inCFType
);
```

Parameters

inEncoder
The archive to store the CFType object.

inKey
A Core Foundation string key identifying the CFType object.

inCFType
The CFType object to store in the archive.

Return Value
A result code.

Discussion
You can only encode base CFType objects that correspond to archivable NSFoundation objects. For example, type `CFStringRef` is supported, but type `HIShapeRef` is not.

You also use this function for storing HIObjects and objects subclassed from HIObject. Currently only the following HIObject subclass types support archiving:

- `HIObjectRef`
- `HIViewRef`
- `WindowRef`
- `ControlRef`
- `MenuRef`

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`HIArchive.h`

HIArchiveEncodeNumber

Stores a number in an archive.

```
OSStatus HIArchiveEncodeNumber (
   HIArchiveRef inEncoder,
   CFStringRef inKey,
   CFNumberType inNumberType,
   const void *inNumberValue
);
```

Parameters

inEncoder
The archive to store the number.

inKey
A Core Foundation string key identifying the number.

inNumberType
A CFNumber type identifying the type of number value to be stored, for example, `kCFNumberSInt32Type`. See CFNumber Reference in Core Foundation Reference Documentation for additional possible values.

inNumberValue
A pointer to the number value.

Return Value
A result code.

Discussion
This function is a convenience wrapper that calls `HIArchiveEncodeCFType` (page 2314) with a CFNumber value.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`HIArchive.h`

HIArchiveGetTypeID

Obtains the CFType ID for HIArchive objects.

```
CFTypeID HIArchiveGetTypeID (
   void
);
```

Return Value
The Core Foundation type ID for the HIArchive object type.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`HIArchive.h`

Data Types

HIArchiveRef

Defines an uncompressed archive object.

```
typedef struct OpaqueHIArchiveRef* HIArchiveRef;
```

Discussion
The structure pointed to by this reference is opaque. `HIArchiveRef` is a CFType, and therefore responds to `CFRetain` and `CFRelease` calls.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`HIArchive.h`

Constants

Archive Decoding Option Constant

Defines options available when calling `HIArchiveCreateForDecoding` (page 2311).

```
enum {

  kHIArchiveDecodeSuperclassForUnregisteredObjects = (1 <<  0)
};
```

Constants

`kHIArchiveDecodeSuperclassForUnregisteredObjects`

If the class of the HIObject you are attempting to decode is not a registered subclass, this option allows `HIArchiveCopyDecodedCFType` (page 2310) to instantiate the object as its superclass, if it exists. For example, if your application has not yet registered `com.myCorp.mycustomView` before attempting to unarchive an instance of that HIView, HIArchive instantiates the data as class `com.apple.hiview`. Only data written to the superclass is decoded; any data unique to the unregistered subclass is ignored. Specifying this option also signals the HIObject to load its custom archive data so you can access it by calling `HIObjectCopyCustomArchiveData`.

This option can be useful when creating an archive editor that doesn't implement all the objects contained in a client archive.

Available in Mac OS X v10.4 and later.

Declared in `HIArchive.h`.

Result Codes

Result Code	Value	Description
`noErr`	0	No error. Available in Mac OS X v10.0 and later.
`hiArchiveTypeMismatchErr`	-6780	The encoding or decoding archive was passed into a noncorresponding function. (For example, an archive created for encoding was passed into a decoding function.) Available in Mac OS X v10.4 and later.
`hiArchiveKeyNotAvailableErr`	-6781	The requested key does not exist in the specified archive. Available in Mac OS X v10.4 and later.
`hiArchiveEncodingCompleteErr`	-6782	`HIArchiveCopyEncodedData` (page 2311) was called on this archive, so no more data can be added. Available in Mac OS X v10.4 and later.
`hiArchiveHIObjectIgnoresArchivingErr`	-6783	The HIObject you wanted to encode does not support the HIArchive protocol. Available in Mac OS X v10.4 and later.

HIGeometry Reference

Framework: Carbon/Carbon.h

Declared in HIGeometry.h

Overview

HIGeometry is a Quartz-compatible API for describing and manipulating basic geometric objects such as points, rectangles, and sizes. HIGeometry expresses all coordinates using floating-point numbers. This API provides functions to convert an object's coordinates into a different coordinate space. These functions support resolution independence mode drawing by taking into account the scale factor of your application's user interface.

Functions by Task

Getting the Scale Factor

`HIGetScaleFactor` (page 2320)
Returns the scale factor of an application's user interface.

Converting Coordinates

These functions allow conversion between global pixel coordinates, global virtual (72-dpi) coordinates, window coordinates, and view coordinates.

`HIPointConvert` (page 2320)
Converts a point from one coordinate space to another.

`HIRectConvert` (page 2321)
Converts a rectangle from one coordinate space to another.

`HISizeConvert` (page 2322)
Converts a size structure from one coordinate space to another.

Functions

HIGetScaleFactor

Returns the scale factor of an application's user interface.

```
float HIGetScaleFactor (
   void
);
```

Return Value
A positive number that represents the scale factor of your application.

Discussion
The scale factor of your application's user interface is the ratio between device space units and user space units. For example, if the scale factor is 2.0 and you draw a horizontal line with a length of 10 units, the graphics system will display a line with a length of 20 pixels.

The scale factor is initialized when your application launches. In Mac OS X v10.4, the scale factor is always 1.0 (unless you use the Quartz Debug application to change it). In future versions of Mac OS X, the scale factor may vary based on user preference or the resolution of the main display.

Application frameworks such as Cocoa and Carbon use the scale factor to adjust the size of your application's user interface so that it has the appropriately scaled appearance when viewed on the main display.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

See Also
`HIWindowGetScaleMode` (page 1889)

Declared In
`HIGeometry.h`

HIPointConvert

Converts a point from one coordinate space to another.

```
void HIPointConvert (
   HIPoint *ioPoint,
   HICoordinateSpace inSourceSpace,
   void *inSourceObject,
   HICoordinateSpace inDestinationSpace,
   void *inDestinationObject
);
```

Parameters
ioPoint
A pointer to the point to convert. On output, the point contains the destination coordinates.

inSourceSpace

A constant specifying the source coordinate space from which the point is to be converted. Some coordinate spaces require the caller to pass extra information in the `inSourceObject` parameter.

inSourceObject

A pointer to an object defining the source coordinate space from which the point is to be converted. You may pass an object of type `WindowRef` (page 1987) or `HIViewRef` (page 2497). If no object is necessary, you must pass `NULL`. See "Coordinate Space Constants" (page 2324) for details on which coordinate spaces require objects.

inDestinationSpace

A constant specifying the destination coordinate space to which the point is to be converted. Some coordinate spaces require the caller to pass extra information in the `inDestinationObject` parameter.

inDestinationObject

A pointer to an object defining the destination coordinate space to which the point is to be converted. You may pass an object of type `WindowRef` (page 1987) or `HIViewRef` (page 2497). If no object is necessary, you must pass `NULL`. See "Coordinate Space Constants" (page 2324) for details on which coordinate spaces require objects.

Discussion

When converting a point from one coordinate space to another, this function takes into account the scale factor of your application's user interface. If the source or destination object is a view, the view should already be embedded in a window. If both the source and destination objects are views, the views are not required to be embedded in the same window.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`HIGeometry.h`

HIRectConvert

Converts a rectangle from one coordinate space to another.

```
void HIRectConvert (
   HIRect *ioRect,
   HICoordinateSpace inSourceSpace,
   void *inSourceObject,
   HICoordinateSpace inDestinationSpace,
   void *inDestinationObject
);
```

Parameters

ioRect

A pointer to the rectangle to convert. On output, the rectangle contains the destination origin and size.

inSourceSpace

A constant specifying the source coordinate space from which the rectangle is to be converted. Some coordinate spaces require the caller to pass extra information in the `inSourceObject` parameter.

inSourceObject

A pointer to an object defining the source coordinate space from which the rectangle is to be converted. You may pass an object of type `WindowRef` (page 1987) or `HIViewRef` (page 2497). If no object is necessary, you must pass `NULL`. See "Coordinate Space Constants" (page 2324) for details on which coordinate spaces require objects.

inDestinationSpace

A constant specifying the destination coordinate space to which the rectangle is to be converted. Some coordinate spaces require the caller to pass extra information in the `inDestinationObject` parameter.

inDestinationObject

A pointer to an object defining the destination coordinate space to which the rectangle is to be converted. You may pass an object of type `WindowRef` (page 1987) or `HIViewRef` (page 2497). If no object is necessary, you must pass `NULL`. See "Coordinate Space Constants" (page 2324) for details on which coordinate spaces require objects.

Discussion

When converting a rectangle from one coordinate space to another, this function takes into account the scale factor of your application's user interface. If the source or destination object is a view, the view should already be embedded in a window. If both the source and destination objects are views, the views are not required to be embedded in the same window.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`HIGeometry.h`

HISizeConvert

Converts a size structure from one coordinate space to another.

```
void HISizeConvert (
   HISize *ioSize,
   HICoordinateSpace inSourceSpace,
   void *inSourceObject,
   HICoordinateSpace inDestinationSpace,
   void *inDestinationObject
);
```

Parameters

ioSize

A pointer to the size structure to convert. On output, the structure contains the destination size.

inSourceSpace

A constant specifying the source coordinate space from which the size is to be converted. Some coordinate spaces require the caller to pass extra information in the `inSourceObject` parameter.

inSourceObject

A pointer to an object defining the source coordinate space from which the size is to be converted. You may pass an object of type `WindowRef` (page 1987) or `HIViewRef` (page 2497). If no object is necessary, you must pass `NULL`. See "Coordinate Space Constants" (page 2324) for details on which coordinate spaces require objects.

inDestinationSpace

A constant specifying the destination coordinate space to which the size is to be converted. Some coordinate spaces require the caller to pass extra information in the `inDestinationObject` parameter.

inDestinationObject

A pointer to an object defining the destination coordinate space to which the size is to be converted. You may pass an object of type `WindowRef` (page 1987) or `HIViewRef` (page 2497). If no object is necessary, you must pass `NULL`. See "Coordinate Space Constants" (page 2324) for details on which coordinate spaces require objects.

Discussion
When converting a size structure (width and height) from one coordinate space to another, this function takes into account the scale factor of your application's user interface. If the source or destination object is a view, the view should already be embedded in a window. If both the source and destination objects are views, the views are not required to be embedded in the same window.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`HIGeometry.h`

Data Types

HIPoint

Defines the position of a point using floating-point coordinates.

```
typedef CGPoint HIPoint;
```

Discussion
The `HIPoint` type is a data structure that defines the position of a point (x,y) in a floating-point coordinate space. When you obtain a point of type `HIPoint` from an HIToolbox function or a Carbon event, typically the y-axis of the drawing coordinate space is inverted with the origin (0,0) in the upper-left corner of the main display. Note that although it replaces the QuickDraw `Point` data structure, the `HIPoint` data structure does not contain the same fields.

Availability
Available in Mac OS X v10.1 and later.

Declared In
`HIGeometry.h`

HISize

Defines the width and height of an object using floating-point coordinates.

```
typedef CGSize HISize;
```

Availability
Available in Mac OS X v10.2 and later.

Declared In
HIGeometry.h

HIRect

Defines the position and size of a rectangle using floating-point coordinates.

```
typedef CGRect HIRect;
```

Discussion
The HIRect type is a data structure that defines the position and size (width and height) of a rectangle in a floating-point coordinate space. When you obtain a rectangle of type HIRect from an HIToolbox function or a Carbon event, typically the y-axis of the drawing coordinate space is inverted with the origin (0,0) in the upper-left corner of the main display. In this case, the position or origin of the rectangle is its upper-left corner. Note that although it replaces the QuickDraw Rect data structure, the HIRect data structure does not contain the same fields.

Availability
Available in Mac OS X v10.2 and later.

Declared In
HIGeometry.h

Constants

Coordinate Space Constants

Specify coordinate spaces used in HIToolbox.

```
typedef UInt32 HICoordinateSpace;
enum {
    kHICoordSpace72DPIGlobal      = 1,
    kHICoordSpaceScreenPixel      = 2,
    kHICoordSpaceWindow           = 3,
    kHICoordSpaceView             = 4
};
```

Constants
kHICoordSpace72DPIGlobal
Specifies a global coordinate space that has been adjusted by the scale factor of your application's user interface. For example, if a user interface object is 125 x 100 pixels on the screen and the scale factor is 1.25, the size of the object in this coordinate space is 100 x 80. The origin of this coordinate space is the upper-left corner of the main display, and the y-axis is inverted. This is the compatibility coordinate space; Carbon functions that do not take an explicit HICoordinateSpace parameter, such as existing Window Manager, QuickDraw, and Display Manager functions, assume that coordinate parameters are expressed in this space. When the scale factor is 1.0, this coordinate space and kHICoordSpaceScreenPixel are the same.

Available in Mac OS X v10.4 and later.

Declared in HIGeometry.h.

`kHICoordSpaceScreenPixel`

Specifies a coordinate space defined by the screen size in pixels. The origin of this coordinate space is the upper-left corner of the main display, and the y-axis is inverted. When the scale factor of your application's user interface is 1.0, this coordinate space and `kHICoordSpace72DPIGlobal` are the same.

Available in Mac OS X v10.4 and later.

Declared in `HIGeometry.h`.

`kHICoordSpaceWindow`

Specifies the coordinate space of a window of type `WindowRef` (page 1987). The origin of this coordinate space is the upper-left corner of the window's structure region, and the y-axis is inverted. When this constant is passed to a function as a source or destination coordinate space, you must also pass a window as a source or destination object.

Available in Mac OS X v10.4 and later.

Declared in `HIGeometry.h`.

`kHICoordSpaceView`

Specifies the coordinate space of a view of type `HIViewRef` (page 2497). The origin of this coordinate space is the upper-left corner of the view's bounds, and the y-axis is inverted. When this constant is passed to a function as a source or destination coordinate space, you must also pass a view as a source or destination object.

Available in Mac OS X v10.4 and later.

Declared in `HIGeometry.h`.

Declared In
`HIGeometry.h`

HIObject Reference

Framework: Carbon/Carbon.h

Declared in HIObject.h

Overview

HIObject is the base class for various objects in the HIToolbox. In Mac OS X v10.2 and later, most common user interface objects (controls, windows, menus, toolbars, and toolbar items) are derived from HIObject. Code that is external to HIToolbox can also create its own subclasses of these objects using the routines contained in this API. There are also polymorphic functions one can use on any HIObject for getting the class ID, and so on.

HIObjects are actually Core Foundation CF types under the hood. This means that they can be put into CF collections and you can retain/release them.

An HIObject is essentially a very basic building-block object which contains an event target. You can create these objects to use as your own Carbon Event receptors in your application, or you can subclass existing HIToolbox objects to suit your needs.

You register your subclasses with `HIObjectRegisterSubclass` (page 2335), passing your class ID, the parent class, and an event handler. You also pass a list of events the handler is interested in.

To create an object of your subclass, you call `HIObjectCreate` (page 2330), passing the class reference you registered, as well as an initialization event.

Construction is two-phase: first the basic construction of the object is done, then initialization is performed. The HIToolbox sends construction events bottom-up, as you would expect in C++ or the like. Here is the list of what goes on to create an object:

1) The HIToolbox creates the base HIObject

2) It then installs the event handler you specified when you registered your subclass. Your handler must listen for `kEventHIObjectConstruct` and `kEventHIObjectDestruct` events. If it does not, the class cannot be registered (you will get a `paramErr`).

3) Next, the HIToolbox directly calls your handler with an `kEventHIObjectConstruct` event. When called like this, you are not really being called in the context of a handler stack, so you cannot do things like `CallNextEventHandler`. The `userData` parameter is what you specified when you registered the class. Typically, during construction you will allocate memory yourself to store your own instance data; this allocation might be as simple as calling `malloc` or `NewPtr`, or it might involve creating your own C++ object. In the construct event, you are passed the base `HIObjectRef` of the object being created. Typically you would store this `HIObjectRef` in your own instance data for later use. When handling this construct event, you should be sure to use `SetEventParameter` to set the `kEventParamHIObjectInstance` parameter in the construction event with your own instance data. You must use `typeVoidPtr` as the type.

4) The HIToolbox looks for your instance of `typeVoidPtr` after you handle the construct event. It then takes that data and stores it off with the object and also sets the user data of the event handler it installed to be this instance data. This means that following the construct event, all calls to your event handler will have the instance data you returned to us.

5) Once construction has completed successfully, we will send your object the initialize event passed into `HIObjectCreate`. At this point, all events are now sent to your object using standard Carbon event mechanisms (it is only the construct event which is special). When we send the initialization event to your subclass, you should pass the event to your superclass before proceeding. You do this with `CallNextEventHandler`. Once back from that call, you should verify that the result is noErr, indicating that the superclass did in fact initialize properly. If it did not, your should return the error that `CallNextEventHandler` returned from your handler as well. The object will be destroyed by the HIToolbox. Your object should be able to be destroyed in a partially initialized state such as this. This stage is optional, i.e. an object does not need to respond to the initialize event unless it is expecting certain parameters to be passed to it at creation time. This is where those parameters can be fetched.

6) Once initialization is successful, the `HIObjectRef` is returned to the caller of `HIObjectCreate`.

When someone has called `CFRelease` enough such that the reference count of the object drops to zero, the object is destroyed. The HIToolbox will send a `kEventHIObjectDestruct` event to your object. Do not call `CallNextEventHandler`. Just clean up and return from your handler.

For more information about HIObjects and the HIView subclass, see *HIView Programming Guide*.

Functions by Task

Registering and Creating HIObjects

`HIObjectRegisterSubclass` (page 2335)
Registers an HIObject subclass.

`HIObjectCreate` (page 2330)
Creates an object derived from HIObject.

`HIObjectCreateFromBundle` (page 2331)
Obtains the HIObject for the given bundle.

`HIObjectUnregisterClass` (page 2340)
Unregisters a previously registered subclass of HIObject.

HIObject Utility Functions

`HIObjectCopyClassID` (page 2329)
Obtains the class ID of a given HIObject.

`HIObjectIsOfClass` (page 2333)
Determines whether an object is of a certain class.

`HIObjectDynamicCast` (page 2331)
Obtains the instance data for a specific class of an HIObject.

`HIObjectGetEventTarget` (page 2332)
Obtains the event target of an `HIObjectRef`.

Accessibility Functions

`HIObjectSetAccessibilityIgnored` (page 2337)
Marks an HIObject as ignored (or not) for the purposes of the accessibility APIs.

`HIObjectIsAccessibilityIgnored` (page 2332)
Reports whether the given HIObject is marked as ignored for accessibility.

`HIObjectSetAuxiliaryAccessibilityAttribute` (page 2338)
Associates an additional accessibility attribute with an HIObject.

`HIObjectOverrideAccessibilityContainment` (page 2334)
Overrides the AXUIElement references supplied by an HIObject.

Archiving Functions

`HIObjectIsArchivingIgnored` (page 2333)
Obtains a Boolean value indicating whether an HIObject is marked as ignored for archiving.

`HIObjectSetArchivingIgnored` (page 2337)
Changes the state of archiving for an HIObject.

`HIObjectCopyCustomArchiveData` (page 2330)
Copies custom archive data that is associated with an HIObject.

`HIObjectSetCustomArchiveData` (page 2339)
Associates custom archive data with an HIObject.

Miscellaneous Functions

`HIObjectPrintDebugInfo` (page 2335)
Prints the internal information of an HIObject for debugging purposes.

Functions

HIObjectCopyClassID

Obtains the class ID of a given HIObject.

```
CFStringRef HIObjectCopyClassID (
   HIObjectRef inObject
);
```

Parameters

inObject
The object whose class ID you want.

Return Value
A reference to the object's class ID.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`HIObject.h`

HIObjectCopyCustomArchiveData

Copies custom archive data that is associated with an HIObject.

```
OSStatus HIObjectCopyCustomArchiveData (
   HIObjectRef inObject,
   CFDictionaryRef *outCustomData
);
```

Parameters

inObject
The HIObject whose custom archive data you want to retrieve.

outCustomData
On return, a pointer to the custom data, or `NULL` if no custom archive data is associated with the specified object. The caller is responsible for releasing the dictionary when it is no longer needed.

Return Value
An operating system result code.

Discussion
This function would be used by an archive editor to get custom archive data associated with an HIObject so that the data can be edited.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`HIObject.h`

HIObjectCreate

Creates an object derived from HIObject.

```
OSStatus HIObjectCreate (
   CFStringRef inClassID,
   EventRef inConstructData,
   HIObjectRef *outObject
);
```

Parameters

inClassID
The class ID of the class of object you want to instantiate.

inConstructData

If your class (or any class you derive from) accepts creation parameters, you need to pass an event into this parameter. The class must be `kEventClassHIObject`, and the kind should be `kEventHIObjectInitialize`. Any other parameters should be added as necessary. Specific subclasses of HIObject which require initialization parameters will specify those parameters in the appropriate headers.

outObject

The instance of the object you create.

Return Value
A result code. See "HIObject Result Codes" (page 2345).

Availability
Available in Mac OS X v10.2 and later.

Declared In
`HIObject.h`

HIObjectCreateFromBundle

Obtains the HIObject for the given bundle.

```
OSStatus HIObjectCreateFromBundle (
   CFBundleRef inBundle,
   HIObjectRef *outObject
);
```

Parameters

inBundle

The bundle with which you want to communicate.

outObject

The HIObject associated with the bundle.

Return Value
A result code. See "HIObject Result Codes" (page 2345). If the bundle's HIObject creation function cannot be found, `cfragNoSymbolErr` will be returned.

Discussion
A bundle can be designed to communicate with an application through an HIObject. The bundle must be designed to create an HIObject and have a defined suite of Carbon Events that clients can use to communicate with the bundle's HIObject. Given a CFBundleRef, this API will tell the bundle to create the HIObject and return it to the caller.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`HIObject.h`

HIObjectDynamicCast

Obtains the instance data for a specific class of an HIObject.

```
void * HIObjectDynamicCast (
   HIObjectRef inObject,
   CFStringRef inClassID
);
```

Parameters

inObject

The object whose class ID you want to check.

inClassID

The class ID to get the instance data for.

Return Value

A void * result containing the instance data for the object, or `NULL` if the object is not an instance of the class.

Discussion

The instance data returned is the same instance data the class's construction event handler returns in the instance data parameter. This is stored off with the class reference so that it can be fetched later for use by this function. It allows your subclass to easily get at the data it created, if your subclass needs that data outside of an event handler. (Inside an event handler, your subclass can get at its instance data via the userData parameter to the event handler.)

Availability

Available in Mac OS X v10.2 and later.

Declared In

`HIObject.h`

HIObjectGetEventTarget

Obtains the event target of an `HIObjectRef`.

```
EventTargetRef HIObjectGetEventTarget (
   HIObjectRef inObject
);
```

Parameters

inObject

The object whose target you want.

Return Value

An `EventTargetRef`.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`HIObject.h`

HIObjectIsAccessibilityIgnored

Reports whether the given HIObject is marked as ignored for accessibility.

```
Boolean HIObjectIsAccessibilityIgnored (
   HIObjectRef inObject
);
```

Parameters

inObject

The object whose accessibility ignored state you want to query.

Return Value

A Boolean whose value is `true` if the object is marked as ignored or `false` if the object is not marked as ignored.

Discussion

See the discussion of `HIObjectSetAccessibilityIgnored` (page 2337) for details on what it means to be accessibility ignored.

Availability

Available in Mac OS X v10.2 and later.

Not available to 64-bit applications.

Declared In

`HIAccessibility.h`

HIObjectIsArchivingIgnored

Obtains a Boolean value indicating whether an HIObject is marked as ignored for archiving.

```
Boolean HIObjectIsArchivingIgnored (
   HIObjectRef inObject
);
```

Parameters

inObject

The HIObject whose archiving-ignored state is to be queried.

Return Value

A Boolean whose value is `true` if the specified HIObject is ignored for archiving; otherwise, `false`.

Discussion

By default, HIObjects are marked as ignored for archiving, which indicates that the HIObject does not support the archiving protocol.

Availability

Available in Mac OS X v10.4.

Declared In

`HIObject.h`

HIObjectIsOfClass

Determines whether an object is of a certain class.

```
Boolean HIObjectIsOfClass (
   HIObjectRef inObject,
   CFStringRef inObjectClassID
);
```

Parameters

inObject

The object whose class ID you want to check.

inObjectClassID

The class ID in question.

Return Value
A Boolean whose value is `true` if the object is of the specified class; otherwise, `false`.

Discussion
You can use this to see whether an object you have derives from an expected superclass.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`HIObject.h`

HIObjectOverrideAccessibilityContainment

Overrides the AXUIElement references supplied by an HIObject.

```
OSStatus HIObjectOverrideAccessibilityContainment (
   HIObjectRef inHIObject,
   AXUIElementRef inDesiredParent,
   AXUIElementRef inDesiredWindow,
   AXUIElementRef inDesiredTopLevelUIElement
);
```

Parameters

inHIObject

The object whose parent attribute you want to override.

inDesiredParent

The UI element value you want the HIObject to supply for the parent attribute. This function makes a copy of the `AXUIElementRef`, so you must release `inDesiredParent` when this function returns. Pass `NULL` if you want the specified HIObject to supply its normal parent.

inDesiredWindow

The UI element value you want the HIObject to supply for the window attribute. This function makes a copy of the `AXUIElementRef`, so you must release this parameter when this function returns. Pass `NULL` if you want the specified HIObject to supply its normal window, if any.

inDesiredTopLevelUIElement

The UI element you want the HIObject to supply for the top-level element attribute. This function makes a copy of the `AXUIElementRef`, so you must release this parameter when this function returns. Passing `NULL` indicates that you want the HIObject to supply is normal top-level element, if any.

Return Value
An operating system result code.

Discussion
Use this function to change the parent that an HIObject would normally supply in the accessibility hierarchy. For example, a pop-up control would call this function on its menu so that the menu supplies the pop-up control as the menu's parent instead of the application, which would normally be supplied as the menu's parent. You can also use this function to change the window and the top-level element that an HIObject would normally supply.

If the specified HIObject is a standard HIToolbox construct, for example, an HIView or a menu, the specified HIObject is not added as an accessibility child of its normal parent. If it is not a standard construct, the caller is responsible for ensuring that the specified HIObject is not added as an accessibility child of its normal parent.

If the desired `AXUIElementRef` parent represents an HIView, menu, or window, the specified HIObject is automatically added as an accessibility child of the specified parent. In all other cases, it is the caller's responsibility to add the specified HIObject manually as a child of the specified parent. To represent an HIView, menu, or window, an `AXUIElementRef` object must contain the appropriate `HIObjectRef` value and an identifier value of `0`.

Currently, containment overrides are only supported by HIObjects that are of type HIView, Menu, or Window.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`HIAccessibility.h`

HIObjectPrintDebugInfo

Prints the internal information of an HIObject for debugging purposes.

```
void HIObjectPrintDebugInfo (
   HIObjectRef inObject
);
```

Parameters
`inObject`
The object to inspect.

Discussion
This function sends the information to `stdout`.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`HIObject.h`

HIObjectRegisterSubclass

Registers an HIObject subclass.

```
OSStatus HIObjectRegisterSubclass (
   CFStringRef inClassID,
   CFStringRef inBaseClassID,
   OptionBits inOptions,
   EventHandlerUPP inConstructProc,
   ItemCount inNumEvents,
   const EventTypeSpec *inEventList,
   void *inConstructData,
   HIObjectClassRef *outClassRef
);
```

Parameters

inClassID

The class ID of your class. It should be unique. We recommend using Java-style com.company.foo naming conventions to avoid collisions.

inBaseClassID

The class ID of the class you derive from. Passing `NULL` indicates you want to subclass HIObject (the base class) directly.

inOptions

Any special options for your class. Currently you must pass `0` for this parameter.

inConstructProc

A universal procedure pointer to the event handler for this subclass. You pass the address of an event handler into this parameter. This handler is called directly, rather than through the normal event-dispatching mechanism. This means that the EventHandlerCallRef passed in will be `NULL`, and you cannot use it for calls like `CallNextEventHandler`. Other than that, you should return a result as usual. After your object is constructed, this procedure is installed as the event handler for the remaining events specified in the `inEventList` parameter. In Mac OS X v10.4 and later, passing `NULL` creates an "abstract class" that cannot be instantiated but can still be used as a base class for subclasses. If you pass `NULL`, `HIObjectCreate` on the class ID will return `hiObjectClassIsAbstractErr`.

inNumEvents

The number of events you are installing.

inEventList

The events your handler wants to receive. You must handle the `kEventHIObjectConstruct` and `kEventHIObjectDestruct` event. If these events are not specified, an error is returned.

inConstructData

Pass any info you want passed into your event handler here. For a C++ hierarchy based on HIObjects, you might actually pass a static method to construct your object here, and the base class event handler to do construction as your event handler.

outClassRef

The newly created class reference. Pass `NULL` if you don't care.

Return Value

A result code. See "HIObject Result Codes" (page 2345).

Availability

Available in Mac OS X version 10.2 (v10.2) and later.

Declared In

`HIObject.h`

HIObjectSetAccessibilityIgnored

Marks an HIObject as ignored (or not) for the purposes of the accessibility APIs.

```
OSStatus HIObjectSetAccessibilityIgnored (
   HIObjectRef inObject,
   Boolean inIgnored
);
```

Parameters

inObject

The object whose accessibility ignored state you want to change.

inIgnored

A Boolean whose value is `true` to mark the object as ignored or `false` to mark the object as not ignored.

Return Value

An operating system result code.

Discussion

An HIObject that is ignored for accessibility will never be shown to an assistive application that uses the accessibility APIs to examine an interface. Your application's accessibility implementation can (and should) still report an ignored HIObject as usual. Carbon's accessibility engine will automatically prune any ignored HIObjects out of the data that is shown to an assistive application.

By default, an HIObject is *not* accessibility-ignored.

Availability

Available in Mac OS X v10.2 and later.

Not available to 64-bit applications.

Declared In

`HIAccessibility.h`

HIObjectSetArchivingIgnored

Changes the state of archiving for an HIObject.

```
OSStatus HIObjectSetArchivingIgnored (
   HIObjectRef inObject,
   Boolean inIgnored
);
```

Parameters

inObject

The HIObject whose archiving ignored state is to be changed.

inIgnored

A Boolean whose value is `true` to indicate that the specified HIObject does not support the archiving protocol and should be ignored; otherwise, `false`.

Return Value

An operating system result code.

Discussion
Call this function to mark or unmark an HIObject as ignored for archiving. By default, HIObjects are marked as ignored for archiving. HIObject subclasses supporting archiving with the `kEventHIObjectInitialize` and `kEventHIObjectEncode` events must set their archiving ignored state to `false` in order to receive archiving requests from clients. A client may still reset the archive ignored state to `true` on a particular object. An HIObject marked as ignored for archiving will never be requested to encode itself into an archive.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`HIObject.h`

HIObjectSetAuxiliaryAccessibilityAttribute

Associates an additional accessibility attribute with an HIObject.

```
OSStatus HIObjectSetAuxiliaryAccessibilityAttribute (
   HIObjectRef inHIObject,
   UInt64 inIdentifier,
   CFStringRef inAttributeName,
   CFTypeRef inAttributeData
);
```

Parameters

inHIObject
The part of the object-identifier pair that is to be associated with attribute data.

inIdentifier
The part of the object-identifier pair with which the attribute data is to be associated. Pass `0` if you want to associate the attribute data to the HIObject as a whole. For example, pass `0` if you want to associate a description attribute to a push button.

inAttributeName
The name of the attribute that is to be associated with the object-identifier pair. This string is retained before it is added to the auxiliary attribute store.

inAttributeData
The data that is to become the attribute's value. This data is retained before it is added to the auxiliary attribute store. You should release `inAttributeData` after `HIObjectSetAuxiliaryAccessibilityAttribute` returns. Pass `NULL` to indicate that the named auxiliary attribute should no longer be associated with the object-identifier pair. When you pass `NULL`, any named attribute data that was previously associated with the object-identifier pair is released.

Return Value
An operating system result code.

Discussion
Use this function to provide the name and data for an accessibility attribute you want to add to the UIElement representing an HIObject-identifier pair. Normally, Carbon events are used to supply accessibility attributes dynamically, but `HIObjectSetAuxiliaryAccessibilityAttribute` allows you to supply them statically.

This function is particularly useful for supplying values for the `kAXDescriptionAttribute`, `kAXTitleUIElementAttribute`, `kAXServesAsTitleUIElementAttribute`, `kAXLinkedUIElementsAttribute`, and other attributes whose values are specific to the layout and usage

of your application. Note, however, that this function can associate only attributes whose values do not change. If you need to supply attributes whose values are determined dynamically or whose values can be set, you must install the normal accessibility Carbon event handlers for normal accessibility.

When an accessibility attribute Carbon event is handled by the HIObject with a given identifier, the HIToolbox automatically supplies the names and values of any auxiliary attributes associated with the HIObject-identifier pair.

The auxiliary attribute store is consulted during the HIObject's default handling of accessibility attribute Carbon events. This means that any programmatic handling of a given accessibility attribute is able to override or block consultation of the store. In general, if the HIToolbox or a Carbon event handler can provide the attribute value in some other way, the store is not consulted.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`HIAccessibility.h`

HIObjectSetCustomArchiveData

Associates custom archive data with an HIObject.

```
OSStatus HIObjectSetCustomArchiveData (
   HIObjectRef inObject,
   CFDictionaryRef inCustomData
);
```

Parameters

inObject
The HIObject with which custom archive data is to be associated.

inCustomData
A `CFDictionaryRef` containing the custom archive data that is to be associated with the specified HIObject. Associating the custom archive data replaces any data that was previously associated. To archive, the dictionary's keys and values must use CFType callbacks. Pass `NULL` to clear any custom archive data that was previously associated.

Return Value
An operating system result code.

Discussion
This function might be used by an archive editor to associate custom archive data that it has edited with an HIObject.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`HIObject.h`

HIObjectUnregisterClass

Unregisters a previously registered subclass of HIObject.

```
OSStatus HIObjectUnregisterClass (
   HIObjectClassRef inClassRef
);
```

Parameters

inClassRef

The class reference of the object class you want to unregister.

Discussion

You receive an error if there are subclasses of your class or existing instances of it. All instances and subclasses must be disposed of and unregistered first.

Availability

Available in Mac OS X v10.2 and later.

Declared In

HIObject.h

Constants

Standard Custom Archive Data Dictionary Keys for Custom Initialize Events

Define standard custom archive dictionary keys for custom initialize events.

```
const CFStringRef kHIObjectCustomDataParameterNamesKey;
const CFStringRef kHIObjectCustomDataParameterTypesKey;
const CFStringRef kHIObjectCustomDataParameterValuesKey;
```

Constants

`kHIObjectCustomDataParameterNamesKey`

The value of this key is an array of strings. Each `CFStringRef` contains an `OSType` that is a Carbon event parameter name.

`kHIObjectCustomDataParameterTypesKey`

The value of this key is an array of strings. Each `CFStringRef` contains an `OSType` that is a Carbon event parameter type.

`kHIObjectCustomDataParameterValuesKey`

The value of this key is an array of strings. Each `CFStringRef` contains a representation of the value.

Discussion

The value for each of these constants is a `CFArrayRef` containing CFStringRefs. For a given dictionary, the names, types, and values arrays should each have the same number of CFStrings. The name-type-value triple at the given index in each array represents a Carbon event parameter in the initialization event for the HIObject.

The current supported set of Carbon event parameter types consists of `'cfst'`, `'TEXT'`, `'bool'`, `'cfrn'`, `'doub'`, `'osst'`, `'long'`, `'magn'`, `'hipt'`, `'hisz'`, and `'hirc'`.

Availability

Available in Mac OS v10.4 and later.

Standard Custom Archive Data Dictionary Class and SuperClass Keys

Define standard custom archive data dictionary keys for classes and superclasses.

```
const CFStringRef kHIObjectCustomDataClassIDKey;
const CFStringRef kHIObjectCustomDataSuperClassIDKey;
```

Constants

`kHIObjectCustomDataClassIDKey`
The class ID.

`kHIObjectCustomDataSuperClassIDKey`
The super-class ID key.

Discussion

These keys define a class and superclass for clients that do not implement the object's true class. Each keyed value is an HIObject class ID.

Availability

Available in Mac OS v10.4 and later.

Standard Custom Archive Data Dictionary Key for ProcPointer-Based CDEFs

Define a standard custom archive data dictionary key for ProcPointer-based CDEFs.

```
const CFStringRef kHIObjectCustomDataCDEFProcIDKey;
```

Constants

`kHIObjectCustomDataCDEFProcIDKey`
The standard custom archive data dictionary key for ProcPointer-based CDEFs.

Discussion

The key value is a `CFStringRef`-based signed 16-bit integer. Use `CFStringGetIntValue` in `CFString.h` to convert `CFStringRef` to `SInt16` and to convert `SInt16` to `CFStringRef`.

Availability

Available in Mac OS v10.4 and later.

HIObject Base Class Events

Define the base-class functionality of HIObjects.

```
enum{
    kEventClassHIObject = 'hiob',
    kEventHIObjectConstruct = 1,
    kEventHIObjectInitialize = 2,
    kEventHIObjectDestruct = 3,
    kEventHIObjectIsEqual = 4,
    kEventHIObjectPrintDebugInfo = 5,
    kEventHIObjectEncode = 6
};
```

Constants

`kEventClassHIObject`

The event class for HIObject events.

Available in Mac OS X v10.2 and later.

Declared in `HIObject.h`.

`kEventHIObjectConstruct`

Your object is being constructed. When your event handler is called with this event, it is being called directly and not through the normal event dispatching mechanism. This means that the `EventHandlerCallRef` passed to your handler is `NULL` and `CallNextEventHandler` does not work. You are passed the actual `HIObjectRef` of your base class for you to record in your instance data. (Available in Mac OS X v10.2 and later.)

Available in Mac OS X v10.2 and later.

Declared in `HIObject.h`.

`kEventHIObjectInitialize`

Your object is being initialized. Your handler should pass this onto the superclass first before handling this event. This is done by calling `CallNextEventHandler` with the event. When that function returns, you should make sure the result is `noErr`. If not, you should not continue to initialize your object. (Available in Mac OS X v10.2 and later.)

Available in Mac OS X v10.2 and later.

Declared in `HIObject.h`.

`kEventHIObjectDestruct`

Your object is being destroyed. This is your chance to dispose of anything you might have allocated for your object. Do not call through with `CallNextEventHandler`. (Available in Mac OS X v10.2 and later.)

Available in Mac OS X v10.2 and later.

Declared in `HIObject.h`.

`kEventHIObjectIsEqual`

`HIObjectIsEqual` has been called, and you are being asked to determine whether your object is equivalent to the one being passed to your handler. If your object is equivalent, you should place `true` in the `kEventParamResult` parameter in the event; if your object is not equivalent, place `false` in the `kEventParamResult` parameter. (Available in Mac OS X v10.2 and later.)

Available in Mac OS X v10.2 and later.

Declared in `HIObject.h`.

`kEventHIObjectPrintDebugInfo`
`HIObjectPrintDebugInfo` has been called, and you are being asked to print your information to stdout. This event is sent to all handlers and you should not call `CallNextEventHandler`. (Available in Mac OS X v10.2 and later.)

Available in Mac OS X v10.2 and later.

Declared in `HIObject.h`.

`kEventHIObjectEncode`
`HIArchiveEncodeHIObject` has been called on your HIObject, and you are being asked to encode your object into an archive. Before handling this event, your handler should pass this event to the superclass by calling `CallNextEventHandler` with the event. If `CallNextEventHandler` does not return `noErr`, you should not continue to encode your instance data. (Available in Mac OS X v10.4 and later.)

Available in Mac OS X v10.4 and later.

Declared in `HIObject.h`.

Discussion
You only need to be aware of these events if you are implementing a subclass.

Table 31-1 Parameter names and types for HIObject base class events

Event kind	Parameter name	Parameter type
`kEventHIObjectConstruct`	`kEventParamHIObjectInstance`	`typeHIObjectRef`
`kEventHIObjectInitialize`	`kEventParamHIArchive`	`typeCFTypeRef`
`kEventHIObjectDestruct`		
`kEventHIObjectIsEqual`	`kEventParamDirectObject`	`typeHIObjectRef`
	`kEventParamResult`	`typeBoolean`
`kEventHIObjectPrintDebugInfo`		
`kEventHIObjectEncode`	`kEventParamHIArchive`	`typeCFTypeRef`

HIObject Base Class Event Parameters

Define constants for HIObject base class event parameters.

```
enum {
    kEventParamHIObjectInstance = 'hioi',
    kEventParamHIArchive = 'hiac',
    typeHIObjectRef = 'hiob'
};
```

Constants

`kEventParamHIObjectInstance`

On entry, the HIObject reference for your object. This parameter is currently only used for the `kEventHIObjectConstruct` event. Typically, you read this parameter from the event and store it in your instance data so that when your instance needs to call HIObject APIs, your instance can use this `HIObjectRef`. On exit, the value of this parameter is `typeVoidPtr` and is a pointer to the instance data you have written into the event using `SetEventParameter`. After your event handler returns, the HIToolbox reads your instance data pointer from the event, and installs the event handlers that were passed to `HIObjectRegisterClassSubclass` on the new object. The HIToolbox uses the instance data pointer as the refcon for the event handlers it installed so that your event handlers can retrieve your instance data pointer from the reference constant (the third parameter to an `EventHandlerProcPtr`).(Available in Mac OS X v10.2 and later.)

Available in Mac OS X v10.2 and later.

Declared in `HIObject.h`.

`kEventParamHIArchive`

An HIArchive used to store or retrieve the HIObject. This parameter is passed in the following cases:

- In the `kEventHIObjectInitialize` event when the HIObject is requested to initialize itself from a decoded archive. If the HIObject is to be initialized normally (that is, not from an archive), the initialize event does not contain `kEventParamHIArchive` .
- In the `kEventHIObjectEncode` event, which is sent to request that the HIObject encode itself within an archive.

(Available in Mac OS X v10.4 and later.)

Available in Mac OS X v10.4 and later.

Declared in `HIObject.h`.

Result Codes

Result Code	Value	Description
`hiObjectClassExistsErr`	-22080	You are trying to register a class ID that already exists. Available in Mac OS X v10.2 and later.
`hiObjectClassHasInstancesErr`	-22081	You are trying to unregister a class which has instances that still exist. Available in Mac OS X v10.2 and later.
`hiObjectClassHasSubclassesErr`	-22082	You are trying to unregister a class which has subclasses registered. They must be unregistered before this class can be unregistered. Available in Mac OS X v10.2 and later.

Result Code	Value	Description
`hiObjectClassIsAbstractErr`	-22083	You are trying to create an HIObject class that is defined as being abstract. You must subclass it instead. Available in Mac OS X v10.2 and later.

HIShape Reference

Framework: Carbon/Carbon.h

Declared in HIShape.h

Overview

HIShape is an abstract shape object that replaces the old QuickDraw region handle. HIShape objects are the preferred way to describe regions in HIView views that use Quartz.

One advantage of HIShape objects is that they can be mutable or immutable. Using immutable (that is, fixed) shapes improves performance because copying it simply requires incrementing the reference count, avoiding the handle-to-handle copies required with the older region handles.

Currently, HIShape objects cannot handle floating point–based shape descriptions. Therefore, any coordinates you specify when defining shapes must end on integer boundaries. The purpose of the shape often dictates whether rounding fractional values up or down will give you the best results. For example, shapes that describe a maximum alllowable area, such as the structure region of a window, should be rounded up (or outward) to the nearest integer. Shapes describing a minimum allowable area, such as the opaque region, should be rounded down (or inward). By default, any fractional coordinate values are truncated.

Typical shapes are rectangular, but you can build more complex shapes by combining them with other shapes using the union, intersection, or difference functions. The Quartz-savvy HITheme APIs also let you obtain shapes for various standard control elements.

In most cases, applications will obtain or set shapes in response to a Carbon event using `GetEventParameter` or `SetEventParameter` with the `typeHIShapeRef` parameter type.

The following functions also take HIShape references as parameters:

- `HIViewNewTrackingArea` and `HIViewChangeTrackingArea`, which let you set or change an HIShape-based tracking area in a view (comparable to window tracking regions).
- `HIViewSetNeedsDisplayInShape`, which is the HIShape-compatible version of `HIViewSetNeedsDisplayInRegion`.

Except where noted, HIShape functions return only the result codes `noErr` or `paramErr`.

HIShape objects are available in Mac OS X version 10.2 and later. However, many of the functions that take HIShape parameters are available only in Mac OS X version 10.4 and later.

Functions by Task

Creating Immutable Shapes

`HIShapeCreateWithQDRgn` (page 2352)
Creates an immutable shape from a QuickDraw region.

`HIShapeCreateWithRect` (page 2353)
Creates an immutable shape from a Quartz rectangle.

`HIShapeCreateEmpty` (page 2350)
Creates an empty immutable shape.

`HIShapeCreateCopy` (page 2350)
Creates an immutable copy of a mutable or immutable shape.

`HIShapeCreateIntersection` (page 2351)
Creates an immutable shape that is the intersection of two shapes.

`HIShapeCreateDifference` (page 2350)
Creates an immutable shape that is the difference of two shapes.

`HIShapeCreateUnion` (page 2352) Deprecated in Mac OS X v10.4
Creates an immutable shape that is the union of two shapes.

Creating Mutable Shapes

`HIShapeCreateMutable` (page 2351)
Creates an empty mutable shape.

`HIShapeCreateMutableCopy` (page 2352)
Creates a mutable copy of an existing shape.

`HIShapeSetEmpty` (page 2358)
Sets the specified mutable shape to be empty.

`HIShapeDifference` (page 2353)
Sets a shape to be the difference of two other shapes.

`HIShapeUnion` (page 2359)
Sets a shape to be the union of two other shapes.

`HIShapeOffset` (page 2357)
Moves a shape by a specified offset.

`HIShapeIntersect` (page 2355) Deprecated in Mac OS X v10.4
Sets a shape to be the intersection of two other shapes.

Querying Shapes

`HIShapeIsEmpty` (page 2356)
Determines whether a given shape is empty.

`HIShapeIsRectangular` (page 2357)
Determines whether a given shape is rectangular.

`HIShapeContainsPoint` (page 2349)
Determines whether a shape contains the given point.

Manipulating Shapes

`HIShapeGetTypeID` (page 2355)
Obtains the Core Foundation type ID for the HIShape class.

`HIShapeGetBounds` (page 2355)
Obtains the bounding rectangle for the given shape.

`HIShapeIntersectsRect` (page 2356)
Determines whether a given shape intersects a given rectangle.

`HIShapeGetAsQDRgn` (page 2354)
Creates a QuickDraw region with the same shape as the specified `HIShape`.

`HIShapeReplacePathInCGContext` (page 2358)
Make the current path in a graphics context match a shape.

`HIShapeSetQDClip` (page 2359)
Sets the clip region in the current QuickDraw port to be the specified shape.

Functions

HIShapeContainsPoint

Determines whether a shape contains the given point.

```
Boolean HIShapeContainsPoint (
   HIShapeRef inShape,
   const CGPoint *inPoint
);
```

Parameters

inShape
The shape to test.

inPoint
The point to test.

Return Value
`True` if the shape contains the point, `False` otherwise.

Discussion
This function is comparable to the old QuickDraw function `PtInRgn`.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`HIShape.h`

HIShapeCreateCopy

Creates an immutable copy of a mutable or immutable shape.

```
HIShapeRef HIShapeCreateCopy (
   HIShapeRef inShape
);
```

Parameters

inShape
The shape you want to copy.

Return Value
The newly-created copy.

Discussion
Copying an immutable shape simply increments its reference count. Copying a mutable shape actually duplicates it in memory.

Availability
Available in Mac OS X v10.2 and later.

Declared In
HIShape.h

HIShapeCreateDifference

Creates an immutable shape that is the difference of two shapes.

```
HIShapeRef HIShapeCreateDifference (
   HIShapeRef inShape1,
   HIShapeRef inShape2
);
```

Parameters

inShape1
The first shape.

inShape2
The shape to subtract from the first shape.

Return Value
The newly-created difference shape.

Availability
Available in Mac OS X v10.2 and later.

Declared In
HIShape.h

HIShapeCreateEmpty

Creates an empty immutable shape.

```
HIShapeRef HIShapeCreateEmpty (
   void
);
```

Return Value
The newly-created empty shape.

Availability
Available in Mac OS X v10.4 and later.

Declared In
HIShape.h

HIShapeCreateIntersection

Creates an immutable shape that is the intersection of two shapes.

```
HIShapeRef HIShapeCreateIntersection (
   HIShapeRef inShape1,
   HIShapeRef inShape2
);
```

Parameters

inShape1
An existing shape.

inShape2
Another existing shape.

Return Value
The newly-created intersection shape.

Discussion
If the two shapes do not intersect, the shape returned is empty.

Availability
Available in Mac OS X v10.2 and later.

Declared In
HIShape.h

HIShapeCreateMutable

Creates an empty mutable shape.

```
HIMutableShapeRef HIShapeCreateMutable (
   void
);
```

Return Value
An empty, mutable shape.

Availability
Available in Mac OS X v10.2 and later.

Declared In
HIShape.h

HIShapeCreateMutableCopy

Creates a mutable copy of an existing shape.

```
HIMutableShapeRef HIShapeCreateMutableCopy (
   HIShapeRef inOrig
);
```

Parameters

inOrig
The shape to copy.

Return Value
The newly-created copy.

Availability
Available in Mac OS X v10.2 and later.

Declared In
HIShape.h

HIShapeCreateUnion

Creates an immutable shape that is the union of two shapes.

```
HIShapeRef HIShapeCreateUnion (
   HIShapeRef inShape1,
   HIShapeRef inShape2
);
```

Parameters

inShape1
An existing shape.

inShape2
Another existing shape.

Return Value
The newly-created union shape.

Availability
Available in Mac OS X v10.2 and later.

Declared In
HIShape.h

HIShapeCreateWithQDRgn

Creates an immutable shape from a QuickDraw region.

```
HIShapeRef HIShapeCreateWithQDRgn (
   RgnHandle inRgn
);
```

Parameters

inRgn

The region to convert to an immutable shape.

Return Value
The newly created shape.

Discussion
You can use this function to convert handle-based regions to HIShape objects. However, you should be judicious about how often you convert, as this function does require memory allocation.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`HIShape.h`

HIShapeCreateWithRect

Creates an immutable shape from a Quartz rectangle.

```
HIShapeRef HIShapeCreateWithRect (
   const CGRect *inRect
);
```

Parameters

inRect

The Quartz-based rectangle to convert.

Return Value
The newly-created shape.

Discussion
This function is comparable to the old QuickDraw function `RectRgn`.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`HIShape.h`

HIShapeDifference

Sets a shape to be the difference of two other shapes.

```
OSStatus HIShapeDifference (
   HIShapeRef inShape1,
   HIShapeRef inShape2,
   HIMutableShapeRef outResult
);
```

Parameters

inShape1

The first shape.

inShape2

The shape to subtract from the first.

outResult

The shape to hold the difference of the two shapes. This parameter can be one of the source shapes.

Return Value
A result code.

Discussion
This function is comparable to the old QuickDraw function `DiffRgn`.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`HIShape.h`

HIShapeGetAsQDRgn

Creates a QuickDraw region with the same shape as the specified `HIShape`.

```
OSStatus HIShapeGetAsQDRgn (
   HIShapeRef inShape,
   RgnHandle outRgn
);
```

Parameters

inShape

The shape to convert.

outRgn

A valid region handle. You must obtain this handle by calling `NewRgn` before calling this function.

Return Value
A result code.

Discussion
You can use this function to create a handle-based region from an HIShape object. This conversion requires memory allocation, so you should convert only when necessary.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`HIShape.h`

HIShapeGetBounds

Obtains the bounding rectangle for the given shape.

```
CGRect * HIShapeGetBounds (
   HIShapeRef inShape,
   CGRect *outRect
);
```

Parameters

inShape

The shape whose bounds you want to obtain.

inRect

A pointer to the `HIRect` structure you want filled with the shape bounds.

Return Value

A pointer to the rectangle you passed in the `inRect` parameter, now set to the shape's bounding rectangle.

Discussion

The function result is a pointer to the `HIRect` structure you passed in the `inRect` parameter.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`HIShape.h`

HIShapeGetTypeID

Obtains the Core Foundation type ID for the HIShape class.

```
CFTypeID HIShapeGetTypeID (
   void
);
```

Return Value

The type ID for the HIShape class.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`HIShape.h`

HIShapeIntersect

Sets a shape to be the intersection of two other shapes.

```
OSStatus HIShapeIntersect (
   HIShapeRef inShape1,
   HIShapeRef inShape2,
   HIMutableShapeRef outResult
);
```

Parameters

inShape1

The first shape.

inShape2

The second shape.

outResult

The shape to hold the intersection of the two shapes. This parameter can be one of the source shapes.

Return Value
A result code.

Discussion
This function is comparable to the old QuickDraw function `SectRgn`.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`HIShape.h`

HIShapeIntersectsRect

Determines whether a given shape intersects a given rectangle.

```
Boolean HIShapeIntersectsRect (
   HIShapeRef inShape,
   const CGRect *inRect
);
```

Parameters

inShape

The shape to test.

inRect

The rectangle to test against.

Return Value
Returns `True` if the area of the shape intersects the rectangle, `False` otherwise.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`HIShape.h`

HIShapeIsEmpty

Determines whether a given shape is empty.

```
Boolean HIShapeIsEmpty (
   HIShapeRef inShape
);
```

Parameters

inShape

The shape to test.

Return Value

`True` if the shape's area is empty, `False` otherwise.

Discussion

This function is comparable to the old QuickDraw function `EmptyRgn`.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`HIShape.h`

HIShapeIsRectangular

Determines whether a given shape is rectangular.

```
Boolean HIShapeIsRectangular (
   HIShapeRef inShape
);
```

Parameters

inShape

The shape to test.

Return Value

`True` if the shape's area is rectangular, `False` otherwise.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`HIShape.h`

HIShapeOffset

Moves a shape by a specified offset.

```
OSStatus HIShapeOffset (
   HIMutableShapeRef inShape,
   CGFloat inDX,
   CGFloat inDY
);
```

Parameters

inShape

The shape to offset.

inDx
The desired x-coordinate offset.

inDy
The desired y-coordinate offset.

Return Value
A result code.

Discussion
This function is comparable to the old QuickDraw function `OffsetRgn`.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`HIShape.h`

HIShapeReplacePathInCGContext

Make the current path in a graphics context match a shape.

```
OSStatus HIShapeReplacePathInCGContext (
   HIShapeRef inShape,
   CGContextRef inContext
);
```

Parameters

inShape
The shape to apply to the path.

inContext
The context to apply the shape to.

Return Value
A result code.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`HIShape.h`

HIShapeSetEmpty

Sets the specified mutable shape to be empty.

```
OSStatus HIShapeSetEmpty (
   HIMutableShapeRef inShape
);
```

Parameters

inShape
The shape to empty.

Return Value
A result code.

Discussion
This function is comparable to the old QuickDraw function `SetEmptyRgn`.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`HIShape.h`

HIShapeSetQDClip

Sets the clip region in the current QuickDraw port to be the specified shape.

```
OSStatus HIShapeSetQDClip (
   HIShapeRef inShape,
   CGrafPtr inPort
);
```

Parameters

inShape
The shape to apply to the clip region.

inPort
The clip region to apply the shape to.

Return Value
A result code. If `HIShapeSetQDClip` cannot allocate a QuickDraw region, it returns `memFullErr`.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`HIShape.h`

HIShapeUnion

Sets a shape to be the union of two other shapes.

```
OSStatus HIShapeUnion (
   HIShapeRef inShape1,
   HIShapeRef inShape2,
   HIMutableShapeRef outResult
);
```

Parameters

inShape1
An existing shape.

inShape2
Another existing shape.

outResult
The shape to hold the union of the two shapes. This parameter can be one of the source shapes.

Return Value
A result code.

Discussion
This function is comparable to the old QuickDraw function `UnionRgn`.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`HIShape.h`

Data Types

HIShapeRef

Defines a shape object.

```
typedef const struct __HIShape *HIShapeRef;
```

Discussion
The structure pointed to by this reference is opaque.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`HIShape.h`

HIMutableShapeRef

Defines a mutable shape object.

```
typedef struct __HIShape *HIMutableShapeRef;
```

Discussion
The structure pointed to by this reference is opaque.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`HIShape.h`

HIToolbar Reference

Framework: Carbon/Carbon.h
Declared in HIToolbar.h

Overview

HIToolbar is the Carbon equivalent of the Cocoa toolbar (specifically, the `NSToolbar` and `NSToolbarItem` classes). The toolbar object and the items associated with a toolbar are both subclassed from HIObject. Toolbar items can have HIViews associated with them.

For more information, see Using HIToolbar.

Functions by Task

Creating Toolbars

`HIToolbarCreate` (page 2365)
Creates a toolbar.

Manipulating Toolbars

`HIToolbarChangeAttributes` (page 2364)
Changes the attributes of a toolbar.

`HIToolbarCopyIdentifier` (page 2364)
Obtains the identifier for a toolbar.

`HIToolbarCopyItems` (page 2365)
Obtains the array of toolbar items for a toolbar.

`HIToolbarGetAttributes` (page 2367)
Obtains the attributes for the given toolbar.

`HIToolbarGetDelegate` (page 2367)
Returns the current delegate in use by a toolbar.

`HIToolbarSetDelegate` (page 2382)
Sets the delegate object for a toolbar.

`HIToolbarGetDisplayMode` (page 2367)
Obtains the current display mode of a toolbar.

`HIToolbarSetDisplayMode` (page 2383)
Sets the current display mode of a toolbar.

`HIToolbarGetDisplaySize` (page 2368)
Obtains the current display size of a toolbar.

`HIToolbarSetDisplaySize` (page 2383)
Sets the current display size of a toolbar.

Creating and Adding Toolbar Items

`HIToolbarCreateItemWithIdentifier` (page 2366)
Creates a toolbar item.

`HIToolbarAppendItem` (page 2363)
Appends an item to the toolbar.

`HIToolbarInsertItemAtIndex` (page 2369)
Inserts a toolbar item at a given index into a toolbar.

`HIToolbarRemoveItemAtIndex` (page 2382)
Removes an item at a given index from a toolbar.

`HIToolbarItemCreate` (page 2375)
Creates a toolbar item.

`HIToolbarSetItemsWithIdentifiers` (page 2384)
Sets a toolbar's items all at once.

Manipulating Toolbar Items

`HIToolbarGetSelectedItemInWindow` (page 2368)
Obtains the toolbar item that is selected in a window.

`HIToolbarItemGetToolbar` (page 2377)
Obtains the toolbar associated with a toolbar item.

`HIToolbarItemGetAttributes` (page 2375)
Obtains the attributes of a toolbar item.

`HIToolbarItemGetAttributesInWindow` (page 2376)
Obtains the attributes of a toolbar item in the specified window.

`HIToolbarItemChangeAttributes` (page 2370)
Changes the attributes of a toolbar item.

`HIToolbarItemChangeAttributesInWindow` (page 2370)
Changes the attributes of a toolbar item in a specific window.

`HIToolbarItemCopyHelpText` (page 2372)
Obtains the help tag text for a toolbar item.

`HIToolbarItemSetHelpText` (page 2379)
Sets the help tag text for a toolbar item.

`HIToolbarItemCopyIdentifier` (page 2373)
Obtains the identifier for a given toolbar item.

`HIToolbarItemCopyImage` (page 2373)
Obtains the image for a toolbar item.

`HIToolbarItemSetImage` (page 2380)
Sets the image for a toolbar item.

`HIToolbarItemCopyLabel` (page 2374)
Obtains the label for a toolbar item.

`HIToolbarItemSetLabel` (page 2381)
Sets the label of a toolbar item.

`HIToolbarItemCopyMenu` (page 2374)
Obtains the submenu for a toolbar item.

`HIToolbarItemSetMenu` (page 2381)
Sets the submenu for a toolbar item.

`HIToolbarItemGetCommandID` (page 2377)
Gets the command ID of a toolbar item.

`HIToolbarItemSetCommandID` (page 2378)
Sets the command ID of a toolbar item.

`HIToolbarItemIsEnabled` (page 2378)
Determines if a toolbar item is enabled.

`HIToolbarItemSetEnabled` (page 2378)
Enables or disables a toolbar item.

`HIToolbarItemSetIconRef` (page 2380)
Sets the icon for a toolbar item.

`HIToolbarItemConfigDataChanged` (page 2372)
Tells the toolbar that the configuration for a toolbar item has changed.

Functions

HIToolbarAppendItem

Appends an item to the toolbar.

```
OSStatus HIToolbarAppendItem (
   HIToolbarRef inToolbar,
   HIToolbarItemRef inItem
);
```

Parameters

inToolbar
The toolbar to receive the new item.

inItem
The item reference of the toolbar item you are adding.

Return Value

An operating system result code.

Discussion
This function appends an item to the end of a toolbar. Generally, you should always add items via identifier, and not with this routine.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`HIToolbar.h`

HIToolbarChangeAttributes

Changes the attributes of a toolbar.

```
OSStatus HIToolbarChangeAttributes (
   HIToolbarRef inToolbar,
   OptionBits inAttrsToSet,
   OptionBits inAttrsToClear
);
```

Parameters

inToolbar
The toolbar whose attributes you want to change.

inAttrsToSet
The attributes you want to set. For possible values, see "Toolbar Attributes" (page 2384).

inAttrsToClear
The attributes you want to clear. For possible values, see "Toolbar Attributes" (page 2384).

Return Value
An operating system result code.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`HIToolbar.h`

HIToolbarCopyIdentifier

Obtains the identifier for a toolbar.

```
OSStatus HIToolbarCopyIdentifier (
   HIToolbarRef inToolbar,
   CFStringRef *outIdentifier
);
```

Parameters

inToolbar
The toolbar whose identifier you want to obtain.

outIdentifier
The identifier. You must release it when you are finished with it.

Return Value
An operating system result code.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
HIToolbar.h

HIToolbarCopyItems

Obtains the array of toolbar items for a toolbar.

```
OSStatus HIToolbarCopyItems (
   HIToolbarRef inToolbar,
   CFArrayRef *outItems
);
```

Parameters

inToolbar
The toolbar whose items you want to receive.

outItems
The array of toolbar items owned by the toolbar. You must release the array when you are finished with it.

Return Value
An operating system result code.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
HIToolbar.h

HIToolbarCreate

Creates a toolbar.

```
OSStatus HIToolbarCreate (
   CFStringRef inIdentifier,
   OptionBits inAttributes,
   HIToolbarRef *outToolbar
);
```

Parameters

inIdentifier
The identifier of the toolbar. If you specify that the toolbar auto-saves its configuration, this identifier is used to mark the config info in your application's preferences. You must specify an identifier.

inAttributes
Any attributes you want to set. For possible values, see "Toolbar Attributes" (page 2384).

outToolbar
The toolbar reference to your new toolbar.

Return Value
An operating system result code.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
HIToolbar.h

HIToolbarCreateItemWithIdentifier

Creates a toolbar item.

```
OSStatus HIToolbarCreateItemWithIdentifier (
   HIToolbarRef inToolbar,
   CFStringRef inIdentifier,
   CFTypeRef inConfigData,
   HIToolbarItemRef *outItem
);
```

Parameters

inToolbar
The toolbar you are adding to.

inIdentifier
The identifier of the item you want to add.

inConfigData
Any config data required by the item to safely construct. Standard items do not require any extra data, so `NULL` can be passed.

outItem
The newly created toolbar item.

Return Value
An operating system result code.

Discussion
This function creates an item specified by a particular identifier. Using this function allows you to create any item a delegate supports by naming its identifier. It also allows you to create standard items supplied by the Toolbox, such as the separator item.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
HIToolbar.h

HIToolbarGetAttributes

Obtains the attributes for the given toolbar.

```
OSStatus HIToolbarGetAttributes (
   HIToolbarRef inToolbar,
   OptionBits *outAttributes
);
```

Parameters

inToolbar

The toolbar whose attributes you desire.

outAttributes

The toolbar's attributes. For details, see "Toolbar Attributes" (page 2384).

Return Value

An operating system result code.

Availability

Available in Mac OS X v10.2 and later.

Not available to 64-bit applications.

Declared In

HIToolbar.h

HIToolbarGetDelegate

Returns the current delegate in use by a toolbar.

```
HIObjectRef HIToolbarGetDelegate (
   HIToolbarRef inToolbar
);
```

Parameters

inToolbar

The toolbar you want the delegate from.

Return Value

An HIObjectRef.

Discussion

The delegate handles the event processing for the toolbar.

Availability

Available in Mac OS X v10.2 and later.

Not available to 64-bit applications.

Declared In

HIToolbar.h

HIToolbarGetDisplayMode

Obtains the current display mode of a toolbar.

```
OSStatus HIToolbarGetDisplayMode (
   HIToolbarRef inToolbar,
   HIToolbarDisplayMode *outDisplayMode
);
```

Parameters

inToolbar

The toolbar whose display mode you want to receive.

outDisplayMode

The display mode.

Return Value
An operating system result code.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
HIToolbar.h

HIToolbarGetDisplaySize

Obtains the current display size of a toolbar.

```
OSStatus HIToolbarGetDisplaySize (
   HIToolbarRef inToolbar,
   HIToolbarDisplaySize *outSize
);
```

Parameters

inToolbar

The toolbar whose display size you want to get.

outSize

The display size.

Return Value
An operating system result code.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
HIToolbar.h

HIToolbarGetSelectedItemInWindow

Obtains the toolbar item that is selected in a window.

```
OSStatus HIToolbarGetSelectedItemInWindow (
   HIToolbarRef inToolbar,
   WindowRef inWindow,
   HIToolbarItemRef *outItem
);
```

Parameters

inToolbar

The toolbar in question.

inWindow

A window containing the toolbar.

outItem

On return, the toolbar item that is selected in the specified window, or `NULL` if no item is selected.

Return Value
An operating system result code.

Discussion
Each window that shares a toolbar may have a different selected item. The `HIToolbarGetSelectedItemInWindow` function returns the selected item in a particular window.

Availability
Available in Mac OS X v10.4 and later.

Not available to 64-bit applications.

Declared In
`HIToolbar.h`

HIToolbarInsertItemAtIndex

Inserts a toolbar item at a given index into a toolbar.

```
OSStatus HIToolbarInsertItemAtIndex (
   HIToolbarRef inToolbar,
   HIToolbarItemRef inItem,
   CFIndex inIndex
);
```

Parameters

inToolbar

The toolbar to receive the new item.

inItem

The item reference of the toolbar item you are adding.

inIndex

The index at which you want to add the item. This index is zero-based.

Return Value
An operating system result code.

Discussion
Generally, you should always add items via identifier, and not with this routine.

Availability
Available in Mac OS X v10.2 and later.

Not available to 64-bit applications.

Declared In
`HIToolbar.h`

HIToolbarItemChangeAttributes

Changes the attributes of a toolbar item.

```
OSStatus HIToolbarItemChangeAttributes (
   HIToolbarItemRef inItem,
   OptionBits inAttrsToSet,
   OptionBits inAttrsToClear
);
```

Parameters

inItem

The item in question.

inAttrsToSet

The attributes to set on the item. For possible attributes, see "Toolbar Item Attributes" (page 2391). Use `kHIToolbarItemNoAttributes` if you are clearing attributes.

inAttrsToClear

The attributes to clear on the item. This value can be `kHIToolbarItemNoAttributes` if you are setting attributes.

Return Value
An operating system result code.

Discussion
Only those attributes defined by the `kHIToolbarItemMutableAttrs` constant can be passed into this function.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`HIToolbar.h`

HIToolbarItemChangeAttributesInWindow

Changes the attributes of a toolbar item in a specific window.

```
OSStatus HIToolbarItemChangeAttributesInWindow (
   HIToolbarItemRef inItem,
   WindowRef inWindow,
   OptionBits inAttrsToSet,
   OptionBits inAttrsToClear,
   OptionBits inAttrsToNoLongerOverride
);
```

Parameters

inItem

The item in question.

inWindow

The window containing the item in question.

inAttrsToSet

The attributes to set on the item. For possible attributes, see "Toolbar Item Attributes" (page 2391). Use `kHIToolbarItemNoAttributes` if you are clearing attributes and have no attributes to set.

inAttrsToClear

The attributes to clear on the item. This value can be `kHIToolbarItemNoAttributes` if you are setting attributes and have no attributes to clear.

inAttrsToNoLongerOverride

The attributes that are to no longer be overridden. Calling this function causes the attributes to be removed from the override mask for the toolbar item in the specified window. Their effective values revert to their non-window-specific attribute values.

Return Value

An operating system result code.

Discussion

This function allows the attributes of a toolbar item in the specified window to be overridden. The attributes used to draw the view of a toolbar item in a particular window are determined by combining the non-window-specific attributes for the item set by `HIToolbarItemChangeAttributes` (page 2370) with the window-specific attributes set by this function. As a result, your application can have a toolbar that is shared across several windows with a toolbar item that is enabled in one window and disabled in another window.

When `HIToolbarItemChangeAttributesInWindow` is called to set or clear attributes, the toolbar item adds the changed attributes to a bitmask of attributes, thereby recording which attributes are overridden for a particular window. Once an attribute is overridden for a window (regardless of whether the attribute is set or cleared), the attribute remains overridden for that window until `HIToolbarItemChangeAttributesInWindow` is called with that attribute specified in the `inAttrsToNoLongerOverride` parameter.

Only those attributes defined by the `kHIToolbarItemMutableAttrs` constant can be passed into this function. For details, see "Toolbar Item Attributes" (page 2391).

Availability

Available in Mac OS X v10.4 and later.

Not available to 64-bit applications.

Declared In

`HIToolbar.h`

HIToolbarItemConfigDataChanged

Tells the toolbar that the configuration for a toolbar item has changed.

```
OSStatus HIToolbarItemConfigDataChanged (
   HIToolbarItemRef inItem
);
```

Parameters

inItem

The item whose configuration changed.

Return Value
An operating system result code.

Discussion
This function tells the toolbar that the config data for a toolbar item has changed and should be written to the toolbar configuration preferences. This function is used when a custom toolbar item has extra configuration data that has changed (for example, you've changed an alias that a toolbar item points to). This function does nothing if the toolbar is not set to auto-save its configuration.

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In
`HIToolbar.h`

HIToolbarItemCopyHelpText

Obtains the help tag text for a toolbar item.

```
OSStatus HIToolbarItemCopyHelpText (
   HIToolbarItemRef inItem,
   CFStringRef *outShortText,
   CFStringRef *outLongText
);
```

Parameters

inItem

The item in question.

outShortText

The short help text. This is what is displayed normally by the help tag system when the user hovers over the toolbar item with the mouse. You should release this string when you are finished with it. If you do not want to receive the short help text, pass `NULL` for this parameter.

outLongText

The long help text. This is what is displayed by the help tag system when the user hovers over the toolbar item with the mouse and holds the command key down. You should release this string when you are finished with it. If you do not want to receive the long help text, pass `NULL` for this parameter.

Return Value
An operating system result code.

Availability
Available in Mac OS X v10.2 and later.

Not available to 64-bit applications.

Declared In
HIToolbar.h

HIToolbarItemCopyIdentifier

Obtains the identifier for a given toolbar item.

```
OSStatus HIToolbarItemCopyIdentifier (
   HIToolbarItemRef inItem,
   CFStringRef *outIdentifier
);
```

Parameters

inItem
The item in question.

outIdentifier
The identifier of the item. You should release this string when you are finished with it.

Return Value
An operating system result code.

Discussion
The toolbar uses this identifier when writing the config information to the preferences (if set up for auto-config).

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
HIToolbar.h

HIToolbarItemCopyImage

Obtains the image for a toolbar item.

```
OSStatus HIToolbarItemCopyImage (
   HIToolbarItemRef inItem,
   CGImageRef *outImage
);
```

Parameters

inItem
The item in question.

outImage
The retained image. You should release it when finished with it.

Return Value
An operating system result code.

Discussion
This image is already retained by the time you receive it, so you can release it when you are done with it.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`HIToolbar.h`

HIToolbarItemCopyLabel

Obtains the label for a toolbar item.

```
OSStatus HIToolbarItemCopyLabel (
   HIToolbarItemRef inItem,
   CFStringRef *outLabel
);
```

Parameters

inItem
The item in question.

outLabel
The label of the item. You should release this when you are finished with it.

Return Value
An operating system result code.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`HIToolbar.h`

HIToolbarItemCopyMenu

Obtains the submenu for a toolbar item.

```
OSStatus HIToolbarItemCopyMenu (
   HIToolbarItemRef inItem,
   MenuRef *outMenu
);
```

Parameters

inItem
The item in question.

outMenu
The retained menu. You should release it when you are finished with it.

Return Value
An operating system result code.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`HIToolbar.h`

HIToolbarItemCreate

Creates a toolbar item.

```
OSStatus HIToolbarItemCreate (
   CFStringRef inIdentifier,
   OptionBits inOptions,
   HIToolbarItemRef *outItem
);
```

Parameters

inIdentifier
The identifier of the item in question.

inOptions
Any options for the item.

outItem
The item you created.

Return Value
An operating system result code.

Discussion
This function creates a toolbar item for use in a toolbar. Typically, you call `HIToolbarItemCreate` from inside your delegate when your delegate is asked to create a toolbar item.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`HIToolbar.h`

HIToolbarItemGetAttributes

Obtains the attributes of a toolbar item.

```
OSStatus HIToolbarItemGetAttributes (
   HIToolbarItemRef inItem,
   OptionBits *outAttributes
);
```

Parameters

inItem
The item in question.

outAttributes
The attributes of the item. For details, see "Toolbar Item Attributes" (page 2391).

Return Value
An operating system result code.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
HIToolbar.h

HIToolbarItemGetAttributesInWindow

Obtains the attributes of a toolbar item in the specified window.

```
OSStatus HIToolbarItemGetAttributesInWindow (
   HIToolbarItemRef inItem,
   WindowRef inWindow,
   OptionBits *outOverriddenAttributes,
   OptionBits *outCombinedAttributes
);
```

Parameters

inItem
The item in question.

inWindow
The window containing `inItem`. Passing `NULL` is the same as calling `HIToolbarItemGetAttributes` (page 2375), which obtains the non-window-specific attributes for the item.

outOverriddenAttributes
On return, a bitmask indicating the attributes for the item that are overridden in the specified window. If you don't need this information, pass `NULL`. For details on this bitmask, see "Toolbar Item Attributes" (page 2391).

outCombinedAttributes
On return, a bitmask indicating the effective attributes for the item in the specified window. The bitmask is produced by combining the non-window-specific attributes for the item with the overridden attributes for this window. If you don't need this information, pass `NULL`. For details on this bitmask, see "Toolbar Item Attributes" (page 2391).

Return Value
An operating system result code.

Discussion
In addition to obtaining the attributes of the given item in a given window, `HIToolbarItemGetAttributesInWindow` obtains information about which attributes are overridden for that window.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
HIToolbar.h

HIToolbarItemGetCommandID

Gets the command ID of a toolbar item.

```
OSStatus HIToolbarItemGetCommandID (
   HIToolbarItemRef inItem,
   MenuCommand *outCommandID
);
```

Parameters

inItem

The item whose command ID is to be obtained.

outCommandID

On return, the item's command ID.

Return Value
An operating system result code.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
HIToolbar.h

HIToolbarItemGetToolbar

Obtains the toolbar associated with a toolbar item.

```
HIToolbarRef HIToolbarItemGetToolbar (
   HIToolbarItemRef inItem
);
```

Parameters

inItem

The item in question.

Return Value
The toolbar item reference of the toolbar this item is bound to, or NULL if this toolbar item is not attached to any toolbar.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
HIToolbar.h

HIToolbarItemIsEnabled

Determines if a toolbar item is enabled.

```
Boolean HIToolbarItemIsEnabled (
   HIToolbarItemRef inItem
);
```

Parameters

inItem

The item in question.

Return Value

A Boolean result indicating whether the item is enabled.

Availability

Available in Mac OS X v10.2 and later.

Not available to 64-bit applications.

Declared In

`HIToolbar.h`

HIToolbarItemSetCommandID

Sets the command ID of a toolbar item.

```
OSStatus HIToolbarItemSetCommandID (
   HIToolbarItemRef inItem,
   MenuCommand inCommandID
);
```

Parameters

inItem

The item whose command ID is to be set.

inCommandID

The command ID to set.

Return Value

An operating system result code.

Discussion

When an toolbar item is clicked, the default behavior is to send out the command assigned to the item. This function lets you to set that command ID. The command is sent out via the `ProcessHICommand` API.

Availability

Available in Mac OS X v10.2 and later.

Not available to 64-bit applications.

Declared In

`HIToolbar.h`

HIToolbarItemSetEnabled

Enables or disables a toolbar item.

```
OSStatus HIToolbarItemSetEnabled (
   HIToolbarItemRef inItem,
   Boolean inEnabled
);
```

Parameters

inItem

The item in question.

inEnabled

The new enabled state.

Return Value
An operating system result code.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`HIToolbar.h`

HIToolbarItemSetHelpText

Sets the help tag text for a toolbar item.

```
OSStatus HIToolbarItemSetHelpText (
   HIToolbarItemRef inItem,
   CFStringRef inShortText,
   CFStringRef inLongText
);
```

Parameters

inItem

The item in question.

inShortText

The short help text. This is what is displayed normally by the help tag system when the user hovers over the toolbar item with the mouse.

inLongText

The long help text. This is what is displayed by the help tag system when the user hovers over the toolbar item with the mouse and holds the command key down. This parameter is optional, you may pass `NULL`.

Return Value
An operating system result code.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`HIToolbar.h`

HIToolbarItemSetIconRef

Sets the icon for a toolbar item.

```
OSStatus HIToolbarItemSetIconRef (
   HIToolbarItemRef inItem,
   IconRef inIcon
);
```

Parameters

inItem

The item in question.

inIcon

The icon reference. The IconRef is retained by the toolbar item. You can release the reference for this icon when `HIToolbarItemSetIconRef` returns.

Return Value

An operating system result code.

Availability

Available in Mac OS X v10.2 and later.

Not available to 64-bit applications.

Declared In

`HIToolbar.h`

HIToolbarItemSetImage

Sets the image for a toolbar item.

```
OSStatus HIToolbarItemSetImage (
   HIToolbarItemRef inItem,
   CGImageRef inImage
);
```

Parameters

inItem

The item in question.

inImage

The image. This image is retained by the toolbar item. You should release the image when you are finished with it.

Return Value

An operating system result code.

Discussion

Currently, the image should be no higher than 32 pixels. This image is used both in the toolbar as well as the configuration sheet, if the toolbar is configurable.

Availability

Available in Mac OS X v10.2 and later.

Not available to 64-bit applications.

Declared In

`HIToolbar.h`

HIToolbarItemSetLabel

Sets the label of a toolbar item.

```
OSStatus HIToolbarItemSetLabel (
   HIToolbarItemRef inItem,
   CFStringRef inLabel
);
```

Parameters

inItem

The item in question.

inLabel

The label. The toolbox will copy the string passed in.

Return Value

An operating system result code.

Discussion

This is what the toolbar view will display underneath the image. It is also used in the configuration palette for configurable toolbars.

Availability

Available in Mac OS X v10.2 and later.

Not available to 64-bit applications.

Declared In

`HIToolbar.h`

HIToolbarItemSetMenu

Sets the submenu for a toolbar item.

```
OSStatus HIToolbarItemSetMenu (
   HIToolbarItemRef inItem,
   MenuRef inMenu
);
```

Parameters

inItem

The item in question.

inMenu

The menu. It is retained by the toolbar item, so you can safely release it after calling this API. On Mac OS X v10.3 and later, you can pass `NULL` for this parameter to remove and release any menu that might be attached.

Return Value

An operating system result code.

Discussion

Normally, items do not have a submenu. You can attach one with this API. The submenu will, by default, show up both in the 'more items' indicator popup attached to the item name. It will also appear if the toolbar is in text only mode and the label is clicked. You should attach a Carbon Event handler to the menu to handle updating the menu items as appropriate before the menu is displayed.

Availability
Available in Mac OS X v10.2 and later
Not available to 64-bit applications.

Declared In
HIToolbar.h

HIToolbarRemoveItemAtIndex

Removes an item at a given index from a toolbar.

```
OSStatus HIToolbarRemoveItemAtIndex (
   HIToolbarRef inToolbar,
   CFIndex inIndex
);
```

Parameters

inToolbar
The toolbar you are removing the item from.

inIndex
The index of the item to remove. This index is zero-based.

Return Value
An operating system result code.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
HIToolbar.h

HIToolbarSetDelegate

Sets the delegate object for a toolbar.

```
OSStatus HIToolbarSetDelegate (
   HIToolbarRef inToolbar,
   HIObjectRef inDelegate
);
```

Parameters

inToolbar
The toolbar whose delegate you want to set.

inDelegate
The HIObjectRef to act as the delegate.

Return Value
An operating system result code.

Discussion
If a toolbar has custom toolbar items, a delegate is required for the toolbar to work properly. The delegate is asked to create the toolbar items. If the delegate does not respond, the toolbar attempts to create the toolbar items, but it can only create standard items.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`HIToolbar.h`

HIToolbarSetDisplayMode

Sets the current display mode of a toolbar.

```
OSStatus HIToolbarSetDisplayMode (
   HIToolbarRef inToolbar,
   HIToolbarDisplayMode inDisplayMode
);
```

Parameters

inToolbar
The toolbar whose display mode you want to set.

inDisplayMode
The display mode. If the toolbar is visible, it will be redrawn as necessary.

Return Value
An operating system result code.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`HIToolbar.h`

HIToolbarSetDisplaySize

Sets the current display size of a toolbar.

```
OSStatus HIToolbarSetDisplaySize (
   HIToolbarRef inToolbar,
   HIToolbarDisplaySize inSize
);
```

Parameters

inToolbar
The toolbar whose display size you want to set.

inSize
The display size. If the toolbar is visible, it will be redrawn as necessary.

Return Value
An operating system result code.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`HIToolbar.h`

HIToolbarSetItemsWithIdentifiers

Sets a toolbar's items all at once.

```
OSStatus HIToolbarSetItemsWithIdentifiers (
   HIToolbarRef inToolbar,
   CFArrayRef inArray
);
```

Parameters

`inToolbar`
The toolbar whose items you want to set.

`inArray`
The array of toolbar items to create.

Return Value
An operating system result code.

Discussion
This function allows you to set a toolbar's items all at once. The entries in the array `inArray` must be either CFStringRefs or CFDictionaryRefs. You do not need to use the same type for all entries; different entries can be of different types. If an array entry is a CFStringRef, the string must contain a toolbar item identifier. If an array entry is a CFDictionaryRef, the dictionary must contain a CFStringRef specifying a toolbar item identifier and may optionally contain a CFTypeRef specifying the toolbar item's configuration data, if the item requires it. The key for the identifier string is `kHIToolbarIdentifierKey` and the key for the configuration data string is `kHIToolbarDataKey`.

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In
`HIToolbar.h`

Constants

Toolbar Attributes

Specify constants for toolbar attributes.

```
enum {
    kHIToolbarNoAttributes = 0,
    kHIToolbarAutoSavesConfig = (1 << 0),
    kHIToolbarIsConfigurable = (1 << 1),
    kHIToolbarValidAttrs = kHIToolbarAutoSavesConfig | kHIToolbarIsConfigurable
};
```

Constants

`kHIToolbarNoAttributes`

Pass this to indicate no attributes at all.

Available in Mac OS X v10.2 and later.

Declared in `HIToolbar.h`.

`kHIToolbarAutoSavesConfig`

Pass this attribute to allow the toolbar to save its configuration automatically to your application's preferences. You must make sure to synchronize the preferences at some point to ensure it gets written to disk. The toolbar will also read its configuration from the preferences if this attribute is set.

Available in Mac OS X v10.2 and later.

Declared in `HIToolbar.h`.

`kHIToolbarIsConfigurable`

This attribute indicates that the toolbar is configurable, that is, the user can drag items around and bring up the configuration palette, etc.

Available in Mac OS X v10.2 and later.

Declared in `HIToolbar.h`.

`kHIToolbarValidAttrs`

The set of all valid toolbar attributes.

Available in Mac OS X v10.2 and later.

Declared in `HIToolbar.h`.

Toolbar Command ID Constants

Specify constants for toolbar Command IDs.

```
enum {
    kHICommandCustomizeToolbar = 'tcfg',
    kHICommandShowToolbar = 'tbsh',
    kHICommandHideToolbar = 'tbhd',
    kHIToolbarCommandPressAction = 'tbpr'
};
```

Constants

`kHICommandCustomizeToolbar`

When sent to a window with a toolbar, this command causes the configuration sheet to appear. You can set a menu item's command to this command ID and it will be handled and updated automatically for you. (Available in Mac OX X v10.2 and later.)

Available in Mac OS X v10.2 and later.

Declared in `HIToolbar.h`.

`kHICommandShowToolbar`
Sending this command causes a window's toolbar to be shown. You can set a menu item's command to this ID and it will be handled and updated automatically for you. (Available in Mac OX X v10.2 and later.)

Available in Mac OS X v10.2 and later.

Declared in `HIToolbar.h`.

`kHICommandHideToolbar`
Sending this command causes a window's toolbar to be hidden. You can set a menu item's command to this ID and it will be handled and updated automatically for you. (Available in Mac OX X v10.2 and later.)

Available in Mac OS X v10.2 and later.

Declared in `HIToolbar.h`.

`kHIToolbarCommandPressAction`
This command, when specified as a toolbar's command ID, causes a `kEventToolbarItemPerformAction` event to be generated when the toolbar item's menu item in the toolbar overflow menu is selected. If the item has any other command ID, a `kEventCommandProcess` event, containing the item's command ID, is generated instead. (Available in Mac OX X v10.2.3 and later.)

Available in Mac OS X v10.3 and later.

Declared in `HIToolbar.h`.

Toolbar Display Mode Constants

Specify constants for toolbar display modes.

```
enum {
    kHIToolbarDisplayModeDefault = 0,
    kHIToolbarDisplayModeIconAndLabel = 1,
    kHIToolbarDisplayModeIconOnly = 2,
    kHIToolbarDisplayModeLabelOnly = 3
};
```

Constants

`kHIToolbarDisplayModeDefault`
Use the default display mode. Currently, this is defined as being both icon and label, but could change in the future.

Available in Mac OS X v10.2 and later.

Declared in `HIToolbar.h`.

`kHIToolbarDisplayModeIconAndLabel`
Display the image as well as the label of the toolbar items.

Available in Mac OS X v10.2 and later.

Declared in `HIToolbar.h`.

`kHIToolbarDisplayModeIconOnly`
Display only the image.

Available in Mac OS X v10.2 and later.

Declared in `HIToolbar.h`.

`kHIToolbarDisplayModeLabelOnly`
Display only the label.

Available in Mac OS X v10.2 and later.

Declared in `HIToolbar.h`.

Toolbar Display Size Constants

Specify constants for the display size of items in the toolbar.

```
enum {
    kHIToolbarDisplaySizeDefault = 0,
    kHIToolbarDisplaySizeNormal = 1,
    kHIToolbarDisplaySizeSmall = 2
};
```

Constants

`kHIToolbarDisplaySizeDefault`
This indicates to use the default display size. Currently, this is defined as using 32 x 32 icons (“normal” size).

Available in Mac OS X v10.2 and later.

Declared in `HIToolbar.h`.

`kHIToolbarDisplaySizeNormal`
This size uses a larger text and icon size.

Available in Mac OS X v10.2 and later.

Declared in `HIToolbar.h`.

`kHIToolbarDisplaySizeSmall`
This size uses a smaller text and icon size.

Available in Mac OS X v10.2 and later.

Declared in `HIToolbar.h`.

Toolbar Events

Specify toolbar Carbon event constants.

```
enum {
    kEventToolbarGetDefaultIdentifiers = 1,
    kEventToolbarGetAllowedIdentifiers = 2,
    kEventToolbarCreateItemWithIdentifier = 3,
    kEventToolbarCreateItemFromDrag = 4,
    kEventToolbarItemAdded = 5,
    kEventToolbarItemRemoved = 6,
    kEventToolbarDisplayModeChanged = 7,
    kEventToolbarDisplaySizeChanged = 8,
    kEventToolbarLayoutChanged = 9,
    kEventToolbarGetSelectableIdentifiers = 10,
    kEventToolbarBeginMultiChange = 12,
    kEventToolbarEndMultiChange = 13
};
```

Constants

`kEventToolbarGetDefaultIdentifiers`

This event is sent to the delegate to get a list of all of the default item identifiers that should be created for a toolbar. You are passed a mutable array to fill in with the identifiers. (

Available in Mac OS X v10.2.)

Declared in `HIToolbar.h`.

`kEventToolbarGetAllowedIdentifiers`

This event is sent to the delegate to get a list of all the items which could possibly be added to the toolbar. This is sent out when the configuration sheet is about to be displayed.You are passed a mutable array to fill in with the identifiers. (

Available in Mac OS X v10.2.)

Declared in `HIToolbar.h`.

`kEventToolbarCreateItemWithIdentifier`

This event is sent to the delegate to when the toolbar needs to create an item from an identifier. (

Available in Mac OS X v10.2.)

Declared in `HIToolbar.h`.

`kEventToolbarCreateItemFromDrag`

This event is sent to the delegate to when the toolbar needs to create an item from a drag. This allows you to be able to drag items into a toolbar that aren't normal toolbar items. You might use this to allow your toolbar to accept file system items, for example. (

Declared in `HIToolbar.h`.

Available in Mac OS X v10.2.)

`kEventToolbarItemAdded`

This event is sent when an item is added to the toolbar. The toolbar object sends this event to itself, so you need to install a handler on the toolbar to receive this event. (Available in Mac OS X v10.2 and later.)

Available in Mac OS X v10.3 and later.

Declared in `HIToolbar.h`.

`kEventToolbarItemRemoved`

This event is sent when an item is removed from toolbar. It is called after the item has already been removed. The toolbar object sends this event to itself, so you need to install a handler on the toolbar to receive this event. (Available in Mac OS X v10.2 and later.)

Available in Mac OS X v10.3 and later.

Declared in `HIToolbar.h`.

`kEventToolbarDisplayModeChanged`

This event is sent when the display mode is changed for a toolbar. The toolbar object sends this event to itself, so you need to install a handler on the toolbar to receive this event. (Available in Mac OS X v10.2 and later.)

Available in Mac OS X v10.3 and later.

Declared in `HIToolbar.h`.

`kEventToolbarDisplaySizeChanged`

This event is sent when the display size is changed for a toolbar. The toolbar object sends this event to itself, so you need to install a handler on the toolbar to receive this event. (Available in Mac OS X v10.2 and later.)

Available in Mac OS X v10.3 and later.

Declared in `HIToolbar.h`.

`kEventToolbarLayoutChanged`

Sent when the layout of a toolbar changes (either an item has been moved, or the entire contents have been replaced). Basically it is sent for changes that would require a total resync with the current state of things. The toolbar object sends this event to itself, so you must install a handler on the toolbar to receive this event. (Available in Mac OS X v10.2 and later.)

Available in Mac OS X v10.3 and later.

Declared in `HIToolbar.h`.

`kEventToolbarBeginMultiChange`

This event is sent when multiple attributes are going to be changed at the same time. For example, the display mode and size can change at the same time. When this happens, instead of reacting twice (once for a display mode change and once for a display size change), you can listen to see if more than one attribute is about to change and wait to redraw the toolbar until you receive the `kEventToolbarEndMultiChange` event. The toolbar object sends this event to itself, so you need to install a handler on the toolbar to receive this event. (Available in Mac OS X v10.2 and later.)

Available in Mac OS X v10.3 and later.

Declared in `HIToolbar.h`.

`kEventToolbarEndMultiChange`

This event is sent when multiple changes in the toolbar have been made. For details, see the description of `kEventToolbarBeginMultiChange`. The toolbar object sends this event to itself, so you need to install a handler on the toolbar to receive this event. (Available in Mac OS X v10.2 and later.)

Available in Mac OS X v10.3 and later.

Declared in `HIToolbar.h`.

`kEventToolbarGetSelectableIdentifiers`

This event is sent to the delegate after the user clicks a toolbar item in order to get a list of all the items that can acquire a selection highlight when clicked. You are passed a mutable array to fill in with the identifiers. If you pass back a non-empty array and the clicked item's identifier matches one of the identifiers that is in the list, the toolbar automatically draws that item with a selected highlight and removes highlighting from the previously selected item. Note that the selection only changes in the clicked window — it does not change in other windows that share the same toolbar. To share the selection across all windows that use the same toolbar, you need to manually change the `kHIToolbarItemSelected` attribute for the clicked item using `HIToolbarItemChangeAttributes` (page 2370). In this case, you should not handle the `kEventToolbarGetSelectableIdentifiers` event. (Available in Mac OS X v10.4 and later.)

Available in Mac OS X v10.4 and later.

Declared in `HIToolbar.h`.

Discussion
Table 33-1 lists the parameters and types for toolbar events.

Table 33-1 Parameter names and types for toolbar events

Event kind	Parameter name	Parameter type
kEventToolbarGet-DefaultIdentifiers	kEventParamToolbar	typeHIToolbarRef
	kEventParamMutableArray	typeCFMutableArrayRef
kEventToolbarGet-AllowedIdentifiers	kEventParamToolbar	typeHIToolbarRef
	kEventParamMutableArray	typeCFMutableArrayRef
kEventToolbarGetSelectable-Identifiers	kEventParamToolbar	typeHIToolbarRef
	kEventParamMutableArray	typeCFMutableArrayRef
kEventToolbarCreate-ItemWithIdentifier	kEventParamToolbar	typeHIToolbarRef
	kEventParamToolbar-ItemIdentifier	typeCFStringRef
	kEventParamToolbar-ItemConfigData	typeCFTypeRef
	kEventParamToolbarItem	typeHIToolbarItemRef
kEventToolbarCreate-ItemFromDrag	kEventParamDragRef	typeDragRef
	kEventParamToolbarItem	typeHIToolbarItemRef
kEventToolbarItemAdded	kEventParamToolbarItem	typeHIToolbarItemRef
	kEventParamIndex	typeCFIndex
kEventToolbarItemRemoved	kEventParamToolbarItem	typeHIToolbarItemRef
kEventToolbarDisplay-ModeChanged	*None*	
kEventToolbarDisplay-SizeChanged	*None*	
kEventToolbarLayoutChanged	*None*	
kEventToolbarBeginMultiChange	*None*	
kEventToolbarEndMultiChange	*None*	

Toolbar Event Parameters and Types

Specify constants for toolbar event parameters and types.

```
enum {
    kEventParamToolbar = 'tbar',
    kEventParamToolbarItem = 'tbit',
    kEventParamToolbarItemIdentifier = 'tbii',
    kEventParamToolbarItemConfigData = 'tbid',
    typeHIToolbarRef = 'tbar',
    typeHIToolbarItemRef = 'tbit'
};
```

Constants

`kEventParamToolbar`
The toolbar to which an event was sent.
Available in Mac OS X v10.2 and later.
Declared in `HIToolbar.h`.

`kEventParamToolbarItem`
The toolbar item to which an event was sent.
Available in Mac OS X v10.2 and later.
Declared in `HIToolbar.h`.

`kEventParamToolbarItemIdentifier`
The toolbar item's identifier.
Available in Mac OS X v10.2 and later.
Declared in `HIToolbar.h`.

`kEventParamToolbarItemConfigData`
The toolbar item's configuration information.
Available in Mac OS X v10.2 and later.
Declared in `HIToolbar.h`.

`typeHIToolbarRef`
HIToolbarRef
Available in Mac OS X v10.2 and later.
Declared in `HIToolbar.h`.

`typeHIToolbarItemRef`
HIToolbarItemRef
Available in Mac OS X v10.2 and later.
Declared in `HIToolbar.h`.

Toolbar Item Attributes

Specify constants for toolbar item attributes.

```
enum {
    kHIToolbarItemNoAttributes = 0,
    kHIToolbarItemAllowDuplicates = (1 << 0),
    kHIToolbarItemCantBeRemoved = (1 << 1),
    kHIToolbarItemAnchoredLeft = (1 << 2),
    kHIToolbarItemIsSeparator = (1 << 3),
    kHIToolbarItemSendCmdToUserFocus = (1 << 4),
    kHIToolbarItemLabelDisabled = (1 << 5),
    kHIToolbarItemDisabled = (1 << 6),
    kHIToolbarItemSelected = (1 << 7),

    kHIToolbarItemValidAttrs = kHIToolbarItemAllowDuplicates |
kHIToolbarItemIsSeparator | kHIToolbarItemCantBeRemoved |
kHIToolbarItemAnchoredLeft | kHIToolbarItemSendCmdToUserFocus |
kHIToolbarItemLabelDisabled | kHIToolbarItemDisabled | kHIToolbarItemSelected,
    kHIToolbarItemMutableAttrs = kHIToolbarItemCantBeRemoved |
kHIToolbarItemAnchoredLeft | kHIToolbarItemLabelDisabled |  kHIToolbarItemDisabled
 | kHIToolbarItemSelected
};
```

Constants

`kHIToolbarItemNoAttributes`
Pass this to indicate no attributes at all.

Available in Mac OS X v10.2 and later.

Declared in `HIToolbar.h`.

`kHIToolbarItemAllowDuplicates`
This indicates that an item can have more than one instance of itself in the toolbar. If this is not set, only one can be present. By default, the determining factor for what determines if two items are identical is the toolbar identifier.

Available in Mac OS X v10.2 and later.

Declared in `HIToolbar.h`.

`kHIToolbarItemCantBeRemoved`
This item can be rearranged, but it cannot be removed from the toolbar by the user.

Available in Mac OS X v10.2 and later.

Declared in `HIToolbar.h`.

`kHIToolbarItemAnchoredLeft`
This item cannot be moved at all by the user. It is anchored to the left of the toolbar. It is important that there not be any unanchored items to the left of this item, else dragging items around will be a bit wacky. You are responsible for making sure that anchored items are all on the left. This allows you to do toolbars like the one in the System Preferences app, where the first couple of items are stuck in place.

Available in Mac OS X v10.2 and later.

Declared in `HIToolbar.h`.

`kHIToolbarItemIsSeparator`
This indicates the item acts as a separator. This means two things at present. First, it means that it automatically shows up as a divider line in a menu representation of the toolbar, and second it means the view that represents this item can draw in the full top to bottom space that the toolbar item occupies in the toolbar.

Available in Mac OS X v10.2 and later.

Declared in `HIToolbar.h`.

`kHIToolbarItemSendCmdToUserFocus`
: If this attribute bit is set, the command that gets sent out will be directed at the user focus instead of at the window the toolbar is attached to.

  Available in Mac OS X v10.2 and later.

  Declared in `HIToolbar.h`.

`kHIToolbarItemLabelDisabled`
: If this attribute bit is set, clicking on the label of an item does nothing. This attribute is only considered when a custom view is present. When set, this attribute makes the toolbar item view unresponsive to clicks while still allowing clicks to be sent to the custom view. When the toolbar is in text-only mode and this attribute is set, the label is displayed in a disabled (grayed) appearance. You might want to change this attribute when switching between display modes. For example, the view switcher in the Finder does not respond to clicks on the label when in icon and text mode, but it does respond to clicks when in text-only mode. To change this behavior on the fly, listen for `kEventToolbarItemViewConfigForMode` in your custom view and adjust this attribute as you want. (Available in Mac OS X 10.3 and later.)

  Available in Mac OS X v10.3 and later.

  Declared in `HIToolbar.h`.

`kHIToolbarItemDisabled`
: If this attribute bit is set, the item is disabled. Setting this attribute is the same as calling `HIToolbarItemSetEnabled` (page 2378) on the item with the `inEnabled` parameter set to `false`. (Available in Mac OS X v10.4 and later.)

  Available in Mac OS X v10.4 and later.

  Declared in `HIToolbar.h`.

`kHIToolbarItemSelected`
: If this attribute bit is set, the item is drawn with a selected appearance. (Available in Mac OS X v10.4 and later.)

  Available in Mac OS X v10.4 and later.

  Declared in `HIToolbar.h`.

`kHIToolbarItemValidAttrs`
: The set of all valid toolbar item attributes.

  Available in Mac OS X v10.2 and later.

  Declared in `HIToolbar.h`.

`kHIToolbarItemMutableAttrs`
: The set of toolbar item attributes that can be changed by `HIToolbarItemChangeAttributes` (page 2370) and `HIToolbarItemChangeAttributesInWindow` (page 2370). Any other attribute must be specified when it is created.

  Available in Mac OS X v10.2 and later.

  Declared in `HIToolbar.h`.

Toolbar Item Events

Specify constants for toolbar item events.

```
enum {
    kEventToolbarItemImageChanged = 1,
    kEventToolbarItemLabelChanged = 2,
    kEventToolbarItemHelpTextChanged = 3,
    kEventToolbarItemCommandIDChanged = 4,
    kEventToolbarItemGetPersistentData = 5,
    kEventToolbarItemCreateCustomView = 6,
    kEventToolbarItemEnabledStateChanged = 7,
    kEventToolbarItemPerformAction = 8,
    kEventToolbarItemWouldAcceptDrop = 10,
    kEventToolbarItemAcceptDrop = 11,
    kEventToolbarItemSelectedStateChanged = 12
};
```

Constants

`kEventToolbarItemImageChanged`

This event is sent to the item when the image changes. Any interested parties can install handlers on the toolbar item to receive notifications. (Available in Mac OS X v10.2 and later.)

Available in Mac OS X v10.2 and later.

Declared in `HIToolbar.h`.

`kEventToolbarItemLabelChanged`

This event is sent to the item when the label changes. Any interested parties can install handlers on the toolbar item to receive notifications. (Available in Mac OS X v10.2 and later.)

Available in Mac OS X v10.2 and later.

Declared in `HIToolbar.h`.

`kEventToolbarItemHelpTextChanged`

This event is sent to the item when the help text changes. Any interested parties can install handlers on the toolbar item to receive notifications. (Available in Mac OS X v10.2 and later.)

Available in Mac OS X v10.2 and later.

Declared in `HIToolbar.h`.

`kEventToolbarItemCommandIDChanged`

This event is sent to the item when the command ID changes. Any interested parties can install handlers on the toolbar item to receive notifications. (Available in Mac OS X v10.2 and later.)

Available in Mac OS X v10.2 and later.

Declared in `HIToolbar.h`.

`kEventToolbarItemGetPersistentData`

This event is sent to the item when the toolbar is going to write out the configuration information for the item. Any custom items can listen for this event and add any extra information to what is written out into the config so that it can be reanimated later on from the same config data. Typically, you do not need to handle this event. (Available in Mac OS X v10.2 and later.)

Available in Mac OS X v10.2 and later.

Declared in `HIToolbar.h`.

`kEventToolbarItemCreateCustomView`

This event is sent to the toolbar item when it is time to create a view for it to display its contents. Implementors of custom toolbar items can install a handler for this event to create their own custom views for their items. (Available in Mac OS X v10.2 and later.)

Available in Mac OS X v10.2 and later.

Declared in `HIToolbar.h`.

`kEventToolbarItemEnabledStateChanged`
This event is sent to the item when the enabled state changes. Any interested parties can install handlers on the toolbar item to receive notifications. (Available in Mac OS X v10.3 and later.)

Available in Mac OS X v10.2 and later.

Declared in `HIToolbar.h`.

`kEventToolbarItemPerformAction`
This event is sent when a toolbar item is clicked. Subclasses of toolbar items can choose to do special actions by overriding this event. If this event is unhandled, the default action of sending a command event will occur. (Available in Mac OS X v10.2 and later.)

Available in Mac OS X v10.2 and later.

Declared in `HIToolbar.h`.

`kEventToolbarItemWouldAcceptDrop`
This event is sent when a toolbar item is clicked. Subclasses of toolbar items can choose to do special actions by overriding this event. If this event is unhandled, the default action of sending a command event will occur. (Available in Mac OS X v10.3 and later.)

Available in Mac OS X v10.3 and later.

Declared in `HIToolbar.h`.

`kEventToolbarItemAcceptDrop`
This event is sent when a drag enters a toolbar item. If the toolbar item wants to accept drags, it should respond to this event and return `true` in the `kEventParamResult` parameter. The toolbar item will be highlighted appropriately. If you are using a custom view, you do not need to respond to this event because the view system provides full drag and drop capability in order to support custom items that use the standard view. (Available in Mac OS X v10.3 and later.)

Available in Mac OS X v10.3 and later.

Declared in `HIToolbar.h`.

`kEventToolbarItemSelectedStateChanged`
This event is sent to the item itself when the selected state changes. Any interested parties can install handlers on the toolbar item to receive notifications. (Available in Mac OS X v10.4 and later.)

Available in Mac OS X v10.4 and later.

Declared in `HIToolbar.h`.

Discussion
Table 33-2 lists the parameters and types for toolbar item events.

Table 33-2 Parameter names and types for toolbar item events

Event kind	Parameter name	Parameter type
`kEventToolbarItemImageChanged`	*None*	
`kEventToolbarItemLabelChanged`	*None*	
`kEventToolbarItemHelpTextChanged`	*None*	
`kEventToolbarItemCommandIDChanged`	*None*	
`kEventToolbarItemGetPersistentData`	`kEventParamToolbar-ItemConfigData`	`typeCFTypeRef`
`kEventToolbarItemCreateCustomView`	`kEventParamControlRef`	`typeControlRef`

kEventToolbarItemEnabledStateChanged	kEventParamWindowRef (Optional)	typeWindowRef
kEventToolbarItemPerformAction	*None*	
kEventToolbarItemWouldAcceptDrop	kEventParamDragRef	typeDragRef
	kEventParamResult	typeBoolean
kEventToolbarItemAcceptDrop	kEventParamDragRef	typeDragRef
kEventToolbarItemSelectedStateChanged	kEventParamWindowRef (Optional)	typeWindowRef

Toolbar Item View Events

Specify constants for toolbar item view events.

```
enum {
    kEventToolbarItemViewConfigForMode = 3,
    kEventToolbarItemViewConfigForSize = 4,
    kEventToolbarItemViewEnterConfigMode = 5,
    kEventToolbarItemViewExitConfigMode = 6
};
```

Constants

`kEventToolbarItemViewConfigForMode`

This event notifies a toolbar item view that the toolbar's display mode has changed. A custom toolbar item view can respond in any way it sees fit. Usually, responding to this is not necessary; for example, when the toolbar goes from icon to text only the view is automatically hidden, so there is not much to do. This event is for informational purposes only. (Available in Mac OS X v10.3 and later.)

Available in Mac OS X v10.3 and later.

Declared in `HIToolbar.h`.

`kEventToolbarItemViewConfigForSize`

This event notifies a toolbar item view that the toolbar's display size has changed. A custom toolbar item view can respond to this in any way it sees fit. Usually, responding to this is not necessary. Some custom views might need to flush metrics caches when the display size changes. (Available in Mac OS X v10.3 and later.)

Available in Mac OS X v10.3 and later.

Declared in `HIToolbar.h`.

`kEventToolbarItemViewEnterConfigMode`

This event notifies a toolbar item view that configure mode has been entered. A custom toolbar item view can respond to this in any way it sees fit. For example, the space and flexible space mark themselves to draw a rectangle and merely invalidate so they get redrawn so they can be seen during configuration. (Available in Mac OS X v10.3 and later.)

Available in Mac OS X v10.3 and later.

Declared in `HIToolbar.h`.

`kEventToolbarItemViewExitConfigMode`
This event notifies a toolbar item view that configure mode has been exited. A custom toolbar item view can respond to this event in any way it sees fit. (Available in Mac OS X v10.3 and later.)

Available in Mac OS X v10.3 and later.

Declared in `HIToolbar.h`.

Discussion
Table 33-3 lists the parameters and types for toolbar item view events.

Table 33-3 Parameter names and types for toolbar item view events

Event kind	Parameter name	Parameter type
`kEventToolbarItemViewConfigForMode`	`kEventParamToolbar-DisplayMode`	`typeHIToolbar-DisplayMode`
`kEventToolbarItemViewConfigForSize`	`kEventParamToolbar-DisplaySize`	`typeHIToolbar-DisplaySize`
`kEventToolbarItem-ViewEnterConfigMode`	*None*	
`kEventToolbarItem-ViewExitConfigMode`	*None*	

Toolbar View Display Event Parameters and Types

Specify constants for toolbar display event parameters and types.

```
enum {
    kEventParamToolbarDisplayMode = 'tbdm',
    kEventParamToolbarDisplaySize = 'tbds',
    typeHIToolbarDisplayMode = 'tbdm',
    typeHIToolbarDisplaySize = 'tbds'
};
```

Constants
`kEventParamToolbarDisplayMode`
Indicates that the display mode changed.

Available in Mac OS X v10.3 and later.

Declared in `HIToolbar.h`.

`kEventParamToolbarDisplaySize`
Indicates that the display size changed.

Available in Mac OS X v10.3 and later.

Declared in `HIToolbar.h`.

`typeHIToolbarDisplayMode`
HIToolbarDisplayMode.

Available in Mac OS X v10.3 and later.

Declared in `HIToolbar.h`.

`typeHIToolbarDisplaySize`
: HIToolbarDisplaySize.

Available in Mac OS X v10.3 and later.

Declared in `HIToolbar.h`.

HIView Reference

Framework:	Carbon/Carbon.h
Declared in	HIView.h HICocoaView.h HIImageView.h HISegmentedView.h
Companion guide	HIView Programming Guide

Overview

HIView is an object-oriented view system subclassed from HIObject. All controls are implemented as HIView objects ("views"). You can easily subclass HIView classes, making it easy to implement custom controls. Over time, the HIView API will replace the current Control Manager. Using the HIView model, every item within a window is a view: the root control, controls, and even the standard window "widgets" (close, zoom, and minimize buttons, resize control, and so on). Current Control Manager function calls are layered on top of this HIView model.

Additional benefits of the HIView model include the following:

- Quartz is the native drawing system, but you can still use QuickDraw if desired.
- Modern coordinate system not bounded by the 16-bit space of QuickDraw.
- Simplified coordinate system for view bounds and the position of a view within its parent.
- Views can be ordered within a hierarchy layer; that is, it is easy to place controls in front of or behind other controls.

For additional information about using HIViews, see *HIView Programming Guide*.

Functions by Task

Obtaining and Placing Views

`HIViewGetRoot` (page 2463)
Obtains the root view for a window.

`HIViewFindByID` (page 2451)
Obtains a view by its ID.

HIViewGetSuperview (page 2464)
Returns a view's parent view.

HIViewPlaceInSuperviewAt (page 2473)
Places a view at an absolute location within its parent.

HIViewGetNextView (page 2460)
Returns the view behind the specified view.

HIViewGetPreviousView (page 2462)
Returns the view above the specified view.

Working With Subviews

HIViewAddSubview (page 2442)
Adds a subview to the given parent view.

HIViewRemoveFromSuperview (page 2475)
Removes a view from its parent.

HIViewGetFirstSubview (page 2455)
Returns the first subview of a parent view.

HIViewGetLastSubview (page 2458)
Returns the last subview in a parent view.

HIViewCountSubviews (page 2449)
Returns the number of subviews embedded in a view.

HIViewGetIndexedSubview (page 2457)
Obtains the subview of a view by index.

Manipulating Views

HIViewSetVisible (page 2490)
Hides or shows a view.

HIViewIsVisible (page 2471)
Determines whether a view is visible.

HIViewIsLatentlyVisible (page 2470)
Determines whether a view is latently visible.

HIViewSetHilite (page 2481)
Sets highlighting on a view.

HIViewIsActive (page 2467)
Determines whether a view is active.

HIViewSetActivated (page 2477)
Sets a view to be active or inactive.

HIViewIsEnabled (page 2469)
Determines whether a view is enabled.

HIViewSetEnabled (page 2480)
Enables or disables a view.

`HIViewIsCompositingEnabled` (page 2468)
Determines whether compositing is enabled for a view.

`HIViewSetText` (page 2488)
Sets the text of a view to the specified string.

`HIViewCopyText` (page 2448)
Copies the text of a view.

`HIViewGetValue` (page 2465)
Obtains the value of a view.

`HIViewSetValue` (page 2489)
Sets the value of a view.

`HIViewGetMinimum` (page 2459)
Obtains the minimum value of a view.

`HIViewSetMinimum` (page 2484)
Sets a view's minimum value.

`HIViewGetMaximum` (page 2459)
Obtains a view's maximum value.

`HIViewSetMaximum` (page 2484)
Sets a view's maximum value.

`HIViewGetViewSize` (page 2467)
Obtains the view size of a view.

`HIViewSetViewSize` (page 2489)
Sets the view size of a view.

`HIViewIsValid` (page 2471)
Determines whether the specified view is known to the HIToolbox.

`HIViewGetID` (page 2457)
Obtains the HIViewID of a view.

`HIViewSetID` (page 2482)
Sets the `HIViewID` of a view.

`HIViewGetCommandID` (page 2453)
Obtains the command ID of a view.

`HIViewSetCommandID` (page 2479)
Sets the command ID of a view.

`HIViewGetKind` (page 2458)
Obtains the signature and kind of a view.

`HIViewGetAttributes` (page 2452)
Obtains the attributes for a view.

`HIViewChangeAttributes` (page 2444)
Changes the attributes of a view.

`HIViewGetNeedsDisplay` (page 2460)
Determines whether a view needs to be redrawn.

`HIViewSetNeedsDisplay` (page 2485)
Marks a view as needing or not needing to be redrawn.

`HIViewSetNeedsDisplayInRect` (page 2485)
Uses an `HIRect` to mark a portion of a view as needing or not needing to be redrawn.

`HIViewSetNeedsDisplayInShape` (page 2487)
Uses a shape to mark a portion of a view as needing or not needing to be redrawn.

`HIViewSetNeedsDisplayInRegion` (page 2486)
Uses a region to mark a portion of a view as needing or not needing to be redrawn.

`HIViewRender` (page 2475)
Renders the invalid portions of a view.

`HIViewGetSizeConstraints` (page 2463)
Returns the minimum and maximum size for a control.

`HIViewIsDrawingEnabled` (page 2469)
Determines if drawing is currently enabled for a view.

`HIViewSetDrawingEnabled` (page 2479)
Turns control drawing on or off.

`HIViewScrollRect` (page 2477)
Scrolls a view's contents, or a portion thereof.

`HIViewSetZOrder` (page 2490)
Changes the front-to-back ordering of sibling views.

`HIViewReshapeStructure` (page 2476)
Informs the system that the structure region of the given view has changed shape.

`HIViewRegionChanged` (page 2474)
Informs the system that a region of the view has changed.

`HIViewCopyShape` (page 2448)
Copies the shape of a part of a view.

`HIViewGetOptimalBounds` (page 2461)
Obtains the optimal size and text placement of a view.

`HIViewFlashDirtyArea` (page 2452)
Flashes a window's dirty area.

`HIViewGetWindow` (page 2467)
Obtains a reference to the window to which the specified view is bound.

`HIViewGetFeatures` (page 2454)
Obtains the features of the specified view.

`HIViewChangeFeatures` (page 2444)
Changes the features of a view.

`HIViewGetEventTarget` (page 2454)
Returns the `EventTargetRef` for the specified view.

Managing Focus

`HIViewGetFocusPart` (page 2455)
Obtains the part in the specified view that currently has focus.

`HIViewSetNextFocus` (page 2487)
Sets the view that is to receive keyboard focus when keyboard focus advances from the specified view.

`HIViewAdvanceFocus` (page 2442)
Advances the keyboard focus to the next most appropriate view.

HIViewSubtreeContainsFocus (page 2492)
Determines whether a view or any subviews have keyboard focus.

HIViewSetFirstSubViewFocus (page 2480)
Sets the subview that is first to receive keyboard focus.

Processing Events and Hit-Testing for Views

HIViewClick (page 2445)
Passes a mouse-down event to a view.

HIViewSimulateClick (page 2491)
Simulates a mouse click on a given view.

HIViewGetPartHit (page 2462)
Determines the part hit for a given point.

HIViewGetViewForMouseEvent (page 2466)
Returns the appropriate view to handle a mouse event.

HIViewGetSubviewHit (page 2464)
Returns the child of the given view hit by the point passed in.

Manipulating View Coordinates

HIViewGetBounds (page 2453)
Obtains the local bounds of a view.

HIViewSetBoundsOrigin (page 2478)
Sets the origin of the view.

HIViewGetFrame (page 2456)
Obtains the frame bounds of a view.

HIViewSetFrame (page 2481)
Sets the frame of a view.

HIViewMoveBy (page 2472)
Move a view by the specified distance relative to its current location.

HIViewConvertPoint (page 2446)
Converts a point's coordinates from one view to another.

HIViewConvertRect (page 2447)
Converts a rectangle from one view to another.

HIViewConvertRegion (page 2447)
Converts a region from one view to another.

Creating and Manipulating Combo Boxes

HIComboBoxCreate (page 2411)
Creates a combo box control.

HIComboBoxGetAttributes (page 2412)
Gets the attributes of a combo box.

`HIComboBoxChangeAttributes` (page 2410)
Changes the attributes of a combo box.

`HIComboBoxAppendTextItem` (page 2410)
Appends a text item to the combo box disclosure list.

`HIComboBoxCopyTextItemAtIndex` (page 2411)
Copy a text item from a combo box disclosure list

`HIComboBoxGetItemCount` (page 2413)
Gets the number of items in the combo box disclosure list.

`HIComboBoxInsertTextItemAtIndex` (page 2413)
Inserts a CFString in a combo box disclosure list.

`HIComboBoxRemoveItemAtIndex` (page 2414)
Removes an item from a combo box disclosure list.

`HIComboBoxIsListVisible` (page 2414)
Determines whether a combo box disclosure list is visible.

`HIComboBoxSetListVisible` (page 2415)
Hides or shows a combo box disclosure list.

Creating and Manipulating Image Views

`HIImageViewCreate` (page 2417)
Creates an image view.

`HIImageViewCopyImage` (page 2417)
Obtains the image for an image view.

`HIImageViewSetImage` (page 2419)
Sets the image to display in an image view.

`HIImageViewGetAlpha` (page 2418)
Obtains the alpha value for a view.

`HIImageViewSetAlpha` (page 2419)
Sets the alpha value for an image view.

`HIImageViewGetScaleToFit` (page 2418)
Determines whether an image will scale or clip to the view bounds.

`HIImageViewSetScaleToFit` (page 2421)
Specifies whether an image should scale or clip to the view's bounds.

`HIImageViewIsOpaque` (page 2419)
Determines whether an image view is opaque.

`HIImageViewSetOpaque` (page 2420)
Sets the opacity of an image view.

Creating and Manipulating Scroll Views

`HIScrollViewCreate` (page 2423)
Creates a scroll view.

HIScrollViewGetScrollBarAutoHide (page 2424)
Obtains current setting of a scroll view's scroll bar auto-hide setting.

HIScrollViewSetScrollBarAutoHide (page 2425)
Sets a scroll view's auto-hide setting.

HIScrollViewCanNavigate (page 2423)
Determines whether it is possible to navigate in a scroll view.

HIScrollViewNavigate (page 2424)
Changes the portion of a view's target.

Creating and Manipulating Layouts

HIViewGetLayoutInfo (page 2459)
Obtains the layout information of an view.

HIViewSetLayoutInfo (page 2482)
Sets the layout information of an HIView.

HIViewApplyLayout (page 2443)
Applies the current layout to the specified view.

HIViewSuspendLayout (page 2492)
Suspends layout handling for a view and its children.

HIViewResumeLayout (page 2476)
Resumes layout handling for a view and its children.

HIViewIsLayoutActive (page 2470)
Determines whether layout handling is active or suspended.

HIViewIsLayoutLatentlyActive (page 2471)
Determines whether layout handling is latently active or suspended.

Manipulating Tracking Areas

HIViewNewTrackingArea (page 2473)
Creates a new tracking area for a view.

HIViewChangeTrackingArea (page 2445)
Changes the shape of a tracking area.

HIViewGetTrackingAreaID (page 2465)
Obtains the ID of a tracking area.

HIViewDisposeTrackingArea (page 2450)
Disposes of an existing tracking area.

Creating and Manipulating Search Fields

HISearchFieldCreate (page 2427)
Creates a Search field control.

HISearchFieldSetSearchMenu (page 2429)
Sets the search menu associated with a search field view.

`HISearchFieldGetSearchMenu` (page 2428)
Obtains the search menu associated with a search field.

`HISearchFieldChangeAttributes` (page 2425)
Sets the attributes of a search field.

`HISearchFieldGetAttributes` (page 2428)
Obtains the attributes for a search field.

`HISearchFieldSetDescriptiveText` (page 2429)
Sets the description of the search action for a search field.

`HISearchFieldCopyDescriptiveText` (page 2426)
Obtains the description associated with a search field.

Manipulating Menus

`HIMenuViewGetMenu` (page 2422)
Returns the `MenuRef` associated with a view that is a subclass of HIMenuView.

`HIMenuGetContentView` (page 2421)
Obtains an `HIViewRef` that can be used to draw menu content for a menu.

Manipulating Segmented Views

`HISegmentedViewCreate` (page 2432)
Creates a segmented view.

`HISegmentedViewSetSegmentCount` (page 2439)
Sets the number of segments for a segmented view.

`HISegmentedViewGetSegmentCount` (page 2435)
Obtains the number of segments for a segmented view.

`HISegmentedViewSetSegmentBehavior` (page 2437)
Changes the behavior of an individual segment of a segmented view.

`HISegmentedViewGetSegmentBehavior` (page 2433)
Obtains the behavior of an individual segment of a segmented view.

`HISegmentedViewChangeSegmentAttributes` (page 2430)
Changes the attributes of an individual segment of a segmented view.

`HISegmentedViewGetSegmentAttributes` (page 2433)
Returns the attributes of an individual segment of a segmented view.

`HISegmentedViewSetSegmentValue` (page 2441)
Changes the value of an individual segment of a segmented view.

`HISegmentedViewGetSegmentValue` (page 2436)
Returns the value of an individual segment of a segmented view.

`HISegmentedViewSetSegmentEnabled` (page 2439)
Enables or disables an individual segment of a segmented view.

`HISegmentedViewIsSegmentEnabled` (page 2436)
Determines whether an individual segment of a segmented view is enabled.

HISegmentedViewSetSegmentCommand (page 2437)
Sets the command ID for a segment.

HISegmentedViewGetSegmentCommand (page 2434)
Obtains the command ID associated with a segment.

HISegmentedViewSetSegmentLabel (page 2440)
Sets the label string for a segment.

HISegmentedViewCopySegmentLabel (page 2431)
Obtains a copy of the label string associated with a segment.

HISegmentedViewSetSegmentContentWidth (page 2438)
Specifies how the content width of segment is to be calculated.

HISegmentedViewGetSegmentContentWidth (page 2434)
Obtains the content width of a segment.

HISegmentedViewSetSegmentImage (page 2440)
Sets or clears the image associated with a segment.

HISegmentedViewGetSegmentImageContentType (page 2435)
Obtains the type of image content drawn by a segment.

HISegmentedViewCopySegmentImage (page 2430)
Copies the image drawn by a segment.

Working with Core Graphics Images

HICreateTransformedCGImage (page 2415)
Creates a new Core Graphics image with the standard selected or disabled appearance.

HIViewCreateOffscreenImage (page 2449)
Creates a Core Graphics offscreen image of a view.

HIViewDrawCGImage (page 2450)
Draws a Core Graphics image appropriately for a view.

Working with Grow Boxes

HIGrowBoxViewIsTransparent (page 2416)
Determines whether a grow box view is transparent.

HIGrowBoxViewSetTransparent (page 2416)
Makes a grow box view transparent or opaque.

Using Cocoa Views in Carbon Windows

HICocoaViewCreate (page 2408)
Creates a Carbon view that serves as a wrapper for a Cocoa view.

HICocoaViewSetView (page 2409)
Associates a Cocoa view with a HICocoaView wrapper view.

HICocoaViewGetView (page 2409)
Returns the Cocoa view associated with an existing Carbon wrapper view.

Functions

HICocoaViewCreate

Creates a Carbon view that serves as a wrapper for a Cocoa view.

```
OSStatus HICocoaViewCreate (
    NSView *inNSView,
    OptionBits inOptions,
    HIViewRef *outHIView
);
```

Parameters

inNSView

A pointer to the Cocoa view you want to wrap. This function retains the Cocoa view you pass in; on output, you may safely release the view. If you want to create an empty Carbon wrapper view, you may pass `NULL`. An empty wrapper view does not draw or respond to user interaction; you can associate it with a Cocoa view at a later time using the function `HICocoaViewSetView` (page 2409).

inOptions

Options for the new Carbon wrapper view. Currently this parameter must be 0.

outHIView

A pointer to a variable of type `HIViewRef` (page 2497). On output, your variable contains a new Carbon view that serves as a wrapper for the Cocoa view specified in the `inNSView` parameter. You are responsible for releasing the wrapper view when you no longer need it. Note that if you embed the wrapper view in a Carbon window, the view (along with its associated Cocoa view) will be released automatically when the window is destroyed.

Return Value

An operating system result code. This function returns `paramErr` whenever the `inOptions` parameter is not 0 or the `outHIView` parameter is `NULL`.

Discussion

This function creates an HIView-based wrapper for a Cocoa view. You can embed the new wrapper view in a Carbon window and use standard HIView functions to manipulate the view. HICocoaView is supported only in compositing windows.

The following example shows how to use this function to create a wrapped Cocoa view that can be embedded in a Carbon window:

```
NSView *myCocoaView = [[SomeNSView alloc] init];
HIViewRef myHICocoaView;
HICocoaViewCreate (myCocoaView, 0, &myHICocoaView);
[myCocoaView release];
```

Availability

Available in Mac OS X v10.5 and later.

Not available to 64-bit applications.

Declared In

`HICocoaView.h`

HICocoaViewGetView

Returns the Cocoa view associated with an existing Carbon wrapper view.

```
NSView * HICocoaViewGetView (
    HIViewRef inHIView
);
```

Parameters

inHIView

A wrapper view that has an associated Cocoa view.

Return Value

The Cocoa view associated with the specified wrapper view, or `NULL` if the wrapper view is empty or invalid. If you need to save the Cocoa view for later use, you should retain it.

Availability

Available in Mac OS X v10.5 and later.

Not available to 64-bit applications.

Declared In

`HICocoaView.h`

HICocoaViewSetView

Associates a Cocoa view with a HICocoaView wrapper view.

```
OSStatus HICocoaViewSetView (
    HIViewRef inHIView,
    NSView *inNSView
);
```

Parameters

inHIView

An existing HICocoaView wrapper view.

inNSView

A pointer to a Cocoa view. This function retains the Cocoa view you pass in; on output, you may safely release this view. If the HICocoaView wrapper view specified in the `inHIView` parameter already wraps a Cocoa view, this function releases the wrapped view and replaces it with the Cocoa view you pass in.

Return Value

An operating system result code. This function returns `paramErr` if either parameter is `NULL` or invalid.

Discussion

Typically you'll use this function after you instantiate a nib-based Carbon window that contains an empty HICocoaView wrapper view. The empty wrapper view serves as a placeholder until you call this function to associate a Cocoa view with it. HICocoaView is supported only in compositing windows.

The following example shows how to use this function to associate a Cocoa view with an existing wrapper view:

```
NSView *myCocoaView = [[SomeNSView alloc] init];
HICocoaViewSetView (myHICocoaView, myCocoaView);
[myCocoaView release];
```

Availability
Available in Mac OS X v10.5 and later.
Not available to 64-bit applications.

Declared In
`HICocoaView.h`

HIComboBoxAppendTextItem

Appends a text item to the combo box disclosure list.

```
OSStatus HIComboBoxAppendTextItem (
   HIViewRef inComboBox,
   CFStringRef inText,
   CFIndex *outIndex
);
```

Parameters

inComboBox
The combo box having the disclosure list to which the text is to be appended.

inText
The text item to be appended to the combo box disclosure list.

outIndex
On exit, the index of the new item. Can be `NULL` if you don't want this information.

Return Value
An operating system status code.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`HIComboBox.h`

HIComboBoxChangeAttributes

Changes the attributes of a combo box.

```
OSStatus HIComboBoxChangeAttributes (
   HIViewRef inComboBox,
   OptionBits inAttributesToSet,
   OptionBits inAttributesToClear
);
```

Parameters

inComboBox
The combo box whose attributes you want to change.

inAttributesToSet
The attributes to set. For possible values, see "Combo Box Attributes" (page 2500).

inAttributesToClear

The attributes to clear. For possible values, see "Combo Box Attributes" (page 2500).

Return Value
An operating system status code.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`HIComboBox.h`

HIComboBoxCopyTextItemAtIndex

Copy a text item from a combo box disclosure list

```
OSStatus HIComboBoxCopyTextItemAtIndex (
   HIViewRef inComboBox,
   CFIndex inIndex,
   CFStringRef *outString
);
```

Parameters

inComboBox

The combo box that contains the text item you want to copy.

inIndex

The index of the text item. This function returns `paramErr` if the index is out of the bounds of the combo box disclosure list.

outString

A copy of the string at the specified index. You are responsible for releasing the string.

Return Value
An operating system status code.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`HIComboBox.h`

HIComboBoxCreate

Creates a combo box control.

```
OSStatus HIComboBoxCreate (
   const HIRect *boundsRect,
   CFStringRef text,
   const ControlFontStyleRec *style,
   CFArrayRef list,
   OptionBits inAttributes,
   HIViewRef *outComboBox
);
```

Parameters

boundsRect
The bounding box of the control.

text
The default text in the editable portion of the control. Can be `NULL`.

style
The font style of the both editable text and the text in the disclosure list. Can be `NULL`.

list
The default values available in the disclosure list. Can be `NULL`.

inAttributes
The default attributes of the combo box. For possible values, see "Combo Box Attributes" (page 2500).

outComboBox
On exit, a pointer to a reference for the new control.

Discussion
The combo box can be used in compositing mode, as well as traditional Control Manager mode. When created, this view is invisible. To see the view, you must show the view by calling `HIViewSetVisible` (page 2490).

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`HIComboBox.h`

HIComboBoxGetAttributes

Gets the attributes of a combo box.

```
OSStatus HIComboBoxGetAttributes (
   HIViewRef inComboBox,
   OptionBits *outAttributes
);
```

Parameters

inComboBox
The combo box whose attributes you want to obtain.

outAttributes
The attributes of the combo box. For possible values, see "Combo Box Attributes" (page 2500).

Return Value
An operating system status code.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`HIComboBox.h`

HIComboBoxGetItemCount

Gets the number of items in the combo box disclosure list.

```
ItemCount HIComboBoxGetItemCount (
   HIViewRef inComboBox
);
```

Parameters

inComboBox
The combo box.

Return Value
The number of items in the combo box disclosure list.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`HIComboBox.h`

HIComboBoxInsertTextItemAtIndex

Inserts a CFString in a combo box disclosure list.

```
OSStatus HIComboBoxInsertTextItemAtIndex (
   HIViewRef inComboBox,
   CFIndex inIndex,
   CFStringRef inText
);
```

Parameters

inComboBox
The combo box having the disclosure list in which the text is to be inserted.

inIndex
The index at which the text should be inserted. If the index does not fall within the number of items in the combo box list, the text is appended to the end of the list.

inText
The text item to be inserted in the combo box disclosure list.

Return Value
An operating system status code.

Availability
Available in Mac OS X v10.2 and later.

Not available to 64-bit applications.

Declared In
`HIComboBox.h`

HIComboBoxIsListVisible

Determines whether a combo box disclosure list is visible.

```
Boolean HIComboBoxIsListVisible (
   HIViewRef inComboBox
);
```

Parameters

inComboBox
The Combo box whose disclosure list visibility is to be queried.

Return Value
A Boolean whose value is `true` if the combo box disclosure list is visible; otherwise, `false` to indicate that the combo box disclosure list is hidden.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`HIComboBox.h`

HIComboBoxRemoveItemAtIndex

Removes an item from a combo box disclosure list.

```
OSStatus HIComboBoxRemoveItemAtIndex (
   HIViewRef inComboBox,
   CFIndex inIndex
);
```

Parameters

inComboBox
The combo box having the disclosure list that from which you want to remove an item.

inIndex
The index of the item to remove.

Return Value
An operating system status code.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`HIComboBox.h`

HIComboBoxSetListVisible

Hides or shows a combo box disclosure list.

```
OSStatus HIComboBoxSetListVisible (
   HIViewRef inComboBox,
   Boolean inVisible
);
```

Parameters

inComboBox

The combo box.

inVisible

A Boolean whose value is `true` to make the combo box disclosure list visible or `false` to hide the combo box list.

Return Value

An operating system result code.

Availability

Available in Mac OS X v10.4 and later.

Not available to 64-bit applications.

Declared In

`HIComboBox.h`

HICreateTransformedCGImage

Creates a new Core Graphics image with the standard selected or disabled appearance.

```
OSStatus HICreateTransformedCGImage (
   CGImageRef inImage,
   OptionBits inTransform,
   CGImageRef *outImage
);
```

Parameters

inImage

The original image.

inTransform

The transformation to apply to the image. For possible values, see "Transformation Constants" (page 2525).

outImage

A pointer to a value of type `CGImageRef` that, on return, contains the new image. You are responsible for releasing the image.

Return Value

An operating system result code.

Availability

Available in Mac OS X v10.4 and later.

Not available to 64-bit applications.

Declared In
HIView.h

HIGrowBoxViewIsTransparent

Determines whether a grow box view is transparent.

```
Boolean HIGrowBoxViewIsTransparent (
   HIViewRef inGrowBoxView
);
```

Parameters

inGrowBoxView
The grow box view reference to query.

Return Value
A Boolean value that is `true` if the grow box view is transparent; otherwise, `false` which indicates the grow box view is an opaque white square with grow lines.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
HIWindowViews.h

HIGrowBoxViewSetTransparent

Makes a grow box view transparent or opaque.

```
OSStatus HIGrowBoxViewSetTransparent (
   HIViewRef inGrowBoxView,
   Boolean inTransparent
);
```

Parameters

inGrowBoxView
The grow box view reference.

inTransparent
Pass `true` to make the grow box view use its transparent look or `false` to make the grow box view use its opaque appearance.

Return Value
An operating system result code.

Discussion
This function sets a grow box view to be transparent, meaning the grow box lines are drawn over any content under it. When not transparent, the grow box is an opaque white square with the grow lines.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
HIWindowViews.h

HIImageViewCopyImage

Obtains the image for an image view.

```
CGImageRef HIImageViewCopyImage (
   HIViewRef inView
);
```

Parameters

inView
The image view to query.

Return Value
A Core Graphics (Quartz) image reference, or NULL if there is no image set on the view or if the view ref is invalid.

Discussion
The image is retained, so you should release it when you are finished with it.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
HIImageViews.h

HIImageViewCreate

Creates an image view.

```
OSStatus HIImageViewCreate (
   CGImageRef inImage,
   HIViewRef *outView
);
```

Parameters

inImage
An initial image, or NULL. You can use SetControlData to set the image later.

outControl
The new image view.

Return Value
An operating system result code.

Discussion
The view responds to the scrollable interface and can be used in a scrolling view. You can pass an image initially, or set one later.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
HIImageViews.h

HIImageViewGetAlpha

Obtains the alpha value for a view.

```
CGFloat HIImageViewGetAlpha (
   HIViewRef inView
);
```

Parameters

inView
The image view to query.

Return Value
A floating point number representing the alpha from 0.0 through 1.0.

Discussion
An alpha of 1.0 means that the view is fully opaque, and alpha of 0.0 is means the view is fully transparent.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
HIImageViews.h

HIImageViewGetScaleToFit

Determines whether an image will scale or clip to the view bounds.

```
Boolean HIImageViewGetScaleToFit (
   HIViewRef inView
);
```

Parameters

inView
The image view to query.

Return Value
A Boolean whose value is `true` if the image scales or `false` if the image clips.

Discussion
This function determines whether an image view will scale the image it displays to the view bounds or merely clip to the view bounds.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
HIImageViews.h

HIImageViewIsOpaque

Determines whether an image view is opaque.

```
Boolean HIImageViewIsOpaque (
   HIViewRef inView
);
```

Parameters

inView
The image view to query.

Return Value

A Boolean whose value is `true` if the image view is opaque; otherwise, `false`.

Availability

Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In

`HIImageViews.h`

HIImageViewSetAlpha

Sets the alpha value for an image view.

```
OSStatus HIImageViewSetAlpha (
   HIViewRef inView,
   CGFloat inAlpha
);
```

Parameters

inView
The image view to affect.

inAlpha
The new alpha value.

Return Value

An operating system result code.

Discussion

Allows you to set the alpha for an image, making it more or less transparent. An alpha of 1.0 is fully opaque, and an alpha of 0.0 is fully transparent. The default alpha for an image is 1.0.

Availability

Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In

`HIImageViews.h`

HIImageViewSetImage

Sets the image to display in an image view.

```
OSStatus HIImageViewSetImage (
   HIViewRef inView,
   CGImageRef inImage
);
```

Parameters

inView

The image view to affect.

inImage

The image to set.

Return Value
An operating system status code.

Discussion
The image passed in is retained by the view, so you may release the image after calling this function if you no longer need to reference it.

Availability
Available in Mac OS X v10.2 and later.

Not available to 64-bit applications.

Related Sample Code
CarbonCocoa_PictureCursor

Declared In
`HIImageViews.h`

HIImageViewSetOpaque

Sets the opacity of an image view.

```
OSStatus HIImageViewSetOpaque (
   HIViewRef inView,
   Boolean inOpaque
);
```

Parameters

inView

The image view to set.

inOpaque

A Boolean whose value is `true` to make the image view opaque or `false` to disable the opacity setting.

Return Value
An operating system result code.

Discussion
When opacity is enabled, the image view can make certain optimizations for compositing and scrolling. The alpha-related image view APIs are rendered useless when opacity is enabled. An image view, when created, is opaque by default. You must pass `false` to this function in order to change the alpha, etc. or if your image does not fill the full bounds of the view.

Availability
Available in Mac OS X v10.2 and later.

Not available to 64-bit applications.

Related Sample Code
CarbonCocoa_PictureCursor

Declared In
`HIImageViews.h`

HIImageViewSetScaleToFit

Specifies whether an image should scale or clip to the view's bounds.

```
OSStatus HIImageViewSetScaleToFit (
   HIViewRef inView,
   Boolean inScaleToFit
);
```

Parameters

inView
The image view.

inScaleToFit
A Boolean whose value is `true` to indicate that the image should be scaled to fit the view bounds or `false` to indicate that the image should clip to the view's bounds.

Return Value
An operating system status code.

Discussion
Normally, an image view clips to the view's bounds. Use this function to tell the image view to size the image to fit into the view's bounds.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`HIImageViews.h`

HIMenuGetContentView

Obtains an `HIViewRef` that can be used to draw menu content for a menu.

```
OSStatus HIMenuGetContentView (
   MenuRef inMenu,
   ThemeMenuType inMenuType,
   HIViewRef *outView
);
```

Parameters

inMenu
The menu for which an `HIViewRef` is to be obtained.

inMenuType

The type of menu for which the menu content view is to be returned. The same `MenuRef` may have multiple content views, depending on the menu type being displayed.

outView

A pointer to a value of type `HIViewRef` that, on return, represents the view, or `NULL` if the menu does not use an HIView to draw its content. The caller should not release this view.

Return Value
An operating system result code. If the menu uses an MDEF instead of a view to draw its content, this function sets `outView` to `NULL` and returns `noErr`.

Discussion
If the content view has not yet been created, the Menu Manager will create the content view using the view class ID and initialization event associated with the menu. Note that the menu content view is not the same as the window content view; the menu content view is embedded inside the window content view.

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In
`Menus.h`

HIMenuViewGetMenu

Returns the `MenuRef` associated with a view that is a subclass of HIMenuView.

```
MenuRef HIMenuViewGetMenu (
   HIViewRef inView
);
```

Parameters

inView

The view whose menu is to be returned.

Return Value
The `MenuRef` associated with the specified view, or `NULL` if a view is passed that is not a subclass of HIMenuView.

Discussion
An HIMenuView subclass might use call this function to determine the menu it should draw.

Special Considerations

Prior to Mac OS X v10.5, this function returns `NULL` if passed an instance of the standard menu view. In Mac OS X v10.5 and later, this function correctly returns the owning menu of an instance of the standard menu view.

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In
`HIMenuView.h`

HIScrollViewCanNavigate

Determines whether it is possible to navigate in a scroll view.

```
Boolean HIScrollViewCanNavigate (
   HIViewRef inView,
   HIScrollViewAction inAction
);
```

Parameters

inView

The view to query.

inAction

The navigation action to test. For possible values, see "Scroll View Action Constants" (page 2515).

Return Value

A Boolean whose value is `true` if the navigation specified by `inAction` is possible; otherwise, `false`.

Discussion

Use this function to determine whether it is possible to perform a particular navigation within a scroll view. For example, if a scroll view is already at the top of the scrollable content, it is not possible to navigate upward, so the home and page up actions would not be possible. You might use this function to help you update the state of menu items.

Availability

Available in Mac OS X v10.3 and later.

Not available to 64-bit applications.

Declared In

`HIScrollView.h`

HIScrollViewCreate

Creates a scroll view.

```
OSStatus HIScrollViewCreate (
   OptionBits inOptions,
   HIViewRef *outView
);
```

Parameters

inOptions

Options for our scroll view. You must specify either a horizontal or a vertical scroll bar. If neither is passed, an error is returned. For possible values, see "Scroll View Constants" (page 2514).

outView

The new scroll view.

Return Value

An operating system result code.

Discussion

This view has three parts. It can have a horizontal scroll bar, a vertical scroll bar, and a view to be scrolled that must be added by calling `HIViewAddSubview` (page 2442). The added scroll view integrates itself automatically and appropriately.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Related Sample Code
HID Calibrator

Declared In
`HIScrollView.h`

HIScrollViewGetScrollBarAutoHide

Obtains current setting of a scroll view's scroll bar auto-hide setting.

```
Boolean HIScrollViewGetScrollBarAutoHide (
   HIViewRef inView
);
```

Parameters

inView
The view to query.

Return Value
A Boolean whose value is `true` if the auto-hide setting is enabled; otherwise, `false`.

Discussion
When the auto-hide setting is enabled, a scroll view's scroll bars are hidden when the entire scrollable view can be fully displayed in the scroll view's bounds. This is similar to the behavior of the Preview application.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`HIScrollView.h`

HIScrollViewNavigate

Changes the portion of a view's target.

```
OSStatus HIScrollViewNavigate (
   HIViewRef inView,
   HIScrollViewAction inAction
);
```

Parameters

inView
The view to scroll.

inAction
The action to take.

Return Value
An operating system result code.

Discussion
Use this function to programmatically change the portion of a scroll view's target. For example, you can call this function to move to the beginning or end of a document. You can also page up, down, left and right. In general, you should not call this function from embedded content, that is, the scrollable view inside the scroll view. Instead, for those cases, you should position yourself appropriately and tell the scroll view you changed via the `kEventScrollableInfoChanged` Carbon event. This function is merely a programmatic way to scroll as one would by hand using the scroll bars.

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In
`HIScrollView.h`

HIScrollViewSetScrollBarAutoHide

Sets a scroll view's auto-hide setting.

```
OSStatus HIScrollViewSetScrollBarAutoHide (
   HIViewRef inView,
   Boolean inAutoHide
);
```

Parameters

inView
The view to affect.

inAutoHide
A Boolean whose value is `true` to enable auto-hide and `false` to disable auto-hide.

Return Value
An operating system result code.

Discussion
When the auto-hide setting is enabled, a scroll view's scroll bars are hidden when the entire scrollable view can be fully displayed in the scroll view's bounds. This is similar to the behavior of the Preview application.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Related Sample Code
HID Calibrator

Declared In
`HIScrollView.h`

HISearchFieldChangeAttributes

Sets the attributes of a search field.

```
OSStatus HISearchFieldChangeAttributes (
   HIViewRef inSearchField,
   OptionBits inAttributesToSet,
   OptionBits inAttributesToClear
);
```

Parameters

inSearchField

The search field whose attributes are to be changed.

inAttributesToSet

The attributes to set. For possible values, see "Search Field Attribute Constants" (page 2519).

inAttributesToClear

The attributes to clear. For possible values, see "Search Field Attribute Constants" (page 2519).

Return Value

An operating system result code.

Availability

Available in Mac OS X v10.3 and later.

Not available to 64-bit applications.

Declared In

`HISearchField.h`

HISearchFieldCopyDescriptiveText

Obtains the description associated with a search field.

```
OSStatus HISearchFieldCopyDescriptiveText (
   HIViewRef inSearchField,
   CFStringRef *outDescription
);
```

Parameters

inSearchField

The search field whose descriptive text is to be obtained.

outDescription

A pointer to a value of type `CFStringRef` that, on return, represents the description that is associated with the search field specified by `inSearchField`. This parameter cannot be `NULL`. If no description is associated with the search field, on return, `outDescription` is set to `NULL`. If there is a description, a `CFStringRef` is created for you; you are responsible for releasing the `CFStringRef` when you no longer need it.

Return Value

An operating system result code.

Availability

Available in Mac OS X v10.3 and later.

Not available to 64-bit applications.

Declared In

`HISearchField.h`

HISearchFieldCreate

Creates a Search field control.

```
OSStatus HISearchFieldCreate (
   const HIRect *inBounds,
   OptionBits inAttributes,
   MenuRef inSearchMenu,
   CFStringRef inDescriptiveText,
   HIViewRef *outRef
);
```

Parameters

inBounds

The initial bounds of the view, or `NULL`. If this parameter is `NULL`, the view is created with empty bounds (0, 0, 0, 0).

inAttributes

The initial attributes of the search field. Indicates whether the field should contain the Cancel icon. For possible values, see "Search Field Attribute Constants" (page 2519).

inSearchMenu

The menu to be associated with this search field, or `NULL`. If `inSearchMenu` is non-`NULL`, the menu will be retained by the search field and the Search icon will be enabled in the left side of the text field. If this parameter is `NULL`, the view will not display the Search icon in the left portion of the text field. You are expected to install handlers on this menu to handle the visual appearance of the menu (for example, to draw check marks or enable items when the menu receives the `kEventMenuEnableItems` Carbon event), and to keep track of what action should be performed by associating HICommands with each menu item and installing a handler for the `kEventClassCommand`/`kEventCommandProcess` Carbon event.

inDescriptiveText

The text to be displayed in the text field when the field does not have focus and contains no user-entered text, or `NULL`. This text should indicate the search criteria. For example, you may want to identify to the user that the search will cover the "Subject" or "Contents" of a selected range of items. If `inDescriptiveText` is non-`NULL`, the text will be retained by the search field.

outRef

On return, a reference for the new view.

Return Value

An operating system result code.

Discussion

This view is designed to be used by applications that provide searching functionality. Visually, it is a standard text field optionally adorned with a Search icon on the left and a Cancel image on the right. The new control is initially invisible.

When the user accepts the text by pressing the Return or Enter key, a Carbon event of `kEventClassTextField` / `kEventTextAccepted` is sent to the control to indicate that the search should begin. This control also responds to all of the standard control tags used by the `EditUnicodeText` control.

Availability

Available in Mac OS X v10.3 and later.

Not available to 64-bit applications.

Declared In

`HISearchField.h`

HISearchFieldGetAttributes

Obtains the attributes for a search field.

```
OSStatus HISearchFieldGetAttributes (
   HIViewRef inSearchField,
   OptionBits *outAttributes
);
```

Parameters

inSearchField

The search field whose attributes are to be obtained.

outAttributes

A pointer to a value of type `OptionBits` that, on return, contains the attributes of the Search field specified by `inSearchField`. This parameter cannot be `NULL`. For possible values, see "Search Field Attribute Constants" (page 2519).

Return Value

An operating system result code.

Availability

Available in Mac OS X v10.3 and later.

Not available to 64-bit applications.

Declared In

`HISearchField.h`

HISearchFieldGetSearchMenu

Obtains the search menu associated with a search field.

```
OSStatus HISearchFieldGetSearchMenu (
   HIViewRef inSearchField,
   MenuRef *outSearchMenu
);
```

Parameters

inSearchField

The search field for which you want to obtain the search menu.

outSearchMenu

A pointer to a value of type `MenuRef` that, on return, represents the menu associated with the search field. The menu is *not* retained on output, and this parameter cannot be `NULL`.

Return Value

An operating system result code.

Availability

Available in Mac OS X v10.3 and later.

Not available to 64-bit applications.

Declared In

`HISearchField.h`

HISearchFieldSetDescriptiveText

Sets the description of the search action for a search field.

```
OSStatus HISearchFieldSetDescriptiveText (
   HIViewRef inSearchField,
   CFStringRef inDescription
);
```

Parameters

inSearchField

The search field whose descriptive text is to be set.

inDescription

The new description for the search field. If the search field already has a description, it will be released. If this parameter is non-`NULL`, it will be retained by the search field. If this parameter is `NULL`, upon return, no description is associated with the search field.

Return Value

An operating system result code.

Availability

Available in Mac OS X v10.3 and later.

Not available to 64-bit applications.

Declared In

`HISearchField.h`

HISearchFieldSetSearchMenu

Sets the search menu associated with a search field view.

```
OSStatus HISearchFieldSetSearchMenu (
   HIViewRef inSearchField,
   MenuRef inSearchMenu
);
```

Parameters

inSearchField

The Search field with which to associate the search menu.

inSearchMenu

The menu to associate with the Search field, or `NULL`. If a menu is already associated with the Search field, that menu is released. If `inSearchMenu` is non-`NULL`, it will be retained by the Search field and the Search icon will be enabled in the left side of the text field. You are expected to install handlers on this menu to handle the visual appearance of the menu (for example, to draw check marks or enable items when the menu receives the `kEventMenuEnableItems` Carbon event). You are also expected to keep track of the actions that should be performed by associating HICommands with each menu item and installing a handler for the `kEventClassCommand`/`kEventCommandProcess` Carbon event. If `inSearchMenu` is `NULL`, the Search icon is removed from the left side of the text field and no menu is associated with the Search field.

Return Value

An operating system result code.

Availability

Available in Mac OS X v10.3 and later.

Not available to 64-bit applications.

Declared In
HISearchField.h

HISegmentedViewChangeSegmentAttributes

Changes the attributes of an individual segment of a segmented view.

```
OSStatus HISegmentedViewChangeSegmentAttributes (
   HIViewRef inSegmentedView,
   UInt32 inSegmentIndexOneBased,
   OptionBits inAttributesToSet,
   OptionBits inAttributesToClear
);
```

Parameters

inSegmentedView
The segmented view that owns the segment whose attributes you want to change.

inSegmentIndexOneBased
The one-based index of the segment whose attributes you want to change. This must be a non-zero value that is no higher than the segmented view's current segment count.

inAttributesToSet
The attribute bits you want to set for the segment. For possible values, see "Segment Attribute Constants" (page 2520).

inAttributesToClear
The attribute bits you want to clear for the segment.

Return Value
An operating system result code.

Discussion
By default, a segment has no attributes.

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In
HISegmentedView.h

HISegmentedViewCopySegmentImage

Copies the image drawn by a segment.

```
OSStatus HISegmentedViewCopySegmentImage (
   HIViewRef inSegmentedView,
   UInt32 inSegmentIndexOneBased,
   HIViewImageContentInfo *ioImage
);
```

Parameters

inSegmentedView

The segmented view that owns the segment whose image you want to copy.

inSegmentIndexOneBased

The one-based index of the segment whose image you want to set. This must be a non-zero value that is no higher than the segmented view's current segment count.

ioImage

A pointer to a `HIViewImageContentInfo` structure whose `contentType` field specifies the type of image you want to copy. If the segment uses the type of image you specified, on return, the appropriate field of the union contains a copy of the image. If the segment index is an illegal value, the result is undefined. You are responsible for releasing the image.

Return Value

An operating system result code.

Discussion

Call `HISegmentedViewGetSegmentImageContentType` (page 2435) to get the image type.

Availability

Available in Mac OS X v10.3 and later.

Not available to 64-bit applications.

Declared In

`HISegmentedView.h`

HISegmentedViewCopySegmentLabel

Obtains a copy of the label string associated with a segment.

```
OSStatus HISegmentedViewCopySegmentLabel (
   HIViewRef inSegmentedView,
   UInt32 inSegmentIndexOneBased,
   CFStringRef *outLabel
);
```

Parameters

inSegmentedView

The segmented view that owns the segment whose label you want to copy.

inSegmentIndexOneBased

The one-based index of the segment whose label is to be copied. This must be a non-zero value that is no higher than the segmented view's current segment count.

outLabel

A pointer to a `CFStringRef` that represents the copy of the label associated with the segment. You are responsible for releasing the the string containing the copy of the label. If no label is associated with the specified segment, `outLabel` is set to `NULL`.

Return Value
An operating system result code.

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In
`HISegmentedView.h`

HISegmentedViewCreate

Creates a segmented view.

```
OSStatus HISegmentedViewCreate (
   const HIRect *inBounds,
   HIViewRef *outRef
);
```

Parameters

inBounds
The bounds of the view to be created, or `NULL`. If `NULL`, the view is created with `CGRectZero` bounds.

outRef
A valid pointer to an `HIViewRef` that, on return, represents the newly created view.

Return Value
An operating system result code.

Discussion
You can use a segmented view to implement the icon/column/list view switcher as seen in the Finder. After creating a segmented view, set the number of segments by calling `HISegmentedViewSetSegmentCount` (page 2439). Each segment can be configured independently by calling other HISegmentedView APIs. Changing the number of segments and configuring each segment changes the appearance of the segmented view. After configuring the view, you may want to call `HIViewGetOptimalBounds` (page 2461) on the view and resize it so the content fits optimally.

The value of the whole segmented view corresponds to the index of the currently selected segment with the radio behavior. If you set the value of the whole segmented view to *n* by calling `HIViewSetValue` (page 2489), the value of each radio-behavior segment is set to zero except for the segment at index *n*. If segment *n* also has radio behavior, its value will be set to one. When a radio-behavior segment is clicked, the value of the whole segmented view is set to the segment's index.

Segmented views work in both compositing and non-compositing modes.

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In
`HISegmentedView.h`

HISegmentedViewGetSegmentAttributes

Returns the attributes of an individual segment of a segmented view.

```
OptionBits HISegmentedViewGetSegmentAttributes (
   HIViewRef inSegmentedView,
   UInt32 inSegmentIndexOneBased
);
```

Parameters

inSegmentedView

The segmented view that owns the segment to query.

inSegmentIndexOneBased

The one-based index of the segment whose attributes you want to query. This must be a non-zero value that is no higher than the segmented view's current segment count.

Return Value

The attribute bits that are set for the specified segment. For possible values, see "Segment Attribute Constants" (page 2520). If the segment index is an illegal value, the result is undefined.

Availability

Available in Mac OS X v10.3 and later.

Not available to 64-bit applications.

Declared In

`HISegmentedView.h`

HISegmentedViewGetSegmentBehavior

Obtains the behavior of an individual segment of a segmented view.

```
HISegmentBehavior HISegmentedViewGetSegmentBehavior (
   HIViewRef inSegmentedView,
   UInt32 inSegmentIndexOneBased
);
```

Parameters

inSegmentedView

The segmented view that owns the segment whose behavior is to be obtained.

inSegmentIndexOneBased

The one-based index of the segment whose behavior you want to obtain. This must be a non-zero value that is no higher than the segmented view's current segment count.

Return Value

A value of type `HISegmentBehavior` describing the behavior of the given segment. For possible values, see "Segment Behavior Constants" (page 2521). If the segment index is an illegal value, the result is undefined.

Availability

Available in Mac OS X v10.3 and later.

Not available to 64-bit applications.

Declared In

`HISegmentedView.h`

HISegmentedViewGetSegmentCommand

Obtains the command ID associated with a segment.

```
UInt32 HISegmentedViewGetSegmentCommand (
   HIViewRef inSegmentedView,
   UInt32 inSegmentIndexOneBased
);
```

Parameters

inSegmentedView

The segmented view that owns the segment for which the command ID is to be obtained.

inSegmentIndexOneBased

The one-based index of the segment for which the command ID is to be obtained. This must be a non-zero value that is no higher than the segmented view's current segment count.

Return Value

The command ID associated with the specified segment. If the segment index is an illegal value, the result is undefined.

Availability

Available in Mac OS X v10.3 and later.

Not available to 64-bit applications.

Declared In

`HISegmentedView.h`

HISegmentedViewGetSegmentContentWidth

Obtains the content width of a segment.

```
CGFloat HISegmentedViewGetSegmentContentWidth (
   HIViewRef inSegmentedView,
   UInt32 inSegmentIndexOneBased,
   Boolean *outAutoCalculated
);
```

Parameters

inSegmentedView

The segmented view that owns the segment whose content width is to be obtained.

inSegmentIndexOneBased

The one-based index of the segment. This must be a non-zero value that is no higher than the segmented view's current segment count.

outAutoCalculateWidth

A pointer to a Boolean whose value, on return, is `true` if the segment calculates its own width automatically; otherwise, `false`. Pass `NULL` if you don't want this information.

Return Value

The width of the content for the given segment. If the segment index is an illegal value, the result is undefined.

Discussion

The content width is the horizontal area taken up by a segment's label (if any) and image (if any).

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In
`HISegmentedView.h`

HISegmentedViewGetSegmentCount

Obtains the number of segments for a segmented view.

```
UInt32 HISegmentedViewGetSegmentCount (
   HIViewRef inSegmentedView
);
```

Parameters

`inSegmentedView`
The segmented view for which the number of segments is to be obtained.

Return Value
A `UInt32` whose value is the number of segments in the segmented view specified by `inSegmentedView`.

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In
`HISegmentedView.h`

HISegmentedViewGetSegmentImageContentType

Obtains the type of image content drawn by a segment.

```
HIViewImageContentType HISegmentedViewGetSegmentImageContentType (
   HIViewRef inSegmentedView,
   UInt32 inSegmentIndexOneBased
);
```

Parameters

`inSegmentedView`
The segmented view that owns the segment whose image content type you want to obtain.

`inSegmentIndexOneBased`
The one-based index of the segment whose image you want to set. This must be a non-zero value that is no higher than the segmented view's current segment count.

Return Value
The image content type of the image drawn by the specified segment. If the segment index is an illegal value, the result is undefined.

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In
HISegmentedView.h

HISegmentedViewGetSegmentValue

Returns the value of an individual segment of a segmented view.

```
SInt32 HISegmentedViewGetSegmentValue (
   HIViewRef inSegmentedView,
   UInt32 inSegmentIndexOneBased
);
```

Parameters

inSegmentedView
The segmented view that owns the segment to query.

inSegmentIndexOneBased
The one-based index of the segment whose value you want to obtain. This must be a non-zero value that is no higher than the segmented view's current segment count.

Return Value
An `SInt32` containing the value of the specified segment. Zero means that the segment is unpressed or unselected, and one means the segment is pressed or selected.

Discussion
Getting a segment value is only meaningful for segments with the sticky, toggles, or radio behaviors. The value of segments that have the momentary behavior is undefined.

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In
HISegmentedView.h

HISegmentedViewIsSegmentEnabled

Determines whether an individual segment of a segmented view is enabled.

```
Boolean HISegmentedViewIsSegmentEnabled (
   HIViewRef inSegmentedView,
   UInt32 inSegmentIndexOneBased
);
```

Parameters

inSegmentedView
The segmented view that owns the segment to query.

inSegmentIndexOneBased
The one-based index of the segment to query. This must be a non-zero value that is no higher than the segmented view's current segment count.

Return Value
A Boolean whose value is `true` if the segment is enabled or `false` if the segment is not enabled.

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In
`HISegmentedView.h`

HISegmentedViewSetSegmentBehavior

Changes the behavior of an individual segment of a segmented view.

```
OSStatus HISegmentedViewSetSegmentBehavior (
   HIViewRef inSegmentedView,
   UInt32 inSegmentIndexOneBased,
   HISegmentBehavior inBehavior
);
```

Parameters

inSegmentedView
The segmented view that owns the segment whose behavior is to be set.

inSegmentIndexOneBased
The one-based index of the segment whose behavior you want to set. This must be a non-zero value that is no higher than the segmented view's current segment count.

inBehavior
The behavior you want the segment to have. For possible values, see "Segment Behavior Constants" (page 2521).

Return Value
An operating system result code.

Discussion
By default, a segment has the `kHISegmentBehaviorMomentary` behavior.

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In
`HISegmentedView.h`

HISegmentedViewSetSegmentCommand

Sets the command ID for a segment.

```
OSStatus HISegmentedViewSetSegmentCommand (
   HIViewRef inSegmentedView,
   UInt32 inSegmentIndexOneBased,
   UInt32 inCommand
);
```

Parameters

inSegmentedView
The segmented view that owns the segment for which the command ID is to be set.

inSegmentIndexOneBased

The one-based index of the segment for which the command ID is to be set. This must be a non-zero value that is no higher than the segmented view's current segment count.

inCommand

The command ID you want to associate with the segment. When the command ID is `0` for a segment, the `kEventCommandProcess` event is not sent when the segment is clicked.

Return Value
An operating system result code.

Discussion
When any non-zero command ID is set, the segmented view sends an HICommand whenever the segment is clicked. By default, the command is sent to the segmented view and up the containment hierarchy. If you want the command to start at the user focus instead, set the `kHISegmentCmdToUserFocus` attribute for the segment.

Availability
Available in Mac OS X v10.3 and later.

Not available to 64-bit applications.

Declared In
`HISegmentedView.h`

HISegmentedViewSetSegmentContentWidth

Specifies how the content width of segment is to be calculated.

```
OSStatus HISegmentedViewSetSegmentContentWidth (
   HIViewRef inSegmentedView,
   UInt32 inSegmentIndexOneBased,
   Boolean inAutoCalculateWidth,
   CGFloat inWidth
);
```

Parameters

inSegmentedView

The segmented view that owns the segment whose method of calculating content width you want to specify.

inSegmentIndexOneBased

The one-based index of the segment. This must be a non-zero value that is no higher than the segmented view's current segment count.

inAutoCalculateWidth

A Boolean whose value is `true` if you want the segment to calculate its own width automatically (in which case, the `inWidth` parameter is ignored), or `false` if you want the value of the `inWidth` parameter to be associated with the segment.

inWidth

The width in pixels.

Return Value
An operating system result code.

Discussion
The content width is the horizontal area taken up by a segment's label (if any) and image (if any).

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In
`HISegmentedView.h`

HISegmentedViewSetSegmentCount

Sets the number of segments for a segmented view.

```
OSStatus HISegmentedViewSetSegmentCount (
   HIViewRef inSegmentedView,
   UInt32 inSegmentCount
);
```

Parameters

inSegmentedView
The segmented menu for which the number of segments is to be set.

inSegmentCount
A positive integer specifying the number of segments the view is to have.

Return Value
An operating system result code.

Discussion
The content for any previous segments beyond the new count is released. All new segments beyond the previous count are initialized with basic settings: Momentary, no attributes, zero value, enabled, no command, no label, no content, and auto-calculated content width. You should configure any new segments to match your needs.

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In
`HISegmentedView.h`

HISegmentedViewSetSegmentEnabled

Enables or disables an individual segment of a segmented view.

```
OSStatus HISegmentedViewSetSegmentEnabled (
   HIViewRef inSegmentedView,
   UInt32 inSegmentIndexOneBased,
   Boolean inEnabled
);
```

Parameters

inSegmentedView
The segmented view that owns the segment that is to be enabled or disabled.

inSegmentIndexOneBased

The one-based index of the segment to enable or disable. This must be a non-zero value that is no higher than the segmented view's current segment count.

inEnabled

A Boolean whose value is `true` to enable the segment or `false` to disable the segment.

Return Value
An operating system result code.

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In
`HISegmentedView.h`

HISegmentedViewSetSegmentImage

Sets or clears the image associated with a segment.

```
OSStatus HISegmentedViewSetSegmentImage (
   HIViewRef inSegmentedView,
   UInt32 inSegmentIndexOneBased,
   const HIViewImageContentInfo *inImage
);
```

Parameters

inSegmentedView

The segmented view that owns the segment whose image you want to set.

inSegmentIndexOneBased

The one-based index of the segment whose image you want to set. This must be a non-zero value that is no higher than the segmented view's current segment count.

inImage

A pointer to an `HIViewImageContentInfo` structure with the image information for the specified segment. Segments support three types of image content: `kControlNoContent`, `kControlContentIconRef`, and `kControlContentCGImageRef`.

Return Value
An operating system result code.

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In
`HISegmentedView.h`

HISegmentedViewSetSegmentLabel

Sets the label string for a segment.

```
OSStatus HISegmentedViewSetSegmentLabel (
   HIViewRef inSegmentedView,
   UInt32 inSegmentIndexOneBased,
   CFStringRef inLabel
);
```

Parameters

inSegmentedView

The segmented view that owns the segment whose label you want to sets.

inSegmentIndexOneBased

The one-based index of the segment whose label is to be set. This must be a non-zero value that is no higher than the segmented view's current segment count.

inLabel

A `CFStringRef` with the text of the label. The segmented view will copy the string you passed in. To eliminate the label from the segment, pass `NULL` or an empty `CFStringRef`.

Return Value
An operating system result code.

Discussion
By default, a segment has no label string.

Availability
Available in Mac OS X v10.3 and later.

Not available to 64-bit applications.

Declared In
`HISegmentedView.h`

HISegmentedViewSetSegmentValue

Changes the value of an individual segment of a segmented view.

```
OSStatus HISegmentedViewSetSegmentValue (
   HIViewRef inSegmentedView,
   UInt32 inSegmentIndexOneBased,
   SInt32 inValue
);
```

Parameters

inSegmentedView

The segmented view that owns the segment to query.

inSegmentIndexOneBased

The one-based index of the segment whose value you want to change. This must be a non-zero value that is no higher than the segmented view's current segment count.

inValue

The value you want to set. Zero means that the segment is unpressed or unselected, and one means the segment is pressed or selected. Setting any other value will result in an undefined behavior.

Return Value
An operating system result code.

Discussion
Setting a segment value is only meaningful for segments with the sticky, toggles, or radio behaviors. Setting the value of segments that have the momentary behavior to something other than zero results in a behavior that is undefined.

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In
`HISegmentedView.h`

HIViewAddSubview

Adds a subview to the given parent view.

```
OSStatus HIViewAddSubview (
   HIViewRef inParent,
   HIViewRef inNewChild
);
```

Parameters

inParent
The view that will receive the new subview.

inNewChild
The subview being added.

Return Value
An operating system result code. The result code `errNeedsCompositedWindow` is returned if you try to embed into the content view in a non-compositing window; you can only embed into the content view in a compositing window.

Discussion
The new subview is added to the front of the list of subviews (that is, it is made topmost). The subview being added is not retained by the new parent view. Do not release the view after adding it, or it will cease to exist. All views in a window are released automatically when the window is destroyed.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Related Sample Code
HID Calibrator

Declared In
`HIView.h`

HIViewAdvanceFocus

Advances the keyboard focus to the next most appropriate view.

```
OSStatus HIViewAdvanceFocus (
   HIViewRef inRootForFocus,
   EventModifiers inModifiers
);
```

Parameters

inRootForFocus

The subtree to manipulate. The focus does not leave `inRootToFocus`. Typically, you would pass the content of the window or the root. If focused on the toolbar, for example, the focus is limited to the toolbar only. In this case, for example, the Toolbox passes the toolbar view in as the focus root.

inModifiers

The keyboard event modifiers that caused `HIViewAdvanceFocus` (page 2442) to be called. These modifiers are used to determine the focus direction as well as other alternate focusing behaviors.

Return Value
An operating system result code.

Discussion
Unless overridden in some fashion (either by overriding certain Carbon events or by calling the `HIViewSetNextFocus` (page 2487), the Toolbox uses a spatially determinant method of focusing, attempting to focus left to right and top to bottom in a window, taking groups of controls into account.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`HIView.h`

HIViewApplyLayout

Applies the current layout to the specified view.

```
OSStatus HIViewApplyLayout (
   HIViewRef inView
);
```

Parameters

inView

The view to which the layout is to be applied.

Return Value
An operating system result code.

Discussion
Side bindings have no effect, but positioning and scaling are applied, in that order.

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In
`HIView.h`

HIViewChangeAttributes

Changes the attributes of a view.

```
OSStatus HIViewChangeAttributes (
   HIViewRef inView,
   OptionBits inAttrsToSet,
   OptionBits inAttrsToClear
);
```

Parameters

inView

The view whose attributes you want to change.

inAttrsToSet

The attributes you want to change. For possible values, see "HIView Attributes" (page 2508).

inAttrsToClear

The attributes you want to clear. For possible values, see "HIView Attributes" (page 2508).

Return Value

An operating system result code.

Discussion

You can set and clear attributes simultaneously.

Availability

Available in Mac OS X v10.2 and later.

Not available to 64-bit applications.

Declared In

`HIView.h`

HIViewChangeFeatures

Changes the features of a view.

```
OSStatus HIViewChangeFeatures (
   HIViewRef inView,
   HIViewFeatures inFeaturesToSet,
   HIViewFeatures inFeaturesToClear
);
```

Parameters

inView

The view whose features are to be changed.

inFeaturesToSet

The features that are to be set. For details, see "HIView Feature Constants" (page 2509).

inFeaturesToClear

The features that are to be cleared. For details, see "HIView Feature Constants" (page 2509).

Return Value

An operating system result code.

Discussion
The view itself typically controls its features. For example, the view might decide that under some situations it is opaque and in others it is transparent. In general entities outside of the view itself should not call this function. The only exception might be user-interface building tools that want to make sure a view always responds to clicks, for example, so it could override mouse tracking to drag items.

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Related Sample Code
HID Calibrator

Declared In
`HIView.h`

HIViewChangeTrackingArea

Changes the shape of a tracking area.

```
OSStatus HIViewChangeTrackingArea (
   HIViewTrackingAreaRef inArea,
   HIShapeRef inShape
);
```

Parameters
inArea
The tracking area to change.
inShape
The shape to use. Pass `NULL` to use the entire structure region of the view.

Return Value
An operating system result code.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`HIView.h`

HIViewClick

Passes a mouse-down event to a view.

```
OSStatus HIViewClick (
   HIViewRef inView,
   EventRef inEvent
);
```

Parameters
inView
The view to handle the event.

inEvent
The mouse event to handle.

Return Value
An operating system result code.

Discussion
After a successful call to `HIViewGetViewForMouseEvent` for a mouse down event, you should call this function to have the view handle the click. In general we recommend using the Standard Window Handler instead of calling this function yourself.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`HIView.h`

HIViewConvertPoint

Converts a point's coordinates from one view to another.

```
OSStatus HIViewConvertPoint (
   HIPoint *ioPoint,
   HIViewRef inSourceView,
   HIViewRef inDestView
);
```

Parameters
ioPoint
The point to convert.

inSourceView
The view whose coordinate system `ioPoint` is starting out in. You can pass `NULL` to indicate that `ioPoint` is a window-relative point.

inDestView
The view whose coordinate system `ioPoint` should end up in. You can pass `NULL` to indicate that `ioPoint` is a window-relative point.

Return Value
An operating system result code.

Discussion
Both views must have a common ancestor, that is, they must both be in the same window.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`HIView.h`

HIViewConvertRect

Converts a rectangle from one view to another.

```
OSStatus HIViewConvertRect (
   HIRect *ioRect,
   HIViewRef inSourceView,
   HIViewRef inDestView
);
```

Parameters

ioRect

The rectangle to convert.

inSourceView

The view whose coordinate system `ioRect` is starting out in. You can pass `NULL` to indicate that `ioRect` is a window-relative rectangle.

inDestView

The view whose coordinate system `ioRect` should end up in. You can pass `NULL` to indicate that `ioRect` is a window-relative rectangle.

Return Value

An operating system result code.

Discussion

Both views must have a common ancestor, that is, they must both be in the same window.

Availability

Available in Mac OS X v10.2 and later.

Not available to 64-bit applications.

Declared In

`HIView.h`

HIViewConvertRegion

Converts a region from one view to another.

```
OSStatus HIViewConvertRegion (
   RgnHandle ioRgn,
   HIViewRef inSourceView,
   HIViewRef inDestView
);
```

Parameters

ioRgn

The region to convert.

inSourceView

The view whose coordinate system `ioRgn` is starting out in. You can pass `NULL` to indicate that `ioRgn` is a window-relative region.

inDestView

The view whose coordinate system `ioRgn` should end up in. You can pass `NULL` to indicate that `ioRgn` is a window-relative region.

Return Value
An operating system result code.

Discussion
Both views must have a common ancestor, that is, they must both be in the same window.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`HIView.h`

HIViewCopyShape

Copies the shape of a part of a view.

```
OSStatus HIViewCopyShape (
   HIViewRef inView,
   HIViewPartCode inPart,
   HIShapeRef *outShape
);
```

Parameters

inView
The view having a part whose shape is to be copied.

inPart
The part of the view whose shape is to be copied. For possible values, see "HIViewPartCode Constants" (page 2513).

outShape
On exit, the newly created shape. You are responsible for releasing the copied shape.

Return Value
An operating system result code.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`HIView.h`

HIViewCopyText

Copies the text of a view.

```
CFStringRef HIViewCopyText (
   HIViewRef inView
);
```

Parameters

inView
The view whose text is to be copied.

Return Value
A CFString containing a copy of the view's text. The caller is responsible for releasing the CFString.

Discussion
This function attempts to copy the text that is displayed when drawing the view and is generally successful on views that handle the `kControlEditTextCFStringTag` `GetControlData` tag. If this function can't copy that text, it copies the text in the view's title instead.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`HIView.h`

HIViewCountSubviews

Returns the number of subviews embedded in a view.

```
CFIndex HIViewCountSubviews (
   HIViewRef inView
);
```

Parameters
inView
The view whose subviews are to be counted.

Return Value
The number of subviews for the specified view.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`HIView.h`

HIViewCreateOffscreenImage

Creates a Core Graphics offscreen image of a view.

```
OSStatus HIViewCreateOffscreenImage (
   HIViewRef inView,
   OptionBits inOptions,
   HIRect *outFrame,
   CGImageRef *outImage
);
```

Parameters
inView
The view you want to create an image of.
inOptions
Options. Currently you must pass zero.

outFrame

The frame of the view within the resultant image. It is in the coordinate system of the image, where 0,0 is the top left corner of the image. This is so you can know exactly where the control is in the image when the control draws outside its bounds for things such as shadows.

outImage

The image of the view, including anything that would be drawn outside the view's frame.

Return Value
An operating system status code.

Discussion
This function creates an `CGImageRef` for the specified view. The view and any of its children are rendered in the resultant image.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`HIView.h`

HIViewDisposeTrackingArea

Disposes of an existing tracking area.

```
OSStatus HIViewDisposeTrackingArea (
   HIViewTrackingAreaRef inArea
);
```

Parameters

inArea

The tracking area that is to be disposed of.

Return Value
An operating system result code.

Discussion
All tracking areas attached to a view are automatically disposed of when the view is disposed of, so you don't need to dispose of a tracking area explicitly unless you want to remove it from a view before the view is disposed of.

After you call this function, the reference is invalid.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`HIView.h`

HIViewDrawCGImage

Draws a Core Graphics image appropriately for a view.

```
OSStatus HIViewDrawCGImage (
   CGContextRef inContext,
   const HIRect *inBounds,
   CGImageRef inImage
);
```

Parameters

inContext

The context to draw in.

inBounds

The bounds to draw the image into.

inImage

The image to draw.

Return Value
An operating system status code.

Discussion
This function is similar to `CGContextDrawImage`, but it flips the context so that the image is drawn correctly. The origin of a view is at the top left corner, so you are really drawing upside-down. This call insulates you from that fact.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`HIView.h`

HIViewFindByID

Obtains a view by its ID.

```
OSStatus HIViewFindByID (
   HIViewRef inStartView,
   HIViewID inID,
   HIViewRef *outView
);
```

Parameters

inStartView

The view to start searching at.

inID

The ID of the view you are looking for.

outControl

Receives the control if found.

Return Value
An operating system result code.

Discussion
Allows you to find a particular view by its ID. The `HIViewID` type used by this function is identical to the `ControlID` type used by the Control Manager.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Related Sample Code
CarbonCocoa_PictureCursor
HID Calibrator
HID Config Save
QTCarbonShell
QTMetaData

Declared In
`HIView.h`

HIViewFlashDirtyArea

Flashes a window's dirty area.

```
OSStatus HIViewFlashDirtyArea (
   WindowRef inWindow
);
```

Parameters
inWindow
The window whose dirty area is to be flashed.

Return Value
An operating system result code.

Discussion
As a debugging aid, this function flashes the area for an entire window that will be redrawn at the next draw time.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`HIToolboxDebugging.h`

HIViewGetAttributes

Obtains the attributes for a view.

```
OSStatus HIViewGetAttributes (
   HIViewRef inView,
   OptionBits *outAttrs
);
```

Parameters
inView
The view to inspect.

outAttrs
The attributes of the view. For possible values, see "HIView Attributes" (page 2508).

Return Value
An operating system result code.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
HIView.h

HIViewGetBounds

Obtains the local bounds of a view.

```
OSStatus HIViewGetBounds (
   HIViewRef inView,
   HIRect *outRect
);
```

Parameters
inView
The view whose local bounds are to be obtained.

outRect
The local bounds of the view.

Return Value
An operating system result code.

Discussion
The local bounds are the coordinate system that is completely view-relative. A view's top left coordinate starts out at 0, 0. Most operations use local coordinates. Note, however, that the frame is used to move a view, not local coordinates.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Related Sample Code
HID Calibrator
HID Config Save

Declared In
HIView.h

HIViewGetCommandID

Obtains the command ID of a view.

```
OSStatus HIViewGetCommandID (
   HIViewRef inView,
   UInt32 *outCommandID
);
```

Parameters

inView

The view whose command ID is to be obtained.

outID

A pointer to a value of type `UInt32` that, on return, contains the view's command ID.

Return Value
An operating system result code.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`HIView.h`

HIViewGetEventTarget

Returns the `EventTargetRef` for the specified view.

```
EventTargetRef HIViewGetEventTarget (
   HIViewRef inView
);
```

Parameters

inImage

The view for which the `EventTargetRef` should be returned.

Return Value
The `EventTargetRef`.

Discussion
Once you obtain this reference, you can install an event handler and send events to the target.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`HIView.h`

HIViewGetFeatures

Obtains the features of the specified view.

```
OSStatus HIViewGetFeatures (
   HIViewRef inView,
   HIViewFeatures *outFeatures
);
```

Parameters

inView

The view to query.

outFeatures

A pointer to a value of the `HIViewFeatures` that, on return, contains the view's features. For more information, see "HIView Feature Constants" (page 2509).

Return Value
An operating system result code.

Discussion
This function returns feature bits for the view but does not return older Control Manager features, such as `kControlSupportsDataAccess`.

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In
`HIView.h`

HIViewGetFirstSubview

Returns the first subview of a parent view.

```
HIViewRef HIViewGetFirstSubview (
   HIViewRef inView
);
```

Parameters

inView

The view whose subview you are fetching.

Return Value
An HIView reference, or `NULL` if this view has no subviews or is invalid.

Discussion
Returns the first subview of a container. The first subview is the topmost subview in z-order.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`HIView.h`

HIViewGetFocusPart

Obtains the part in the specified view that currently has focus.

```
OSStatus HIViewGetFocusPart (
   HIViewRef inView,
   HIViewPartCode *outFocusPart
);
```

Parameters

inView

The view to inquire about.

outFocusPart

The part that currently has focus. For more information, see "HIViewPartCode Constants" (page 2513).

Return Value
An operating system result code.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
HIView.h

HIViewGetFrame

Obtains the frame bounds of a view.

```
OSStatus HIViewGetFrame (
   HIViewRef inView,
   HIRect *outRect
);
```

Parameters

inView

The view whose frame you want to obtain.

outRect

The frame of the view.

Return Value
An operating system result code.

Discussion
The frame bounds is the bounds of a view relative to its parent's local coordinate system.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Related Sample Code
HID Calibrator

Declared In
HIView.h

HIViewGetID

Obtains the HIViewID of a view.

```
OSStatus HIViewGetID (
   HIViewRef inView,
   HIViewID *outID
);
```

Parameters

inView

The view whose validity is to be checked.

outID

A pointer to a value of type `HIViewID` that, on return, contains the view's `HIViewID`.

Return Value

An operating system result code.

Availability

Available in Mac OS X v10.4 and later.

Not available to 64-bit applications.

Declared In

`HIView.h`

HIViewGetIndexedSubview

Obtains the subview of a view by index.

```
OSStatus HIViewGetIndexedSubview (
   HIViewRef inView,
   CFIndex inSubviewIndex,
   HIViewRef *outSubview
);
```

Parameters

inView

The view whose indexed subview is being requested.

inSubviewIndex

The index of the requested subview.

outSubview

A pointer to an `HIViewRef` that, on return, represents the indexed subview.

Return Value

The number of subviews for the specified view.

Discussion

Instead of calling `HIViewGetIndexedSubview` repeatedly, it may be more efficient to iterate through the subviews of a view with calls to `HIViewGetFirstSubview` (page 2455) and `HIViewGetNextView` (page 2460).

Availability

Available in Mac OS X v10.4 and later.

Not available to 64-bit applications.

Declared In
HIView.h

HIViewGetKind

Obtains the signature and kind of a view.

```
OSStatus HIViewGetKind (
   HIViewRef inView,
   HIViewKind *outViewKind
);
```

Parameters

inView

The view whose signature and kind is to be obtained.

outViewKind

A pointer to a HIViewKind structure that, on return, contains the view's signature and kind. For details, see HIViewKind (page 2497).

Return Value
An operating system result code.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
HIView.h

HIViewGetLastSubview

Returns the last subview in a parent view.

```
HIViewRef HIViewGetLastSubview (
   HIViewRef inView
);
```

Parameters

inView

The view whose subview you are fetching.

Return Value
An HIView reference, or NULL if this view has no subviews or is invalid.

Discussion
Returns the last subview of a container. The last subview is the bottommost subview in z-order.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
HIView.h

HIViewGetLayoutInfo

Obtains the layout information of an view.

```
OSStatus HIViewGetLayoutInfo (
   HIViewRef inView,
   HILayoutInfo *outLayoutInfo
);
```

Parameters

inOptions

The view whose layout information is to be obtained.

outLayoutInfo

A pointer to an `HILayoutInfo` (page 2493) into which to copy the view's layout information. If the version field of this structure is not valid, the call will fail.

Return Value

An operating system result code.

Availability

Available in Mac OS X v10.3 and later.

Not available to 64-bit applications.

Related Sample Code

HID Calibrator

Declared In

`HIView.h`

HIViewGetMaximum

Obtains a view's maximum value.

```
SInt32 HIViewGetMaximum (
   HIViewRef inView
);
```

Parameters

inView

The view whose maximum value is to be obtained.

Return Value

The maximum value of the specified view.

Availability

Available in Mac OS X v10.4 and later.

Not available to 64-bit applications.

Declared In

`HIView.h`

HIViewGetMinimum

Obtains the minimum value of a view.

```
SInt32 HIViewGetMinimum (
   HIViewRef inView
);
```

Parameters

inView

The view whose minimum value is to be obtained.

Return Value
The minimum value of the specified view.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`HIView.h`

HIViewGetNeedsDisplay

Determines whether a view needs to be redrawn.

```
Boolean HIViewGetNeedsDisplay (
   HIViewRef inView
);
```

Parameters

inView

The view to inspect.

Return Value
A Boolean whose value is `true` if the view passed in or any of its subviews require redrawing; otherwise, `false`.

Discussion
A view or subview requires redrawing if any part of the view or subview has been invalidated.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`HIView.h`

HIViewGetNextView

Returns the view behind the specified view.

```
HIViewRef HIViewGetNextView (
   HIViewRef inView
);
```

Parameters

inView

The view to use as reference.

Return Value
An HIView reference, or `NULL` if this view has no view behind it or is invalid.

Discussion
Returns the view after the specified view, in z-order.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`HIView.h`

HIViewGetOptimalBounds

Obtains the optimal size and text placement of a view.

```
OSStatus HIViewGetOptimalBounds (
   HIViewRef inView,
   HIRect *outBounds,
   CGFloat *outBaseLineOffset
);
```

Parameters

inView

The view whose optimal size and text placement are to be obtained.

outBounds

A pointer to a value of type `HIRect` that, on return, contains the view's optimal bounds. Pass `NULL` if you don't need this information.

outBaseLineOffset

A pointer to a value of type `float` that, on return, contains the view's optimal text placement. Pass `NULL` if you don't need this information.

Return Value
An operating system result code.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`HIView.h`

HIViewGetPartHit

Determines the part hit for a given point.

```
OSStatus HIViewGetPartHit (
   HIViewRef inView,
   const HIPoint *inPoint,
   HIViewPartCode *outPart
);
```

Parameters

inView

The view to test the part hit.

inPoint

The view-relative point to use.

outPart

The part hit by `inPoint`.

Return Value
An operating system result code.

Discussion
Given a view, and a view-relative point, this function returns the part code hit as determined by the view.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`HIView.h`

HIViewGetPreviousView

Returns the view above the specified view.

```
HIViewRef HIViewGetPreviousView (
   HIViewRef inView
);
```

Parameters

inView

The view to use as reference.

Return Value
An HIView reference, or `NULL` if this view has no view in front of it or is invalid.

Discussion
Returns the view before the specified view, in z-order.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`HIView.h`

HIViewGetRoot

Obtains the root view for a window.

```
HIViewRef HIViewGetRoot (
   WindowRef inWindow
);
```

Parameters

inWindow
The window to get the root for.

Return Value
The root view for the window, or `NULL` if an invalid window is passed.

Discussion
Note that the root view is not the same as the Control Manager root control.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Related Sample Code
CarbonCocoa_PictureCursor
HID Calibrator
HID Config Save
QTCarbonShell
QTMetaData

Declared In
`HIView.h`

HIViewGetSizeConstraints

Returns the minimum and maximum size for a control.

```
OSStatus HIViewGetSizeConstraints (
   HIViewRef inView,
   HISize *outMinSize,
   HISize *outMaxSize
);
```

Parameters

inView
The view to inspect.

outMinSize
The minimum size the view can be.

outMaxSize
The maximum size the view can be.

Return Value
An operating system result code.

Discussion
These sizes can, for example, be used to help auto-position subviews. To get meaningful results, the control must respond to the `kEventControlGetSizeContraints` event.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`HIView.h`

HIViewGetSubviewHit

Returns the child of the given view hit by the point passed in.

```
OSStatus HIViewGetSubviewHit (
   HIViewRef inView,
   const HIPoint *inPoint,
   Boolean inDeep,
   HIViewRef *outView
);
```

Parameters

inView
The view you wish to position.

inPoint
The mouse coordinate to use. This is passed in the local coordinate system of inView.

inDeep
Pass true to find the deepest child hit, false to go only one level deep (just check direct children of inView).

outView
The view hit by inPoint, or NULL if no subview was hit.

Return Value
An operating system result code.

Discussion
This function is more primitive than using `HIViewGetViewForMouseEvent` (page 2466), and should be used only in non-event-handling cases.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`HIView.h`

HIViewGetSuperview

Returns a view's parent view.

```
HIViewRef HIViewGetSuperview (
   HIViewRef inView
);
```

Parameters

inView

The view whose parent you are interested in getting.

Return Value
An HIView reference, or `NULL` if this view has no parent or is invalid.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`HIView.h`

HIViewGetTrackingAreaID

Obtains the ID of a tracking area.

```
OSStatus HIViewGetTrackingAreaID (
   HIViewTrackingAreaRef inArea,
   HIViewTrackingAreaID *outID
);
```

Parameters

inArea

The tracking area whose ID is to be obtained.

outID

On return, the ID for the tracking area specified by `inArea`.

Return Value
An operating system result code.

Discussion
The tracking area ID that is obtained is the value that was specified when `HIViewNewTrackingArea` (page 2473) was called to create the tracking area.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`HIView.h`

HIViewGetValue

Obtains the value of a view.

```
SInt32 HIViewGetValue (
   HIViewRef inView
);
```

Parameters

inView

The view whose value is to be obtained.

Return Value
The view's value.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
HIView.h

HIViewGetViewForMouseEvent

Returns the appropriate view to handle a mouse event.

```
OSStatus HIViewGetViewForMouseEvent (
   HIViewRef inView,
   EventRef inEvent,
   HIViewRef *outView
);
```

Parameters

inView

The view to start from. You should pass the window's root view.

inEvent

The mouse event in question.

outView

The view that the mouse event should be sent to.

Return Value
An operating system result code.

Discussion
This function is a little higher-level than `HIViewGetSubviewHit` (page 2464). This function finds the deepest view that should handle the mouse event. It also sends a Carbon event to each view asking it to return the appropriate subview, which allows parent views to catch clicks on their subviews. This function is the recommended function to use before processing mouse events. Using one of the more primitive functions may result in an undefined behavior.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
HIView.h

HIViewGetViewSize

Obtains the view size of a view.

```
SInt32 HIViewGetViewSize (
   HIViewRef inView
);
```

Parameters

inView
The view whose view size is to be obtained.

Return Value

The view size.

Discussion

The view size is the size of the content to which a view's display is proportioned. The view size is commonly used to set the proportional size of a scroll bar's thumb indicator.

Availability

Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In

`HIView.h`

HIViewGetWindow

Obtains a reference to the window to which the specified view is bound.

```
WindowRef HIViewGetWindow (
   HIViewRef inView
);
```

Parameters

inView
The view to query.

Return Value

An operating system result code, or `NULL` if the view reference specified by `inView` is invalid or if the view is not bound to any window.

Availability

Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In

`HIView.h`

HIViewIsActive

Determines whether a view is active.

```
Boolean HIViewIsActive (
   HIViewRef inView,
   Boolean *outIsLatentActive
);
```

Parameters

inView

The view that is to be queried.

outIsLatentActive

A pointer to a Boolean that, on return, is set to `true` if the view is latently active or `false` if the view is not latently active. Pass `NULL` if you don't need this information.

Return Value
A Boolean whose value indicates whether the view is active (`true`) or not (`false`).

Discussion
A view's active state is affected by the active state of its parents. If `HIViewIsActive` finds that any parent view is inactive, it returns `false` to indicate that the view specified by `inView` is considered to be inactive too. In addition, `HIViewIsActive` can optionally check to see if a view is latently active, even if the view's parents are inactive.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`HIView.h`

HIViewIsCompositingEnabled

Determines whether compositing is enabled for a view.

```
Boolean HIViewIsCompositingEnabled (
   HIViewRef inView
);
```

Parameters

inView

The view that is to be queried.

Return Value
A Boolean whose value indicates whether compositing is enabled (`true`) or not (`false`).

Discussion
Checking a window's `kWindowCompositingAttribute` attribute is not sufficient for determining whether a view is in compositing or non-compositing mode because some of a window's views can be in either mode at the same time. Call this function to determine the current compositing mode of a view.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`HIView.h`

HIViewIsDrawingEnabled

Determines if drawing is currently enabled for a view.

```
Boolean HIViewIsDrawingEnabled (
   HIViewRef inView
);
```

Parameters

inView

The view to get the drawing state for.

Return Value

A Boolean value indicating that drawing is on (`true`) or off (`false`).

Availability

Available in Mac OS X v10.2 and later.

Not available to 64-bit applications.

Declared In

`HIView.h`

HIViewIsEnabled

Determines whether a view is enabled.

```
Boolean HIViewIsEnabled (
   HIViewRef inView,
   Boolean *outIsLatentEnabled
);
```

Parameters

inView

The view to query.

outIsLatentEnabled

A pointer to a Boolean that, on return, is set to `true` if the view is latently enabled or `false` if the view is not latently enabled. Pass `NULL` if you don't need this information.

Return Value

A Boolean whose value indicates whether the view is enabled (`true`) or not (`false`).

Discussion

A view's enabled state is affected by the enabled state of its parents. If `HIViewIsEnabled` finds that any parent view is disabled, it returns `false` to indicate that the view specified by `inView` is considered to be disabled too. In addition, `HIViewIsEnabled` can optionally check to see if a view is latently enabled, even if its parents are disabled.

Availability

Available in Mac OS X v10.4 and later.

Not available to 64-bit applications.

Declared In

`HIView.h`

HIViewIsLatentlyVisible

Determines whether a view is latently visible.

```
Boolean HIViewIsLatentlyVisible (
   HIViewRef inView
);
```

Parameters

inView

The view whose latent visibility is to be queried.

Return Value

A Boolean value indicating whether the view is latently visible (`true`) or hidden (`false`).

Discussion

A view's visibility is affected by the visibility of its parents. If any parent view is invisible, the view specified by `inView` is considered to be invisible too. `HIViewIsLatentlyVisible` returns whether a view is latently visible, even if its parents are invisible.

Availability

Available in Mac OS X v10.4 and later.

Not available to 64-bit applications.

Declared In

`HIView.h`

HIViewIsLayoutActive

Determines whether layout handling is active or suspended.

```
Boolean HIViewIsLayoutActive (
   HIViewRef inView
);
```

Parameters

inView

The view for which the status of layout handling is to be determined.

Return Value

A Boolean whose value is `true` if the view would respond to any linked relative's changes; otherwise `false`.

Discussion

A view's layout active state is also affected by the layout active state of its parents. If a parent view has an inactive layout, this view is also considered to have an inactive layout. To determine the latent active state of a view, see `HIViewIsLayoutLatentlyActive` (page 2471).

Note that this function does not determine whether the view's layout is valid.

Availability

Available in Mac OS X v10.3 and later.

Not available to 64-bit applications.

Declared In

`HIView.h`

HIViewIsLayoutLatentlyActive

Determines whether layout handling is latently active or suspended.

```
Boolean HIViewIsLayoutLatentlyActive (
   HIViewRef inView
);
```

Parameters

inView
The view for which the status of layout handling is to be determined.

Return Value

A Boolean whose value is `true` if the view would latently respond to any linked relative's changes; otherwise `false`.

Discussion

This function determines whether a view's layout is latently active, even if one of its parent's layouts is not active.

Availability

Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In

`HIView.h`

HIViewIsValid

Determines whether the specified view is known to the HIToolbox.

```
Boolean HIViewIsValid (
   HIViewRef inView
);
```

Parameters

inView
The view to check.

Return Value

A Boolean whose value is `true` if the view is known to the HIToolbox; otherwise, `false`.

Discussion

This function does *not* check the data in the view for validity.

Availability

Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In

`HIView.h`

HIViewIsVisible

Determines whether a view is visible.

```
Boolean HIViewIsVisible (
   HIViewRef inView
);
```

Parameters

inView

The view whose visibility you want to determine.

Return Value
A Boolean value indicating whether the view is visible (`true`) or hidden (`false`).

Discussion
`HIViewIsVisible` returns the effective visibility of a view, which is determined both by the view's own visibility and the visibility of its parent views. If a parent view is invisible, this view is considered to be invisible too.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`HIView.h`

HIViewMoveBy

Move a view by the specified distance relative to its current location.

```
OSStatus HIViewMoveBy (
   HIViewRef inView,
   CGFloat inDX,
   CGFloat inDY
);
```

Parameters

inView

The view you want to move.

inDX

The horizontal distance to move the view. Negative values move the view to the left, positive values to the right.

inDY

The vertical distance to move the view. Negative values move the view up, positive values down.

Return Value
An operating system result code.

Discussion
This function affects the view's frame but does not affect the view's bounds.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`HIView.h`

HIViewNewTrackingArea

Creates a new tracking area for a view.

```
OSStatus HIViewNewTrackingArea (
   HIViewRef inView,
   HIShapeRef inShape,
   HIViewTrackingAreaID inID,
   HIViewTrackingAreaRef *outRef
);
```

Parameters

inView

The view for which a new tracking area is to be created.

inShape

The shape to use. Pass `NULL` to use the entire structure region of the view. On return, you may safely release the shape.

inID

An identifier for the new tracking area. You can specify any value you want, or zero if you don't want to associate an identifier with the new tracking area.

outRef

On return, a reference to the new tracking area. In Mac OS X v10.5 and later, you may pass `NULL` if you don't need this information. The new tracking area is automatically destroyed when the view is released; you do not need to dispose of the tracking area yourself unless you remove it from the view before the view is released.

Return Value

An operating system result code.

Availability

Available in Mac OS X v10.4 and later.

Not available to 64-bit applications.

Related Sample Code

CarbonCocoa_PictureCursor

Declared In

`HIView.h`

HIViewPlaceInSuperviewAt

Places a view at an absolute location within its parent.

```
OSStatus HIViewPlaceInSuperviewAt (
   HIViewRef inView,
   CGFloat inX,
   CGFloat inY
);
```

Parameters

inView

The view you want to position.

inX

The absolute horizontal coordinate at which to position the view.

inY

The absolute vertical coordinate at which to position the view.

Return Value
An operating system result code.

Discussion
This function affects the view's frame but does not affect the view's bounds.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`HIView.h`

HIViewRegionChanged

Informs the system that a region of the view has changed.

```
OSStatus HIViewRegionChanged (
   HIViewRef inView,
   HIViewPartCode inRegionCode
);
```

Parameters

inView

The view whose region changed.

inView

The region that changed. At present, the region can only be the structure, opaque, and clickable regions. For possible constants, see "HIView Meta-Parts Constants" (page 2511).

Return Value
An operating system result code.

Discussion
The view system may respond to the information provided by this function in some way. For example, if a view's clickable region changes, call this function to tell the Toolbox to resynchronize the region it uses for asynchronous window dragging (if enabled). Likewise, if a view's opaque region changes, call this function to adjust the window's opaque shape.

You don't need to call this function when a view is moved or resized because the HIToolbox automatically handles those kinds of changes.

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In
`HIView.h`

HIViewRemoveFromSuperview

Removes a view from its parent.

```
OSStatus HIViewRemoveFromSuperview (
   HIViewRef inView
);
```

Parameters

inView

The view to remove.

Return Value
An operating system result code.

Discussion
The subview that is removed from the parent is not released and still exists.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Related Sample Code
QTCarbonShell

Declared In
`HIView.h`

HIViewRender

Renders the invalid portions of a view.

```
OSStatus HIViewRender (
   HIViewRef inView
);
```

Parameters

inView

The view that is to be rendered.

Return Value
An operating system result code.

Discussion
Normally, areas are redrawn at event loop time, but there might be times when an immediate redraw is needed. You should call this function sparingly because it does a fully composited redraw for the area of the view. That is, all other views that intersect the area of the specified view are also redrawn. Calling this function for several views at a particular level of a hierarchy would be costly, so you should only pass the root view of a window to this function.

The behavior of this function when passed a non-root view changed in Mac OS X v10.4. In Mac OS X v10.3, when called on a non-root view, this function validated all of the views in the window that intersect the specified view, including portions that did not intersect the specified view. Consequently, all of the views were not actually redrawn. In Mac OS X v10.4, when called on a non-root view, this function only validates those portions of each view that intersect the specified view.

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In
`HIView.h`

HIViewReshapeStructure

Informs the system that the structure region of the given view has changed shape.

```
OSStatus HIViewReshapeStructure (
   HIViewRef inView
);
```

Parameters

inView
The view to reshape and invalidate.

Return Value
An operating system result code.

Discussion
This function is used by custom views. If a view decides that its structure will change shape, it should call this function. This tells the toolbox to recalculate and invalidate as appropriate. You might, for example, call this function when gaining or losing a focus ring.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`HIView.h`

HIViewResumeLayout

Resumes layout handling for a view and its children.

```
OSStatus HIViewResumeLayout (
   HIViewRef inView
);
```

Parameters

inView
The view for which layout handling is to be resumed.

Return Value
An operating system result code.

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In
HIView.h

HIViewScrollRect

Scrolls a view's contents, or a portion thereof.

```
OSStatus HIViewScrollRect (
   HIViewRef inView,
   const HIRect *inRect,
   CGFloat inDX,
   CGFloat inDY
);
```

Parameters

inView
The view to scroll.

inRect
The rect to scroll. Pass NULL to mean the entire view. The rect passed cannot be bigger than the view's bounds. It must be in the local coordinate system of the view.

inDX
The horizontal distance to scroll. Positive values shift to the right, negative values shift to the left.

inDY
The vertical distance to scroll. Positive values shift downward, negative values shift upward.

Return Value
An operating system result code.

Discussion
This function blits the contents of the view as appropriate to scroll and then invalidates those portions that need to be redrawn. Be warned that this is a raw bit scroll. Anything that overlaps the target view will be scrolled as well.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
HIView.h

HIViewSetActivated

Sets a view to be active or inactive.

```
OSStatus HIViewSetActivated (
   HIViewRef inView,
   Boolean inSetActivated
);
```

Parameters

inView
The view that is to be set.

inSetActivated
A Boolean whose value is `true` to set the view to be active or `false` to set the view to be inactive.

Return Value
An operating system result code.

Discussion
This function affects the effective activation of a view, which affects the effective activation of the view's children. If the view is being set to inactive, all children become inactive as well, but their latent activation does not change. If the children are latently inactive and the view is made active, the children remain latently inactive.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`HIView.h`

HIViewSetBoundsOrigin

Sets the origin of the view.

```
OSStatus HIViewSetBoundsOrigin (
   HIViewRef inView,
   CGFloat inX,
   CGFloat inY
);
```

Parameters
inView
The view whose origin you wish to adjust.
inX
The X coordinate.
inY
The Y coordinate.

Return Value
An operating system result code.

Discussion
This effectively also moves all subcontrols of a view as well. This call will not invalidate the view, in case you might want to move the contents with `HIViewScrollRect` instead of redrawing the complete content.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Related Sample Code
HID Calibrator

Declared In
`HIView.h`

HIViewSetCommandID

Sets the command ID of a view.

```
OSStatus HIViewSetCommandID (
   HIViewRef inView,
   UInt32 inCommandID
);
```

Parameters

inView

The view whose command ID is to be set.

inID

The command ID to set.

Return Value

An operating system result code.

Availability

Available in Mac OS X v10.4 and later.

Not available to 64-bit applications.

Declared In

`HIView.h`

HIViewSetDrawingEnabled

Turns control drawing on or off.

```
OSStatus HIViewSetDrawingEnabled (
   HIViewRef inView,
   Boolean inEnabled
);
```

Parameters

inView

The view to enable or disable drawing for.

inEnabled

A Boolean value indicating whether drawing should be on (`true`) or off (`false`).

Return Value

An operating system result code.

Discussion

You can use this function to ensure that no drawing events are sent to the specified control. Functions such as `Draw1Control` and `HIViewSetNeedsDisplay` cannot draw when control drawing is off.

Availability

Available in Mac OS X v10.2 and later.

Not available to 64-bit applications.

Declared In

`HIView.h`

HIViewSetEnabled

Enables or disables a view.

```
OSStatus HIViewSetEnabled (
   HIViewRef inView,
   Boolean inSetEnabled
);
```

Parameters

inView

The view that is to be set.

inSetEnabled

A Boolean whose value is `true` to enable the view or `false` to disable the view.

Return Value

An operating system result code.

Discussion

Any subviews of the view specified by `inView` become enabled or disabled in accordance with the value of `inSetEnabled`.

Availability

Available in Mac OS X v10.4 and later.

Not available to 64-bit applications.

Declared In

`HIView.h`

HIViewSetFirstSubViewFocus

Sets the subview that is first to receive keyboard focus.

```
OSStatus HIViewSetFirstSubViewFocus (
   HIViewRef inParent,
   HIViewRef inSubView
);
```

Parameters

inParent

The parent view.

inSubView

The first subview that is to receive keyboard focus. Pass `NULL` to tell the view system to use the default rules.

Return Value

An operating system result code.

Discussion

This function sets the first subview to shift focus to whenever the keyboard focus is advanced and the container view is entered.

Availability

Available in Mac OS X v10.2 and later.

Not available to 64-bit applications.

Declared In
HIView.h

HIViewSetFrame

Sets the frame of a view.

```
OSStatus HIViewSetFrame (
   HIViewRef inView,
   const HIRect *inRect
);
```

Parameters

inView
The view whose frame is to be set.

inRect
The new frame to set.

Return Value
An operating system result code.

Discussion
This function effectively moves the view within its parent. It also marks the view (and anything that was exposed behind it) to be redrawn.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Related Sample Code
HID Calibrator

Declared In
HIView.h

HIViewSetHilite

Sets highlighting on a view.

```
OSStatus HIViewSetHilite (
   HIViewRef inView,
   HIViewPartCode inHilitePart
);
```

Parameters

inView
The view for which highlighting is to be set.

inHilitePart
The part of the view whose highlighting is to be set. For possible values, see "HIViewPartCode Constants" (page 2513).

Return Value
An operating system result code.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`HIView.h`

HIViewSetID

Sets the `HIViewID` of a view.

```
OSStatus HIViewSetID (
   HIViewRef inView,
   HIViewID inID
);
```

Parameters

inView
The view whose `HIViewID` is to be set.

inID
The `HIViewID` to set.

Return Value
An operating system result code.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`HIView.h`

HIViewSetLayoutInfo

Sets the layout information of an HIView.

```
OSStatus HIViewSetLayoutInfo (
   HIViewRef inView,
   const HILayoutInfo *inLayoutInfo
);
```

Parameters

inView
The view whose layout information is to be set.

inLayoutInfo
A pointer to an `HILayoutInfo` (page 2493) structure containing the layout values that are to be set.

Return Value
An operating system result code.

Discussion
Layouts are used to automatically change the size and positioning of a view when another view changes size or position. Layout changes only take effect in two cases:

- When `HIViewApplyLayout` (page 2443) is called after the view's layout has been set by calling `HIViewSetLayoutInfo`. The `HIViewApplyLayout` function is most useful when first applying a scaling or positioning layout to a view in order to set up the view's initial position and size relative to the view referenced in the layout.
- When the related view changes its size or position.

A layout allows a view's size or position to be modified in three ways: side binding, axial scaling, and axial positioning. In the following examples, View A initially has a left side of 100 and a right side of 150, and is placed inside a parent view that has a width of 275.

- Bindings — Setting up a binding relationship between two views specifies that one edge of a view is to change by an amount equal to the change in an edge of another view. If View A has a right-side binding to its parent's maximum (or right) side, when the parent's right side changes, View A's right side changes by the same amount. If the parent view resizes to be 325 units wide (50 units wider than before), View A is resized so that its left side changes to 150 units and its right side changes to 200, which is 50 units more to the right than before. You can think of bindings as a way to maintain distance. An edge bound to another view's edge always maintains its offset from that related edge. In other view systems, this concept is often referred to as "springs and rods." Note that a binding does *not* cause one side of the view to exactly align with the side of another view; a binding merely causes one side of the view to change by the same amount as another view changes. To align one edge of a view to another view's edge, use positioning. Bindings depend on changes in size or position of the related view. As a result, calling `HIViewApplyLayout` does not activate side bindings, as no changes have occurred. Bindings are implemented using the `HIBinding` (page 2493) structure and one `HISideBinding` (page 2494) structure per view edge.
- Scaling — Setting up a scaling relationship between two views specifies that the axial size (that is, the width or height) of a view is to be a specified ratio of the size of another view when that other view moves or resizes. If View A has an x-axis scaling for its parent view with a ratio of 0.8, when the parent view's width changes, View A's width changes to be the parent's width multiplied by 0.8. If the parent view resizes to be 325 units wide, View A resizes so that its left side stays at 100 and its right side changes to 360 (100 + 325 * 0.8). Note that when a scaling layout is first set up on a view with `HIViewSetLayoutInfo`, no scaling is applied to the view because scaling only occurs when the related view resizes. If scaling is required at initial setup, call `HIViewSetLayoutInfo` and then `HIViewApplyLayout` (page 2443). Scaling is implemented using the `HIScaling` (page 2494) structure and one `HIAxisScale` (page 2494) structure per view axis.
- Positioning — Setting up a positioning relationship between two views specifies that the axial position (i.e., vertical or horizontal) of a view is to change so that the view aligns with the minimum, maximum, or center of another view when that other view resizes. If View A has an x-axis position with its parent view with center positioning specified, when the parent view changes size, View A moves so that it is centered horizontally relative to its parent. If the parent view resizes to be 300 units wide, View A repositions so that its left side is at 125 and its right side is at 175, centered in the parent view. Positioning is implemented using the `HIPositioning` (page 2495) structure and one `HIAxisPosition` (page 2495) per view axis.

The HIView layout engine applies transformations to a view sequentially. First, bindings are applied. Then scaling is applied, which could override some of the previously applied bindings. Then positioning is applied, which could also override some of the previously applied bindings. The bindings are applied recursively to a container's subviews, which requires care on your part to avoid infinite recursion, especially when applying inter-relational bindings. For example, if View A's x axis is scaled relative to View B and View B's x-axis is scaled to View A, your application could hang when the layouts are applied because View A would affect View B, which would affect View A, and so on.

For more information on using the layout engine, see *HIView Programming Guide*.

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Related Sample Code
HID Calibrator

Declared In
`HIView.h`

HIViewSetMaximum

Sets a view's maximum value.

```
OSStatus HIViewSetMaximum (
   HIViewRef inView,
   SInt32 inMaximum
);
```

Parameters

inView
The view whose maximum value is to be set.

inView
The maximum value to set.

Return Value
An operating system result code.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`HIView.h`

HIViewSetMinimum

Sets a view's minimum value.

```
OSStatus HIViewSetMinimum (
   HIViewRef inView,
   SInt32 inMinimum
);
```

Parameters

inView
The view whose minimum value is to be set.

inMinimum
The value to set as the view's minimum value.

Return Value
An operating system result code.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`HIView.h`

HIViewSetNeedsDisplay

Marks a view as needing or not needing to be redrawn.

```
OSStatus HIViewSetNeedsDisplay (
   HIViewRef inView,
   Boolean inNeedsDisplay
);
```

Parameters

inView
The view to mark as dirty (needing to be redrawn) or clean (not needing to be redrawn).

inNeedsDisplay
A Boolean whose value is `true` to mark the view as dirty or `false` to mark it as clean.

Return Value
An operating system result code.

Discussion
If the view is not visible or is obscured completely by other views, no action is taken.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Related Sample Code
HID Calibrator
HID Config Save
HID Explorer

Declared In
`HIView.h`

HIViewSetNeedsDisplayInRect

Uses an `HIRect` to mark a portion of a view as needing or not needing to be redrawn.

```
OSStatus HIViewSetNeedsDisplayInRect (
   HIViewRef inView,
   const HIRect *inRect,
   Boolean inNeedsDisplay
);
```

Parameters

inView

The view having a region that is to be marked as dirty (needs to be redrawn) or clean (valid and not needing to be redrawn).

inRect

The area, in view-relative coordinates, that is to be marked.

inNeedsDisplay

A Boolean whose value is `true` to mark the area described by `inRect` as dirty or `false` to mark it as clean.

Return Value
An operating system result code.

Discussion
If the view is not visible or is obscured completely by other views, no action is taken. The area specified by `inRect` is intersected with the view's visible region.

Availability
Available in Mac OS X v10.4 and later.

Not available to 64-bit applications.

Declared In
`HIView.h`

HIViewSetNeedsDisplayInRegion

Uses a region to mark a portion of a view as needing or not needing to be redrawn.

```
OSStatus HIViewSetNeedsDisplayInRegion (
   HIViewRef inView,
   RgnHandle inRgn,
   Boolean inNeedsDisplay
);
```

Parameters

inView

The view having a region that is to be marked as dirty (needs to be redrawn) or clean (valid and not needing to be redrawn).

inRgn

The region, in view-relative coordinates, to mark as dirty or clean.

inNeedsDisplay

A Boolean whose value is `true` to mark the region described by `inRgn` as dirty or `false` to mark it as clean.

Return Value
An operating system result code.

Discussion
If the view is not visible or is obscured completely by other views, no action is taken. The specified region is effectively intersected with the view's visible region.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`HIView.h`

HIViewSetNeedsDisplayInShape

Uses a shape to mark a portion of a view as needing or not needing to be redrawn.

```
OSStatus HIViewSetNeedsDisplayInShape (
   HIViewRef inView,
   HIShapeRef inArea,
   Boolean inNeedsDisplay
);
```

Parameters

`inView`
The view having a shape that is to be marked as dirty (needs to be redrawn) or clean (valid and not needing to be redrawn).

`inArea`
The area, in view-relative coordinates, that is to be marked.

`inNeedsDisplay`
A Boolean whose value is `true` to mark the area described by `inArea` as dirty or `false` to mark it as clean.

Return Value
An operating system result code.

Discussion
If the view is not visible or is obscured completely by other views, no action is taken. The area specified by `inArea` is intersected with the view's visible region.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`HIView.h`

HIViewSetNextFocus

Sets the view that is to receive keyboard focus when keyboard focus advances from the specified view.

```
OSStatus HIViewSetNextFocus (
   HIViewRef inView,
   HIViewRef inNextFocus
);
```

Parameters

inView

The view.

inNextFocus

The view that is to receive keyboard focus when focus is advanced from the view specified by `inView`. The view must have the same parent as the view specified by `inView`. Pass `NULL` to tell the view system to use the default rules.

Return Value
An operating system result code.

Discussion
This function sets the view to which keyboard focus is to be shifted the next time keyboard focus is advanced from the view specified by `inView`.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`HIView.h`

HIViewSetText

Sets the text of a view to the specified string.

```
OSStatus HIViewSetText (
   HIViewRef inView,
   CFStringRef inText
);
```

Parameters

inView

The view whose text is to be set.

inText

The text that is to be set.

Return Value
An operating system result code.

Discussion
This function attempts to set the text that is displayed when drawing the view and is generally successful on views that handle the `kControlEditTextCFStringTag` `SetControlData` tag. If this function can't set that text, it sets the text in the view's title instead.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Related Sample Code
HID Explorer

Declared In
`HIView.h`

HIViewSetValue

Sets the value of a view.

```
OSStatus HIViewSetValue (
   HIViewRef inView,
   SInt32 inValue
);
```

Parameters

inView
The view whose value is to be set.

inValue
The value to set.

Return Value
An operating system result code.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`HIView.h`

HIViewSetViewSize

Sets the view size of a view.

```
OSStatus HIViewSetViewSize (
   HIViewRef inView,
   SInt32 inViewSize
);
```

Parameters

inView
The view whose view size is to be set.

inViewSize
The view size that is to be set.

Return Value
An operating system result code.

Discussion
The view size is the size of the content to which a view's display is proportioned. The view size is commonly used to set the proportional size of a scroll bar's thumb indicator.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`HIView.h`

HIViewSetVisible

Hides or shows a view.

```
OSStatus HIViewSetVisible (
   HIViewRef inView,
   Boolean inVisible
);
```

Parameters

inView
The view to hide or show.

inVisible
A Boolean value that indicates whether you want to hide the view (`false`) or show the view (`true`).

Return Value
An operating system result code.

Discussion
Marks the area the view will occupy or previously occupied as needing to be redrawn later.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Related Sample Code
CarbonCocoa_PictureCursor
HID Calibrator

Declared In
`HIView.h`

HIViewSetZOrder

Changes the front-to-back ordering of sibling views.

```
OSStatus HIViewSetZOrder (
   HIViewRef inView,
   HIViewZOrderOp inOp,
   HIViewRef inOther
);
```

Parameters

inView
The view whose Z-order you want to change.

inOp

Causes `inView` to be ordered above or below `inOther`. For possible values, see "HIView Z-Ordering Constants" (page 2511).

inOther

Another optional view to use as a reference. Pass `NULL` to indicate an absolute position.

Return Value
An operating system result code.

Discussion
For example, passing `kHIViewZOrderAbove` as the value on `inOp` and `NULL` as the value of `inOther` moves a view to the front of all of its siblings. Passing `kHIViewZOrderBelow` as the value of `inOp` and `NULL` as the value of `inOther` moves a view to the back of all its siblings.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`HIView.h`

HIViewSimulateClick

Simulates a mouse click on a given view.

```
OSStatus HIViewSimulateClick (
   HIViewRef inView,
   HIViewPartCode inPartToClick,
   UInt32 inModifiers,
   HIViewPartCode *outPartClicked
);
```

Parameters

inView

The view to test the part hit.

inPartToClick

The part the view should consider to be clicked.

inModifiers

The modifiers the view can consider for its click action.

outPartClicked

The part that was hit; can be `kHIViewNoPart` if no action occurred. For possible values, see "HIViewPartCode Constants" (page 2513). Pass `NULL` if you don't need the part code returned.

Return Value
An operating system result code.

Discussion
This function is used to simulate a mouse click on a given view. It sends a `kEventControlSimulateHit` event to the specified view and also sends `kEventControlHit` and (if the Hit event is not handled) `kEventCommandProcess` events.

Note that not all windows respond to the events that this API sends. A fully Carbon-event-based window most likely responds exactly as if the user had really clicked in the view. A window that uses Classic event record-based APIs (`WaitNextEvent` or `ModalDialog`) typically does not respond at all. To simulate a click in such a window, you may need to post a mouse-down/mouse-up pair or use a Dialog Manager event filter proc to simulate a hit in a dialog item.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`HIView.h`

HIViewSubtreeContainsFocus

Determines whether a view or any subviews have keyboard focus.

```
Boolean HIViewSubtreeContainsFocus (
   HIViewRef inSubtreeStart
);
```

Parameters

inSubtreeStart
The view to query.

Return Value
A Boolean whose value is `true` if the view or any of its children have keyboard focus; otherwise, `false`.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`HIView.h`

HIViewSuspendLayout

Suspends layout handling for a view and its children.

```
OSStatus HIViewSuspendLayout (
   HIViewRef inView
);
```

Parameters

inView
The view for which layout handling is to be suspended.

Return Value
An operating system result code.

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In
`HIView.h`

Data Types

HILayoutInfo

Structure that stores the layout of an HIView.

```
struct HILayoutInfo {
    UInt32 version;
    HIBinding binding;
    HIScaling scale;
    HIPositioning position;
};
```

Fields

`version`
The version of this structure. The current version is `kHILayoutInfoVersionZero`.

`binding`
An `HIBinding` structure describing the bindings to apply to the sides of an HIView.

`scale`
An `HIScaling` structure describing the axial scaling to apply to an HIView.

`position`
An `HIPositioning` structure describing the positioning to apply to an HIView.

Discussion
This structure is provided as a parameter to `HIViewGetLayoutInfo` (page 2459) and `HIViewSetLayoutInfo` (page 2482).

HIBinding

Represents a set of top, left, bottom, and right bindings for an view.

```
struct HIBinding {
    HISideBinding top;
    HISideBinding left;
    HISideBinding bottom;
    HISideBinding right;
};
```

Fields

`top`
The top side bindings.

`left`
The left side bindings.

`bottom`
The bottom side bindings.

`right`
The right side bindings.

Discussion
These constants are used in conjunction with the HIView layout engine.

HISideBinding

Structure for storing the binding for the side of a view.

```
struct HISideBinding {
    HIViewRef toView;
    HIBindingKind kind;
    float offset;
};
```

Fields
`toView`
An `HIViewRef` to the view to which this side is bound. This field can be `NULL`, which indicates that the side is bound to its parent view.

`kind`
The bind kind. For possible values, see "HILayout Binding Kind Constants" (page 2505).

`offset`
Reserved; must be set to zero.

Discussion
The layout engine can automatically reposition and resize views for which relationships have been set up. (Call `HIViewSetLayoutInfo` (page 2482) to establish these relationships.) A side binding is entirely related to the change of the parent's position or size but only as the size affects the maximum edge position. A side binding doesn't mean "move to where my relative's side is" but rather "move as my relative's side has moved."

HIScaling

A set of scaling descriptions for the axes of a view.

```
struct HIScaling {
    HIAxisScale x;
    HIAxisScale y;
};
```

Fields
`x`
The horizontal scaling for a view.

`x`
The vertical scaling for a view.

HIAxisScale

Represents a scale description for an axis of a view.

```
struct HIAxisScale {
    HIViewRef toView;
    HIScaleKind kind;
    float ratio;
};
```

Fields

`toView`

An `HIViewRef` to the view to which this axis is scaled. This field can be `NULL`, which indicates that the axis is scaled relative to its parent's view.

`kind`

The type of scaling. Currently, this field must be `kHILayoutScaleAbsolute`.

`ratio`

A value that indicates how much to scale the view. Zero indicates no scaling. A value of one indicates that the view is to always have the same axial size.

HIPositioning

A positioning description for an HIView.

```
struct HIPositioning {
    HIAxisPosition x;
    HIAxisPosition y;
};
```

Fields

`x`

The X axis.

`y`

The Y axis.

HIAxisPosition

An axial position description for an HIView.

```
struct HIAxisPosition {
    HIViewRef toView;
    HIPositionKind kind;
    float offset;
};
```

Fields

`toView`

An `HIViewRef` to the view relative to which a view positioned. This field can be `NULL`, which indicates that the axis is positioned relative to its parent's view.

`kind`

The type of positioning. For possible values, see "HIPositionKind Constants" (page 2506).

`offset`

After the position kind has been applied, the origin component that corresponds to the positioning axis is offset by this value. For example, left aligned + 10.

HIViewContentInfo

Describes the content of a view.

```
struct HIViewContentInfo {
    HIViewContentType contentType;
    union {
        IconSuiteRef iconSuite;
        IconRef iconRef;
        CGImageRef imageRef;
    } u;
};
typedef struct HIViewContentInfo HIViewContentInfo;
typedef HIViewContentInfo * HIViewContentInfoPtr;
```

Fields

`contentType`
The type of content in the union. For possible values, see "HIViewContentType Constants" (page 2512).

Availability

Available in Mac OS X v10.4 and later.

Declared In

`HIView.h`

HIViewID

Defines the HIView ID.

```
typedef ControlID HIViewID;
```

Availability

Available in Mac OS X v10.2 and later.

Declared In

`HIView.h`

HIViewFrameMetrics

Describes the offsets from the structure to the content area of a view.

```
struct HIViewFrameMetrics {
    float top;
    float left;
    float bottom;
    float right;
};
```

Fields

`top`
Height of the top of the structure area.

`left`
Width of the left of the structure area.

`bottom`
Height of the bottom of the structure area.

`right`
Width of the right of the structure area.

Discussion
The top metric is the difference between the vertical coordinate of the top edge of the view's structure region and the vertical coordinate of the top edge of the view's content region. This structure is returned by a view in response to a `kEventControlGetFrameMetrics` event.

HIViewKind

Represents the signature and kind of the view.

```
struct HIViewKind {
    OSType signature;
    OSType kind;
}
```

Fields
`signature`
The signature of the view. Apple reserves all signatures made up of only lowercase characters.

`kind`
The kind of the view. Apple reserves all kinds made up of only lowercase characters.

HIViewRef

Define an HIView reference.

```
typedef ControlRef HIViewRef;
```

Availability
Available in Mac OS X v10.2 and later.

Declared In
`HIObject.h`

HIViewTrackingAreaRef

Define an HIView tracking area reference.

```
typedef struct OpaqueHIViewTrackingAreaRef HIViewTrackingAreaRef;
```

Availability
Available in Mac OS X v10.4 and later.

Declared In
`HIView.h`

HIViewTrackingAreaID

Define an HIView tracking area ID.

```
typedef UInt64 HIViewTrackingAreaID;
```

Availability
Available in Mac OS X v10.4 and later.

Declared In
`HIView.h`

Constants

Class ID Constants

Specify class identifiers for view-related subclasses of HIObject.

```
#define kHIViewClassID              CFSTR("com.apple.hiview")
#define kHIGrowBoxViewClassID       CFSTR("com.apple.higrowboxview")
#define kHIImageViewClassID         CFSTR("com.apple.HIImageView")
#define kHIMenuViewClassID          CFSTR("com.apple.HIMenuView")
#define kHIStandardMenuViewClassID  CFSTR("com.apple.HIStandardMenuView")
#define kHISegmentedViewClassID     CFSTR("com.apple.HISegmentedView")
#define kHIScrollViewClassID        CFSTR("com.apple.HIScrollView")
#define kHIComboBoxClassID          CFSTR("com.apple.HIComboBox")
#define kHISearchFieldClassID       CFSTR("com.apple.HISearchField")
#define kHICocoaViewClassID         CFSTR("com.apple.HICocoaView")
```

Constants

`kHIViewClassID`
Class identifier for the HIView class.
Available in Mac OS X v10.2 and later.
Declared in `HIView.h`.

`kHIGrowBoxViewClassID`
Class identifier for the HIGrowBoxView class.
Available in Mac OS X v10.2 and later.
Declared in `HIWindowViews.h`.

`kHIImageViewClassID`
Class identifier for the HIImageView class.
Available in Mac OS X v10.3 and later.
Declared in `HIImageViews.h`.

`kHIMenuViewClassID`
Class identifier for the HIMenuView class.
Available in Mac OS X v10.3 and later.
Declared in `HIMenuView.h`.

`kHIStandardMenuViewClassID`
Class identifier for the HIStandardMenuView class.
Available in Mac OS X v10.3 and later.
Declared in `HIMenuView.h`.

`kHISegmentedViewClassID`
Class identifier for the HISegmentedView class.
Available in Mac OS X v10.3 and later.
Declared in `HISegmentedView.h`.

`kHIScrollViewClassID`
Class identifier for the HIScrollView class.
Available in Mac OS X v10.3 and later.
Declared in `HIScrollView.h`.

`kHIComboBoxClassID`
Class identifier for the HIComboBox class.
Available in Mac OS X v10.3 and later.
Declared in `HIComboBox.h`.

`kHISearchFieldClassID`
Class identifier for the HISearchField class.
Available in Mac OS X v10.3 and later.
Declared in `HISearchField.h`.

`kHICocoaViewClassID`
Class identifier for the HICocoaView class.
Available in Mac OS X v10.5 and later.
Declared in `HICocoaView.h`.

Clock Event Constant

Specify the constant for changes in date or time in the clock control (`kEventClassClockView`).

```
enum {
    kEventClockDateOrTimeChanged = 1
};
```

Constants

`kEventClockDateOrTimeChanged`
An event sent by the clock control when the user changes the date or time. Clients can register for this notification in order to update state based on the date or time in the clock. This event is sent to the view only; it is not propagated. The event is sent to all handlers installed on the control.
Available in Mac OS X v10.4 and later.
Declared in `HIClockView.h`.

Discussion

Table 34-1 shows the event parameters associated with this event.

Table 34-1 Parameter names and types for date or time change events

Event kind	Parameter name	Parameter type

kEventClockDateOrTimeChanged	kEventParamDirectObject	typeControlRef

Combo Box Attributes

Specify constants for combo box attributes.

```
enum {
    kHIComboBoxNoAttributes = 0L,
    kHIComboBoxAutoCompletionAttribute = (1L << 0),
    kHIComboBoxAutoDisclosureAttribute = (1L << 1),
    kHIComboBoxAutoSortAttribute = (1L << 2),
    kHIComboBoxAutoSizeListAttribute = (1L << 3),
    kHIComboBoxStandardAttributes = (kHIComboBoxAutoCompletionAttribute |
kHIComboBoxAutoDisclosureAttribute | kHIComboBoxAutoSizeListAttribute)
};
```

Constants

`kHIComboBoxNoAttributes`
Indicates the lack of any attributes.

Available in Mac OS X v10.2 and later.

Declared in `HIComboBox.h`.

`kHIComboBoxAutoCompletionAttribute`
The control will attempt to auto complete the text the user is typing with an item in the combo box list that is the closest appropriate match.

Available in Mac OS X v10.2 and later.

Declared in `HIComboBox.h`.

`kHIComboBoxAutoDisclosureAttribute`
The control will disclose the combo box list after the user enters text.

Available in Mac OS X v10.2 and later.

Declared in `HIComboBox.h`.

`kHIComboBoxAutoSortAttribute`
The items in the combo box list will be automatically sorted in alphabetical order.

Available in Mac OS X v10.2 and later.

Declared in `HIComboBox.h`.

`kHIComboBoxAutoSizeListAttribute`
The combo box list will be automatically sized to fit the Human Interface Guidelines.

Available in Mac OS X v10.2 and later.

Declared in `HIComboBox.h`.

`kHIComboBoxStandardAttributes`
The minimum set of commonly used combo box attributes.

Available in Mac OS X v10.2 and later.

Declared in `HIComboBox.h`.

Combo Box Data Tags

Specify constants for combo box data tags.

```
enum {
    kHIComboBoxListTag = 'cbls',
    kHIComboBoxListPixelWidthTag = 'cblw',
    kHIComboBoxListPixelHeightTag = 'cblh',
    kHIComboBoxNumVisibleItemsTag = 'cbni'
};
```

Constants

`kHIComboBoxListTag`

Extract the contents of the combo box list as a `CFArray`. The `CFArray` is retained; if you get the array, you own it and must release it. If you set it by calling `SetControlData`, the toolbox makes a copy of it, and you are free to release your reference. The reference count is bumped on get/retains on set.

Available in Mac OS X v10.2 and later.

Declared in `HIComboBox.h`.

`kHIComboBoxListPixelWidthTag`

A `UInt32` containing the width of the combo box list. The width can be customized. This tag disables the auto size attribute.

Available in Mac OS X v10.2 and later.

Declared in `HIComboBox.h`.

`kHIComboBoxListPixelHeightTag`

A `UInt32` containing the height of the combo box list. The height can be customized. This tag disables the auto size attribute.

Available in Mac OS X v10.2 and later.

Declared in `HIComboBox.h`.

`kHIComboBoxNumVisibleItemsTag`

A `UInt32` containing the number of visible items in the combo box list. The number can be customized. This tag disables the auto size attribute.

Available in Mac OS X v10.2 and later.

Declared in `HIComboBox.h`.

Discussion

The combo box view also supports these tags previously defined for the `EditUnicodeText` control. These tags are available through `GetControlData` and `SetControlData` with a `ControlPartCode` of `kHIComboBoxEditTextPart`:

- `kControlFontStyleTag`
- `kControlEditTextFixedTextTag`
- `kControlEditTextTextTag`
- `kControlEditTextKeyFilterTag`
- `kControlEditTextValidationProcTag`
- `kControlEditUnicodeTextPostUpdateProcTag`
- `kControlEditTextSelectionTag`
- `kControlEditTextKeyScriptBehaviorTag`

- kControlEditTextCharCount
- kControlEditTextCFStringTag

Combo Box List Item Event Constants

Specify a constant for combo box list item events (kEventClassHIComboBox).

```
enum {
    kEventComboBoxListItemSelected = 1,
    kEventParamComboBoxListSelectedItemIndex = 'cbli'
};
```

Constants

kEventComboBoxListItemSelected

This event is sent as a notification when an item in the combo box disclosure list has been selected. This event is sent to all handlers installed on the control. This does not imply that the selection has been accepted; for that you will need to register for the kEventClassTextField/kEventTextAccepted event.

Available in Mac OS X v10.4 and later.

Declared in HIComboBox.h.

kEventParamComboBoxListSelectedItemIndex

Indicates the index of a selected combo box list item.

Available in Mac OS X v10.4 and later.

Declared in HIComboBox.h.

Discussion

Table 34-2 shows the event parameters associated with these events.

Table 34-2 Parameter names and types for combo box events

Event kind	Parameter name	Parameter type
kEventComboBoxList-ItemSelected	kEventParamDirectObject	typeControlRef
	kEventParamComboBoxListItemSelected-ItemIndex	typeCFIndex

Combo Box Part Constants

Specify constants for combo box part codes.

```
enum {
    kHIComboBoxEditTextPart = 5,
    kHIComboBoxDisclosurePart = 28
};
```

Constants

`kHIComboBoxEditTextPart`
Edit text part.
Available in Mac OS X v10.2 and later.
Declared in `HIComboBox.h`.

`kHIComboBoxDisclosurePart`
Disclosure part.
Available in Mac OS X v10.2 and later.
Declared in `HIComboBox.h`.

Discussion

Combo box part code constants are used when calling `HIViewSetHilite` (page 2481), `HIViewCopyShape` (page 2448) and are returned by `HIViewRegionChanged` (page 2474), `HIViewGetFocusPart` (page 2455). They are also used by the hit testing functions, `HIViewSimulateClick` (page 2491) and `HIViewGetPartHit` (page 2462).

Control Kind Constants

Specify constants for control kinds.

```
enum {
    kControlKindHIScrollView = 'scrl',
    kControlKindHIImageView = 'imag',
    kControlKindHIComboBox = 'cbbx',
    kControlKindHISearchField = 'srfd',
    kControlKindHIMenuView = 'menu',
    kControlKindHIStandardMenuView = 'smnu',
    kHISegmentedViewKind = 'sgmt',
    kControlKindHIGrowBoxView = 'grow',
    kControlKindHICocoaView = 'hins'
};
```

Constants

`kControlKindHIScrollView`
Control kind for a scroll view.
Available in Mac OS X v10.4 and later.
Declared in `HIScrollView.h`.

`kControlKindHIImageView`
Control kind for an image view.
Available in Mac OS X v10.4 and later.
Declared in `HIImageViews.h`.

`kControlKindHIComboBox`
Control kind for a combo box.
Available in Mac OS X v10.2 and later.
Declared in `HIComboBox.h`.

`kControlKindHISearchField`
Control kind for a search field.
Available in Mac OS X v10.3 and later.
Declared in `HISearchField.h`.

`kControlKindHIMenuView`
Control kind for a menu view.
Available in Mac OS X v10.4 and later.
Declared in `HIMenuView.h`.

`kControlKindHIStandardMenuView`
Control kind for a standard menu view.
Available in Mac OS X v10.4 and later.
Declared in `HIMenuView.h`.

`kHISegmentedViewKind`
Control kind for a segmented view.
Available in Mac OS X v10.3 and later.
Declared in `HISegmentedView.h`.

`kControlKindHIGrowBoxView`
Control kind for a grow box view.
Available in Mac OS X v10.4 and later.
Declared in `HIWindowViews.h`.

`kControlKindHICocoaView`
Control kind for a view that wraps a Cocoa view.
Available in Mac OS X v10.5 and later.
Declared in `HICocoaView.h`.

Discussion
These constants are returned by `HIViewGetKind` (page 2458) and `GetControlKind` (page 598).

Event Class Constants

Specify event class constants.

```
enum {
    kEventClassClockView = 'cloc',
    kEventClassScrollable = 'scrl',
    kEventClassHIComboBox = 'hicb',
    kEventClassSearchField = 'srfd',
    kEventClassTextField = 'txfd'
};
```

Constants
`kEventClassClockView`
Event class for events related to a clock view.
Available in Mac OS X v10.4 and later.
Declared in `HIClockView.h`.

`kEventClassScrollable`
Event class for events related to a scrollable view.
Available in Mac OS X v10.2 and later.
Declared in `HIScrollView.h`.

`kEventClassHIComboBox`
Event class for events related to a combo box view.
Available in Mac OS X v10.4 and later.
Declared in `HIComboBox.h`.

`kEventClassSearchField`
Event calls for events related to a search field view.
Available in Mac OS X v10.3 and later.
Declared in `HISearchField.h`.

`kEventClassTextField`
Event class for events related to text field views.
Available in Mac OS X v10.3 and later.
Declared in `HITextViews.h`.

HILayout Binding Kind Constants

Specify binding constants used by the HIView layout engine.

```
enum {
    kHILayoutBindNone = 0,
    kHILayoutBindMin = 1,
    kHILayoutBindMax= 2,
    kHILayoutBindLeft = kHILayoutBindMin,
    kHILayoutBindRight = kHILayoutBindMax,
    kHILayoutBindTop = kHILayoutBindMin,
    kHILayoutBindBottom = kHILayoutBindMax
};
typedef UInt16 HIBindingKind;
```

Constants

`kHILayoutBindNone`
No binding is to occur.
Available in Mac OS X v10.3 and later.
Declared in `HIView.h`.

`kHILayoutBindMin`
Bind to the minimum of the axis.
Available in Mac OS X v10.3 and later.
Declared in `HIView.h`.

`kHILayoutBindMax`
Bind to the maximum of the axis.
Available in Mac OS X v10.3 and later.
Declared in `HIView.h`.

`kHILayoutBindLeft`
Synonym for `kHILayoutBindMin`.

Available in Mac OS X v10.3 and later.

Declared in `HIView.h`.

`kHILayoutBindRight`
Synonym for `kHILayoutBindMax`.

Available in Mac OS X v10.3 and later.

Declared in `HIView.h`.

`kHILayoutBindTop`
Synonym for `kHILayoutBindMin`.

Available in Mac OS X v10.3 and later.

Declared in `HIView.h`.

`kHILayoutBindBottom`
Synonym for `kHILayoutBindMax`.

Available in Mac OS X v10.3 and later.

Declared in `HIView.h`.

Discussion
Mac OS X v10.3 provides a layout engine for HIViews that allows applications to specify the layout relationships between multiple views. When necessary, the layout engine automatically repositions and resizes views that have layout information. For more information on how to use the layout engine, see `HIViewSetLayoutInfo` (page 2482).

Horizontal and vertical bindings are very similar in application, but along different axes, so the binding kinds have been abstracted to minimum and maximum. Synonyms are provided for convenience, and you are encouraged to use them.

HILayoutInfoVersion Constant

Specify version `0` of the `HILayoutInfo` structure

```
enum {
    kHILayoutInfoVersionZero = 0
};
```

Constants
`kHILayoutInfoVersionZero`
The version of the `HILayoutInfo` structure is `0`.

Available in Mac OS X v10.3 and later.

Declared in `HIView.h`.

HIPositionKind Constants

Specify position constants used by the HIView layout engine.

```
enum {
    kHILayoutPositionNone = 0,
    kHILayoutPositionCenter = 1,
    kHILayoutPositionMin = 2,
    kHILayoutPositionMax = 3,
    kHILayoutPositionLeft = kHILayoutPositionMin,
    kHILayoutPositionRight = kHILayoutPositionMax,
    kHILayoutPositionTop = kHILayoutPositionMin,
    kHILayoutPositionBottom = kHILayoutPositionMax
};
typedef UInt16 HIPositionKind;
```

Constants

`kHILayoutPositionNone`
No positioning is to occur.
Available in Mac OS X v10.3 and later.
Declared in `HIView.h`.

`kHILayoutPositionCenter`
Bind to the center.
Available in Mac OS X v10.3 and later.
Declared in `HIView.h`.

`kHILayoutPositionMin`
Bind to the minimum of the axis.
Available in Mac OS X v10.3 and later.
Declared in `HIView.h`.

`kHILayoutPositionMax`
Bind to the maximum of the axis.
Available in Mac OS X v10.3 and later.
Declared in `HIView.h`.

`kHILayoutPositionLeft`
Synonym for `kHILayoutPositionMin`.
Available in Mac OS X v10.3 and later.
Declared in `HIView.h`.

`kHILayoutPositionRight`
Synonym for `kHILayoutPositionMax`.
Available in Mac OS X v10.3 and later.
Declared in `HIView.h`.

`kHILayoutPositionTop`
Synonym for `kHILayoutPositiondMin`.
Available in Mac OS X v10.3 and later.
Declared in `HIView.h`.

`kHILayoutPositionBottom`
Synonym for `kHILayoutPositionMax`.
Available in Mac OS X v10.3 and later.
Declared in `HIView.h`.

HIScaleKind Constant

Specify a constant that indicates the scale is determined from the axis size.

```
enum {
    kHILayoutScaleAbsolute = 0
};
typedef UInt16 HIScaleKind;
```

Constants

`kHILayoutScaleAbsolute`

Indicates that the scale is determined from the axis size.

Available in Mac OS X v10.3 and later.

Declared in `HIView.h`.

Discussion

This constant is used in conjunction with the HIView layout engine.

HIView Attributes

Specify constants that change the behavior of controls.

```
enum {
    kHIViewAttributeSendCommandToUserFocus = 1 << 0,
    kHIViewAttributeIsFieldEditor = 1 << 1,
    kHIViewSendCommandToUserFocus = kHIViewAttributeSendCommandToUserFocus
};
```

Constants

`kHIViewAttributeSendCommandToUserFocus`

When set, the control sends the command it generates to the user focus; the command propagates as it would naturally from there. (The default is to send the command to itself and then to its parent and so forth.) You may want to use this setting for views that are not typically in a focused window. For example, a push button in a toolbar window might use this setting to cause its command to be sent to the focused window rather than to the toolbar window.

Available in Mac OS X v10.4 and later.

Declared in `HIView.h`.

`kHIViewAttributeIsFieldEditor`

Indicates that text editing controls should behave appropriately for editing fields in a dialog; specifically, the control should ignore the Return, Enter, Escape, and Tab keys, and allow them to be processed by other participants in the event flow.

Available in Mac OS X v10.4 and later.

Declared in `HIView.h`.

`kHIViewSendCommandToUserFocus`

Legacy constant. Use `kHIViewAttributeSendCommandToUserFocus` instead.

Available in Mac OS X v10.2 and later.

Declared in `HIView.h`.

Discussion

These constants are used by `HIViewGetAttributes` (page 2452) to get a view's attributes and by `HIViewChangeAttributes` (page 2444) to set or change a view's attributes.

HIView Feature Constants

Specify constants for view features.

```
enum {
    kHIViewFeatureSupportsGhosting = 1 << 0,
    kHIViewFeatureAllowsSubviews = 1 << 1,
    kHIViewFeatureGetsFocusOnClick = 1 << 8,
    kHIViewFeatureSupportsLiveFeedback = 1 << 10,
    kHIViewFeatureSupportsRadioBehavior = 1 << 11,
    kHIViewFeatureAutoToggles = 1 << 14,
    kHIViewFeatureIdlesWithTimer = 1 << 23,
    kHIViewFeatureInvertsUpDownValueMeaning = 1 << 24,
    kHIViewFeatureIsOpaque = 1 << 25,
    kHIViewFeatureDoesNotDraw = 1 << 27,
    kHIViewFeatureDoesNotUseSpecialParts = 1 << 28,
    kHIViewFeatureIgnoresClicks = 1 << 29
};
typedef UInt64 HIViewFeatures;
```

Constants

`kHIViewFeatureSupportsGhosting`
This view supports using the ghosting protocol when live tracking is not enabled. Use this constant instead of the legacy constant, `kHIViewSupportsGhosting`.

Available in Mac OS X v10.4 and later.

Declared in `HIView.h`.

`kHIViewFeatureAllowsSubviews`
This view allows subviews to be embedded within it. Use this constant instead of the legacy constant, `kHIViewAllowsSubviews`.

Available in Mac OS X v10.4 and later.

Declared in `HIView.h`.

`kHIViewFeatureGetsFocusOnClick`
If this view is clicked, the keyboard focus should be set to this view automatically; used primarily for edit text controls. Use this constant instead of the legacy constant, `kHIViewGetsFocusOnClick`.

Available in Mac OS X v10.4 and later.

Declared in `HIView.h`.

`kHIViewFeatureSupportsLiveFeedback`
This view supports the live feedback protocol, which is necessary for implementing live scroll bar tracking. Clients of a view should never disable this bit. Use this constant instead of the legacy constant, `kHIViewSupportsLiveFeedback`.

Available in Mac OS X v10.4 and later.

Declared in `HIView.h`.

`kHIViewFeatureSupportsRadioBehavior`
This view can be placed in a radio group. Radio buttons and bevel buttons report this behavior. Use this constant instead of the legacy constant, `kHIViewSupportsRadioBehavior`.

Available in Mac OS X v10.4 and later.

Declared in `HIView.h`.

`kHIViewFeatureAutoToggles`
: This view supports the auto-toggle protocol and should at the very least auto-toggle between off and on. The view can support a Carbon event for more advanced auto-toggling of its value. The tab view supports this, for example, so that when a tab is clicked its value changes automatically. Use this constant instead of the legacy constant, `kHIViewAutoToggles`.

  Available in Mac OS X v10.4 and later.

  Declared in `HIView.h`.

`kHIViewFeatureIdlesWithTimer`
: An informational bit. Turning this bit off would not necessarily disable any timer a view might be using, but a timer could obey this bit if desired. Use this constant instead of the legacy constant, `kHIViewIdlesWithTimer`.

  Available in Mac OS X v10.4 and later.

  Declared in `HIView.h`.

`kHIViewFeatureInvertsUpDownValueMeaning`
: This bit tells the Control Manager that when the Up button part of the control is clicked, the value of the control should increase. A Scroll bar, conversely, decreases in value when its Up button is clicked. Use this constant instead of the legacy constant, `kHIViewInvertsUpDownValueMeaning`.

  Available in Mac OS X v10.4 and later.

  Declared in `HIView.h`.

`kHIViewFeatureIsOpaque`
: When set, the view's structure region is used to determine the view's opaque region, and calling the view can usually be avoided. Use this constant instead of the legacy constant, `kHIViewIsOpaque`.

  Available in Mac OS X v10.4 and later.

  Declared in `HIView.h`.

`kHIViewFeatureDoesNotDraw`
: This bit is an optimization that tells the Control Manager that a view is transparent and does not do any drawing, so the Control Manager doesn't have to invalidate the view and instead should invalidate views behind this view. For example, on a metal window, the content view is actually fully transparent, so invalidating it is unnecessary. Use this constant instead of the legacy constant, `kHIViewDoesNotDraw`.

  Available in Mac OS X v10.4 and later.

  Declared in `HIView.h`.

`kHIViewFeatureDoesNotUseSpecialParts`
: Indicates to the Control Manager that this view doesn't use the special part codes for indicator, inactive, and disabled. Use this constant instead of the legacy constant, `kHIViewDoesNotUseSpecialParts`.

  Available in Mac OS X v10.4 and later.

  Declared in `HIView.h`.

`kHIViewFeatureIgnoresClicks`
: When set, this bit tells the Control Manager that it does not need to ask the control for it's clickable region. A view such as the visual separator would set this bit, and metal windows set this bit when doing asynchronous window dragging. This bit is typically set in conjunction with the `kHIViewFeatureDoesNotDrawBit`. Use this constant instead of the legacy constant, `kHIViewIgnoresClicks`.

  Available in Mac OS X v10.4 and later.

  Declared in `HIView.h`.

Discussion
View feature flags are generally determined by the view itself and are not typically changed by clients of the view. Call `HIViewGetFeatures` (page 2454) to obtain a view's features, and `HIViewChangeFeatures` (page 2444) to set and clear a view's features.

HIView Meta-Parts Constants

Specify meta-parts constants used when calling `HIViewCopyShape`.

```
enum {
    kHIViewStructureMetaPart = -1,
    kHIViewContentMetaPart = -2,
    kHIViewOpaqueMetaPart = -3,
    kHIViewClickableMetaPart = -4
};
```

Constants

`kHIViewStructureMetaPart`
The structure region is the total area over which the view draws.
Available in Mac OS X v10.4 and later.
Declared in `HIView.h`.

`kHIViewContentMetaPart`
The content region is only defined by views that can embed other views. It is the area that embedded content can live.
Available in Mac OS X v10.4 and later.
Declared in `HIView.h`.

`kHIViewOpaqueMetaPart`
The portion of the view that is opaque. No views behind this portion of the view will be asked to draw because their drawing would be completely overwritten by this view's drawing.
Available in Mac OS X v10.4 and later.
Declared in `HIView.h`.

`kHIViewClickableMetaPart`
Used for asynchronous window dragging only. The default is the structure region.
Available in Mac OS X v10.4 and later.
Declared in `HIView.h`.

Discussion
Meta-parts are used when calling `HIViewCopyShape` (page 2448). Meta-parts define a region of a view and they might be defined by a view. Along with these parts, you can also pass in normal part codes to get the regions of those parts. Not all views fully support this feature.

HIView Z-Ordering Constants

Specify constants that set a view's z-order.

```
enum {
    kHIViewZOrderAbove = 1,
    kHIViewZOrderBelow = 2,
};
typedef UInt32 HIViewZOrderOp;
```

Constants

`kHIViewZOrderAbove`
Order the view above another view.
Available in Mac OS X v10.2 and later.
Declared in `HIView.h`.

`kHIViewZOrderBelow`
Order the view below another view.
Available in Mac OS X v10.2 and later.
Declared in `HIView.h`.

Discussion
These constants are used when calling `HIViewSetZOrder` (page 2490).

HIViewContentType Constants

Specify constants that describe a view's content.

```
enum {
    kHIViewContentTextOnly = 0,
    kHIViewContentNone = 0,
    kHIViewContentIconSuiteRef = 129,
    kHIViewContentIconRef = 132,
    kHIViewContentCGImageRef = 134
};
```

Constants

`kHIViewContentTextOnly`
The view has no content other than text.
Available in Mac OS X v10.4 and later.
Declared in `HIView.h`.

`kHIViewContentNone`
The view has no content.
Available in Mac OS X v10.4 and later.
Declared in `HIView.h`.

`kHIViewContentIconSuiteRef`
The view's content is an `IconSuiteRef`.
Available in Mac OS X v10.4 and later.
Declared in `HIView.h`.

`kHIViewContentIconRef`
The view's content is an `IconRef`.
Available in Mac OS X v10.4 and later.
Declared in `HIView.h`.

`kHIViewContentCGImageRef`
The view's content is an `CGImageRef`.

Available in Mac OS X v10.4 and later.

Declared in `HIView.h`.

Discussion
These constants are used in the `HIViewContentInfo` structure. For information on that structure, see HIViewContentInfo (page 2496).

HIViewPartCode Constants

Specify view parts constants.

```
enum {
    kHIViewNoPart = 0,
    kHIViewIndicatorPart = 129,
    kHIViewDisabledPart = 254,
    kHIViewInactivePart = 255,
    kHIViewEntireView = kHIViewNoPart
};
typedef ControlPartCode HIViewPartCode;
```

Constants
`kHIViewNoPart`
The entire view; used when not referring to a specific part.

Available in Mac OS X v10.4 and later.

Declared in `HIView.h`.

`kHIViewIndicatorPart`
Indicator part.

Available in Mac OS X v10.4 and later.

Declared in `HIView.h`.

`kHIViewDisabledPart`
Disabled part.

Available in Mac OS X v10.4 and later.

Declared in `HIView.h`.

`kHIViewInactivePart`
Inactive part.

Available in Mac OS X v10.4 and later.

Declared in `HIView.h`.

`kHIViewEntireView`
The entire view; used when not referring to a specific part.

Available in Mac OS X v10.4 and later.

Declared in `HIView.h`.

Discussion
These constants are used when calling `HIViewSetHilite` (page 2481), `HIViewRegionChanged` (page 2474), `HIViewCopyShape` (page 2448), `HIViewSimulateClick` (page 2491), `HIViewGetPartHit` (page 2462), and `HIViewGetFocusPart` (page 2455). These constants are also used with various `kEventClassControl` Carbon events.

Mouse Tracking Area Event Constants

Specify constants for mouse tracking area events (`kEventClassControl`).

```
enum {
    kEventControlTrackingAreaEntered = 23,
    kEventControlTrackingAreaExited = 24,
    kEventParamHIViewTrackingArea = 'ctra',
    typeHIViewTrackingAreaRef = 'ctra'
};
```

Constants

`kEventControlTrackingAreaEntered`

If you installed a mouse tracking area in your view, you will receive this event when the mouse enters that area. The tracking area reference is sent with the event.

Available in Mac OS X v10.4 and later.

Declared in `HIView.h`.

`kEventControlTrackingAreaExited`

If you installed a mouse tracking area in your view, you will receive this event when the mouse leaves that area. The tracking area reference is sent with the event.

Available in Mac OS X v10.4 and later.

Declared in `HIView.h`.

`kEventParamHIViewTrackingArea`

An `HIViewTrackingAreaRef` for the tracking area that was entered.

Available in Mac OS X v10.4 and later.

Declared in `HIView.h`.

Discussion

Table 34-3 shows the event parameters associated with these events.

Table 34-3 Parameter names and types for mouse tracking area events

Event kind	Parameter name	Parameter type
`kEventControl-TrackingAreaEntered`	`kEventParamHIViewTrackingArea`	`typeHIViewTrackingAreaRef`
	`kEventParamKeyModifiers`	`typeUInt32`
	`kEventParamMouseLocation`	`typeHIPoint`
`kEventControl-TrackingAreaExited`	`kEventParamHIViewTrackingArea`	`typeHIViewTrackingAreaRef`
	`kEventParamKeyModifiers`	`typeUInt32`
	`kEventParamMouseLocation`	`typeHIPoint`

Scroll View Constants

Specify constants for scroll view options.

```
enum {
    kHIScrollViewOptionsVertScroll = (1 << 0),
    kHIScrollViewOptionsHorizScroll = (1 << 1),
    kHIScrollViewOptionsAllowGrow = (1 << 2),
    kHIScrollViewValidOptions = (kHIScrollViewOptionsVertScroll |
kHIScrollViewOptionsHorizScroll | kHIScrollViewOptionsAllowGrow)
};
```

Constants

`kHIScrollViewOptionsVertScroll`
Indicates that a vertical scroll bar is desired.

Available in Mac OS X v10.2 and later.

Declared in `HIScrollView.h`.

`kHIScrollViewOptionsHorizScroll`
Indicates that a horizontal scroll bar is desired.

Available in Mac OS X v10.2 and later.

Declared in `HIScrollView.h`.

`kHIScrollViewOptionsAllowGrow`
Indicates that space for a grow box should be taken into account when laying out scroll bars. In Mac OS X v10.3 and earlier, if both horizontal and vertical scroll bars are requested, this attribute is assumed. In Mac OS X v10.4 and later, this attribute is *not* assumed; this allows the scroll view to support independent auto-hiding of the two scroll bars in Mac OS X v10.4 and later. If you want to preserve space for the grow box on all systems, specify this option bit.

Available in Mac OS X v10.2 and later.

Declared in `HIScrollView.h`.

`kHIScrollViewValidOptions`
All valid scroll view options.

Available in Mac OS X v10.2 and later.

Declared in `HIScrollView.h`.

Discussion

These constants are used in conjunction with `HIScrollViewCreate` (page 2423).

Scroll View Action Constants

Specify constants for scroll view navigation actions.

```
enum {
    kHIScrollViewScrollToTop = (1 << 0),
    kHIScrollViewScrollToBottom = (1 << 1),
    kHIScrollViewScrollToLeft = (1 << 2),
    kHIScrollViewScrollToRight = (1 << 3),
    kHIScrollViewPageUp = (1 << 4),
    kHIScrollViewPageDown = (1 << 5),
    kHIScrollViewPageLeft = (1 << 6),
    kHIScrollViewPageRight = (1 << 7)
};
typedef UInt32 HIScrollViewAction;
```

Constants

`kHIScrollViewScrollToTop`
The scroll view should move to the top of the content.
Available in Mac OS X v10.3 and later.
Declared in `HIScrollView.h`.

`kHIScrollViewScrollToBottom`
The scroll view should move to the bottom of the content.
Available in Mac OS X v10.3 and later.
Declared in `HIScrollView.h`.

`kHIScrollViewScrollToLeft`
The scroll view should move to the left of the content.
Available in Mac OS X v10.3 and later.
Declared in `HIScrollView.h`.

`kHIScrollViewScrollToRight`
The scroll view should move to the right of the content.
Available in Mac OS X v10.3 and later.
Declared in `HIScrollView.h`.

`kHIScrollViewPageUp`
The scroll view should page up.
Available in Mac OS X v10.3 and later.
Declared in `HIScrollView.h`.

`kHIScrollViewPageDown`
The scroll view should page down.
Available in Mac OS X v10.3 and later.
Declared in `HIScrollView.h`.

`kHIScrollViewPageLeft`
The scroll view should page left.
Available in Mac OS X v10.3 and later.
Declared in `HIScrollView.h`.

`kHIScrollViewPageRight`
The scroll view should page right.
Available in Mac OS X v10.3 and later.
Declared in `HIScrollView.h`.

Discussion
These constants are used in conjunction with `HIScrollViewNavigate` (page 2424) and `HIScrollViewCanNavigate` (page 2423).

Scrollable Event Constants

Specify constants for scrollable events (`kEventClassScrollable`).

```
enum {
    kEventScrollableGetInfo = 1,
    kEventScrollableInfoChanged = 2,
    kEventScrollableScrollTo = 10
};
```

Constants

`kEventScrollableGetInfo`
This event is sent by an HIScrollView to its scrollable view to determine the current size and origin of the scrollable view. A scrollable view must implement this event in order to scroll properly inside an HIScrollView.

Available in Mac OS X v10.2 and later.

Declared in `HIScrollView.h`.

`kEventScrollableInfoChanged`
This event is not sent by an HIScrollView itself. Instead, it may be sent to an instance of HIScrollView to notify the scroll view that the size or origin of its scrollable view has changed. The HIScrollView responds to this event by sending a `kEventScrollableGetInfo` event to its scrollable view. It then updates the scroll bars appropriately to reflect the changes. It does *not* move the origin of the scrollable view at all. This event is just a notification to allow the scroll view to sync up with its scrollable view.

Available in Mac OS X v10.2 and later.

Declared in `HIScrollView.h`.

`kEventScrollableScrollTo`
This event is sent by an HIScrollView to its scrollable view to request that the scrollable view update its current origin and redraw. Typically, a scrollable view scrolls its content by setting its bounds origin using `HIViewSetBoundsOrigin` (page 2478) or by offsetting its drawing by the scroll origin. If the view embeds other views, it must use `HIViewSetBoundsOrigin` to allow the embedded views to scroll along with their containing view. A view that uses `HIViewSetBoundsOrigin` should call that API in response to this event. A view that offsets its drawing by the scroll origin should update its current origin in its own instance data in response to this event. A scrollable view should also use `HIViewScrollRect` (page 2477) to scroll its content or `HIViewSetNeedsDisplay` (page 2485) to cause itself to redraw using the new origin point. A scrollable view must implement this event in order to scroll properly inside an HIScrollView.

Available in Mac OS X v10.2 and later.

Declared in `HIScrollView.h`.

Discussion
Table 34-4 shows the event parameters associated with these events.

Table 34-4 Parameter names and types for scrollable events

Event kind	Parameter name	Parameter type
`kEventScrollableGetInfo`	`kEventParamImageSize`	`typeHISize`

	kEventParamViewSize	typeHISize
	kEventParamLineSize	typeHISize
	kEventParamOrigin	typeHIPoint
kEventScrollableInfoChanged	kEventParamOrigin	typeHIPoint
kEventScrollableScrollTo	kEventParamOrigin	typeHIPoint

Scrollable Event Parameter Constants

Specify scrollable event parameter constants.

```
enum {
    kEventParamImageSize = 'imsz',
    kEventParamViewSize = 'vwsz',
    kEventParamLineSize = 'lnsz',
    kEventParamOrigin = 'orgn'
};
```

Constants

`kEventParamImageSize`

A value of type `typeHISize` representing the image size. The image size is the total size of the scrollable view, including any parts of the view that are not currently visible. For example, a scrollable view that displays a hundred page document would return an image size equal to one hundred times the height of the page.

Available in Mac OS X v10.2 and later.

Declared in `HIScrollView.h`.

`kEventParamViewSize`

A value of type `typeHISize` representing the view size. The view size is the current size of the scrollable view. Typically, this is the same as the view's bounds and can be acquired by calling `HIViewGetBounds` (page 2453).

Available in Mac OS X v10.2 and later.

Declared in `HIScrollView.h`.

`kEventParamLineSize`

A value of type `typeHISize` representing the line size. The line size is the distance that the HIScrollView should scroll its subview when the user clicks a scroll bar arrow. For example, this might be 10 pixels vertically and 20 pixels horizontally.

Available in Mac OS X v10.2 and later.

Declared in `HIScrollView.h`.

`kEventParamOrigin`
: A value of type `typeHIPoint` representing the origin. The origin is the current view-relative origin within the total scrollable image that is displayed at the top left corner of the view. These coordinates should always be greater than or equal to zero and less than or equal to the view's image size minus its view size. Typically, a view that implements the `kEventScrollableScrollTo` event by calling `HIViewSetBoundsOrigin` (page 2478) returns the current bounds origin for this parameter, and a view implements the `ScrollTo` event by storing the origin in its instance data returns its stored origin for this parameter. For example, a scrollable view that currently is displaying page ten of a hundred page document would return a horizontal origin of zero and a vertical origin equal to ten times the height of the page.

Available in Mac OS X v10.2 and later.

Declared in `HIScrollView.h`.

Search Field Attribute Constants

Specify constants for search field attributes.

```
enum {
    kHISearchFieldNoAttributes = 0,
    kHISearchFieldAttributesCancel = (1 << 0),
    kHISearchFieldAttributesSearchIcon = (1 << 1)
};
```

Constants

`kHISearchFieldNoAttributes`
: Indicates that this view does not have any attributes.

Available in Mac OS X v10.3 and later.

Declared in `HISearchField.h`.

`kHISearchFieldAttributesCancel`
: Indicates that this view contains a Cancel button.

Available in Mac OS X v10.3 and later.

Declared in `HISearchField.h`.

`kHISearchFieldAttributesSearchIcon`
: Indicates that this view contains the Search icon in the text field. If a menu is associated with the search field, this attribute is implicitly set and the Search icon will display with a menu disclosure badge.

Available in Mac OS X v10.4 and later.

Declared in `HISearchField.h`.

Discussion

These constants are used when calling `HISearchFieldCreate` (page 2427), `HISearchFieldGetAttributes` (page 2428), and `HISearchFieldChangeAttributes` (page 2425).

Search Field Data Tags

Specify constants for search field data tags.

Discussion
Search field views support these tags previously defined for the `EditUnicodeText` control. These tags are available through `GetControlData` and `SetControlData` with a `ControlPartCode` of `kControlEditTextPart`:

- `kControlFontStyleTag`
- `kControlEditTextFixedTextTag`
- `kControlEditTextTextTag`
- `kControlEditTextKeyFilterTag`
- `kControlEditTextValidationProcTag`
- `kControlEditUnicodeTextPostUpdateProcTag`
- `kControlEditTextSelectionTag`
- `kControlEditTextKeyScriptBehaviorTag`
- `kControlEditTextCharCount`
- `kControlEditTextCFStringTag`

Availability
Available in Mac OS X v10.3 and later.

Search Field Part Code Constants

Specify constants for search field part codes.

```
enum {
    kControlSearchFieldCancelPart = 30,
    kControlSearchFieldMenuPart = 31
};
```

Constants

`kControlSearchFieldCancelPart`
Cancel part.
Available in Mac OS X v10.3 and later.
Declared in `HISearchField.h`.

`kControlSearchFieldMenuPart`
Menu part.
Available in Mac OS X v10.3 and later.
Declared in `HISearchField.h`.

Segment Attribute Constants

Specify segment attribute constants.

```
enum {
    kHISegmentNoAttributes = 0,
    kHISegmentSendCmdToUserFocus = (1 << 0)
};
```

Constants

`kHISegmentNoAttributes`

Indicates no attributes.

Available in Mac OS X v10.3 and later.

Declared in `HISegmentedView.h`.

`kHISegmentSendCmdToUserFocus`

If this attribute bit is set, the command that is sent when the segment is clicked will be directed at the user focus instead of up the segmented view's containment hierarchy.

Available in Mac OS X v10.3 and later.

Declared in `HISegmentedView.h`.

Discussion

These constants are used when calling `HISegmentedViewCreate` (page 2432) and `HISegmentedViewChangeSegmentAttributes` (page 2430).

Segment Behavior Constants

Specify segment behavior constants.

```
enum {
    kHISegmentBehaviorMomentary = 1,
    kHISegmentBehaviorRadio = 2,
    kHISegmentBehaviorToggles = 3,
    kHISegmentBehaviorSticky = 4
};
```

Constants

`kHISegmentBehaviorMomentary`

Pops back up after being pressed, just like a push button.

Available in Mac OS X v10.3 and later.

Declared in `HISegmentedView.h`.

`kHISegmentBehaviorRadio`

Stays pressed until another segment with the radio behavior is pressed. This makes the segment behave like a radio button. After this segment is clicked, the segmented view's value is changed to this segment's one-based index.

Available in Mac OS X v10.3 and later.

Declared in `HISegmentedView.h`.

`kHISegmentBehaviorToggles`

Behaves like a check box. When clicked, it toggles back and forth between checked and unchecked states. Currently, this constant should not be used; if you use it, you get the same effect as if you used `kHISegmentBehaviorMomentary`.

Available in Mac OS X v10.3 and later.

Declared in `HISegmentedView.h`.

`kHISegmentBehaviorSticky`

After being pressed, this type of segment stays pressed until it is unpressed programmatically. Currently, this constant should not be used; if you use it, you get the same effect as if you used `kHISegmentBehaviorMomentary`.

Available in Mac OS X v10.3 and later.

Declared in `HISegmentedView.h`.

Discussion
These constants are used in conjunction with `HISegmentedViewSetSegmentBehavior` (page 2437) and `HISegmentedViewGetSegmentBehavior` (page 2433).

Standard View Constants

Specify IDs of views that are commonly used.

```
const HIViewID kHIViewWindowContentID;
const HIViewID kHIViewWindowGrowBoxID;
const HIViewID kHIViewMenuContentID;
```

Constants
`kHIViewWindowContentID`

The standard view ID for the content view of a window.

Available in Mac OS X v10.2 and later.

Declared in `HIWindowViews.h`.

`kHIViewWindowGrowBoxID`

The standard view ID for the grow box view of a window. Not all windows have grow boxes, so you might not find this view if you look for it.

Available in Mac OS X v10.2 and later.

Declared in `HIWindowViews.h`.

`kHIViewMenuContentID`

The standard view ID for the content view of a menu. The Menu Manager assigns this view ID to all menu content views.

Available in Mac OS X v10.3 and later.

Declared in `HIMenuView.h`.

Text Field Event Constants

Specify constants for text field events (`kEventClassTextField`).

```
enum {
    kEventTextAccepted = 1,
    kEventTextShouldChangeInRange = 2,
    kEventTextDidChange = 3
};
```

Constants

`kEventTextAccepted`

This event is sent as a notification when the text contained in a control's editable text field has been accepted by the user. Text is accepted when the user presses Return or Enter on the keyboard for the `EditUnicodeText`, `HIComboBox`, and `HISearchField` controls, or when the text has changed in the field and the field loses focus for the `EditUnicodeText`, `HIComboBox`, `HISearchField` and `HITextView` controls. This event is sent to the control containing the text field only, it is not propagated. It is sent to all handlers installed on the control containing the text field.

Available in Mac OS X v10.3 and later.

Declared in `HITextViews.h`.

`kEventTextShouldChangeInRange`

This event is sent whenever the text is about to be modified in text field, either by user input or in other scenarios such as a paste from the clipboard, spell-checking word correction, or Mac OS X Service operation. You can change the text that is inserted by providing a replacement string as a parameter to this event. This event is only sent for Unicode text controls; it is not sent for the Classic non-Unicode `EditText` control. This event is not sent prior to programmatic modification of the text field contents using `SetControlData`. This event is not sent while an active inline editing session is in progress. Once the inline text has been confirmed, this event is sent prior to the confirmed text being inserted into the text field. If you need control over keystrokes during an inline editing session, you can use the `kEventTextInputFilterText` event. This event is sent to the control containing the text field only; it does not propagate.

Available in Mac OS X v10.4 and later.

Declared in `HITextViews.h`.

`kEventTextDidChange`

This event is sent to indicate that the contents of an editable text field have changed. This event is sent by all of the Unicode-based editable text views: `HIComboBox`, `HISearchField`, `HITextView` and `EditUnicodeText`. This event is not sent for the Classic non-Unicode `EditText` control. Note that this event is sent after inline editing operations, such as pressing a dead key, or using a input method that creates an inline editing hole. Most clients of this event should ignore the event during inline editing, and only respond to changes to the text after inline editing completes. A client can check for the presence of the `kEventParamUnconfirmedRange` parameter to determine whether inline editing is currently active; if this parameter is present, the client may wish to ignore the event. This event is not sent after programmatic modification of the text field contents using `SetControlData`. This event is sent only to the control containing the text field; it does not propagate. It is sent to all handlers registered for it.

Available in Mac OS X v10.4 and later.

Declared in `HITextViews.h`.

Discussion

Table 34-5 shows the event parameters associated with these events.

Table 34-5 Parameter names and types for text field events

Event kind	Parameter name	Parameter type
`kEventTextAccepted`	`kEventParamDirectObject`	`typeControlRef`

kEventTextShouldChangeInRange	kEventParamTextSelection	typeCFRange
	kEventParamCandidateText	typeCFStringRef
	kEventParamReplacementText (Optional)	typeCFStringRef
kEventTextDidChange	kEventParamUnconfirmedRange	typeCFRange
	kEventParamUnconfirmedText	typeCFStringRef

Text Field Event Parameter Constants

Specify constants for text field event parameters.

```
enum {
    kEventParamTextSelection = 'txsl',
    kEventParamCandidateText = 'tstx',
    kEventParamReplacementText = 'trtx',
    kEventParamUnconfirmedRange = 'tunr',
    kEventParamUnconfirmedText = 'txun'
};
```

Constants

`kEventParamTextSelection`

The range of the selection that is about to be changed. The units of the selection are in the same units that are returned in a `EditTextSelectionRec`, when called with `GetControlData` using `kControlEditTextSelectionTag`.

Available in Mac OS X v10.4 and later.

Declared in `HITextViews.h`.

`kEventParamCandidateText`

The text that is to replace the selection. Note that this string was originally created with `CFStringCreateWithCharactersNoCopy`, and the original text has a limited life span. If for some reason you need to retain the text past the end of your event handler, you should extract the characters from the string with `CFStringGetCharacters`, and then store those characters or create a new `CFString` from them.

Available in Mac OS X v10.4 and later.

Declared in `HITextViews.h`.

`kEventParamReplacementText`

Optional replacement text.

Available in Mac OS X v10.4 and later.

Declared in `HITextViews.h`.

`kEventParamUnconfirmedRange`

If the text field currently has an open inline hole, this parameter contains the range of text inside the hole. This parameter is optional and is only present during inline editing.

Available in Mac OS X v10.4 and later.

Declared in `HITextViews.h`.

`kEventParamUnconfirmedText`
If the text field currently has an open inline hole, this parameter contains the non-confirmed text currently being edited inside the hole. This parameter is optional and is only present during inline editing. Note that this string was originally created with `CFStringCreateWithCharactersNoCopy`, and the original text has a limited life span. If for some reason you need to retain the text past the end of your event handler, you should extract the characters from the string with `CFStringGetCharacters`, and then store those characters or create a new `CFString` from them.

Available in Mac OS X v10.4 and later.

Declared in `HITextViews.h`.

Transformation Constants

Specify transformation constants.

```
enum {
    kHITransformNone = 0x00,
    kHITransformDisabled = 0x01,
    kHITransformSelected = 0x4000
};
```

Constants

`kHITransformNone`
No visual transformation should be applied.

Available in Mac OS X v10.4 and later.

Declared in `HIView.h`.

`kHITransformDisabled`
The image should be transformed to use a disabled appearance. This transformation should not be combined with any other transformations.

Available in Mac OS X v10.4 and later.

Declared in `HIView.h`.

`kHITransformSelected`
The image should be transformed to use a selected appearance. This transformation should not be combined with any other transformations.

Available in Mac OS X v10.4 and later.

Declared in `HIView.h`.

Discussion
These constants are used when calling `HICreateTransformedCGImage` (page 2415).

kHIViewKindSignatureApple

Specify the signature of all HIToolbox views.

```
enum {
    kHIViewKindSignatureApple = 'appl'
};
```

Constants

`kHIViewKindSignatureApple`

Signature of all toolbox views.

Available in Mac OS X v10.4 and later.

Declared in `HIView.h`.

Result Codes

This table lists the result codes defined in HIView.

Result Code	Value	Description
`errNeedsCompositedWindow`	-30598	This result code is returned by an HIView or a Control Manager function when an action that requires a compositing window is attempted on a non-compositing window. It may also be returned when the value of a parameter is not valid for the requested action, even though the window is a compositing window. Available in Mac OS X v10.3 and later.

HTML Rendering Library Reference (Not Recommended)

Framework: Carbon/Carbon.h

Declared in HTMLRendering.h

Overview

Important: The HTML Rendering Library is deprecated as of Mac OS X v10.4. A much more complete solution for displaying HTML and web content in your application is provided by the Web Kit. See *WebKit Objective-C Programming Guide* for guidelines on using the Web Kit.

Whereas the HTML Rendering Library takes a specified HTML file and draws text and images in a window, the Web Kit inserts a browser window into your document. The user can specify any URL and follow links in the Web Kit view window. Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. With the Web Kit, your application can include anything from an HTML viewing window to a full-featured web browser.

Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

The HTML Rendering Library gives your application the ability to draw text and images in a window, as specified by HTML data. Note that this API does not include other features of HTML browsers, such as history tracking or plug-in support. The kinds of tasks you can perform with the HTML Rendering Library include

- formatting and updating the rendering area with such elements as borders, scroll bars, and size boxes in scroll bars
- setting such graphics and display options as screen depth and graphics ports
- working with events
- drawing pages from HTML data
- obtaining information about pages
- converting URL and file system specification data

Functions by Task

Identifying The HTML Rendering Library

`HRHTMLRenderingLibAvailable` (page 2559)
Reports whether the HTML Rendering Library is available. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`HRGetHTMLRenderingLibVersion` (page 2545) Deprecated in Mac OS X v10.4
Identifies which version of the HTML Rendering Library is available. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

Handling Newly Visited Links

`DisposeHRNewURLUPP` (page 2534) Deprecated in Mac OS X v10.4
Disposes of a previously obtained UPP. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`HRRegisterNewURLUPP` (page 2563) Deprecated in Mac OS X v10.4
Registers an application-defined function that handles newly visited links. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`HRUnregisterNewURLUPP` (page 2574) Deprecated in Mac OS X v10.4
Unregisters a previously registered application-defined function. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`NewHRNewURLUPP` (page 2586) Deprecated in Mac OS X v10.4
Obtains a UPP for an application-defined function that handles newly visited links. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

Handling Previously Visited Links

`DisposeHRWasURLVisitedUPP` (page 2536) Deprecated in Mac OS X v10.4
Disposes of a previously obtained UPP. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`HRRegisterWasURLVisitedUPP` (page 2565) Deprecated in Mac OS X v10.4
Registers an application-defined function that handles previously visited links. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`HRUnregisterWasURLVisitedUPP` (page 2577) Deprecated in Mac OS X v10.4
Unregisters a previously registered application-defined function. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`NewHRWasURLVisitedUPP` (page 2588) Deprecated in Mac OS X v10.4
Obtains a UPP for an application-defined function that handles previously visited links. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

Obtaining and Disposing of HRReference Values

`HRDisposeReference` (page 2538) Deprecated in Mac OS X v10.4
Disposes of a previously obtained `HRReference`. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`HRNewReference` (page 2561) Deprecated in Mac OS X v10.4
Obtains a new `HRReference`. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`HRNewReferenceInWindow` (page 2561) Deprecated in Mac OS X v10.4
Obtains a new `HRReference`. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

Formatting the Rendering Area

`HRSetDrawBorder` (page 2568) Deprecated in Mac OS X v10.4
Specifies whether to draw a border around the rendering area. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`HRSetGrowboxCutout` (page 2571) Deprecated in Mac OS X v10.4
Specifies whether to allow for a size box when drawing scrollbars. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`HRSetScrollbarState` (page 2572) Deprecated in Mac OS X v10.4
Specifies how scrollbars are drawn. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

Intercepting and Redirecting URL's

`DisposeHRURLToFSSpecUPP` (page 2535) Deprecated in Mac OS X v10.4
Disposes of a previously obtained UPP. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`HRRegisterURLToFSSpecUPP` (page 2564) Deprecated in Mac OS X v10.4
Registers an application-defined function that handles previously visited links. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`HRUnregisterURLToFSSpecUPP` (page 2576) Deprecated in Mac OS X v10.4
Unregisters a previously registered application-defined function. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`NewHRURLToFSSpecUPP` (page 2587) Deprecated in Mac OS X v10.4
Obtains a UPP for an application-defined function that intercepts URL's. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

Setting and Updating the Rendering Area

`HRDraw` (page 2539) Deprecated in Mac OS X v10.4
Updates a specified region for a given `HRReference`. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`HRDrawInPort` (page 2540) Deprecated in Mac OS X v10.4
Updates a specified region for a given `HRReference`. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`HRGetRenderedImageSize` (page 2547) Deprecated in Mac OS X v10.4
Reports the size of the rendered image. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`HRGetRenderedImageSize32` (page 2548) Deprecated in Mac OS X v10.4
Reports the size of the rendered image. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`HRScrollToImageLocation32` (page 2567) Deprecated in Mac OS X v10.4
Scrolls to a specified view of the HTML rendering area. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`HRScrollToLocation` (page 2568) Deprecated in Mac OS X v10.4
Scrolls to a specified view of the HTML rendering area. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`HRSetRenderingRect` (page 2571) Deprecated in Mac OS X v10.4
Specifies the boundaries of the HTML rendering area. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

Setting Graphics and Display Options

`HRForceQuickdraw` (page 2540) Deprecated in Mac OS X v10.4
Forces all images to be drawn with QuickDraw. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`HRFreeMemory` (page 2541) Deprecated in Mac OS X v10.4
Attempts to release cache memory for use by your application. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`HRScreenConfigurationChanged` (page 2566) Deprecated in Mac OS X v10.4
Informs the HTML Rendering Library that the screen depth has changed. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`HRSetGrafPtr` (page 2570) Deprecated in Mac OS X v10.4
Sets a new `GrafPort` for a given `HRReference`. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

Working With Events

`HRActivate` (page 2537) Deprecated in Mac OS X v10.4
Activates the window associated with a given `HRReference`. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`HRDeactivate` (page 2537) Deprecated in Mac OS X v10.4
Deactivates the window associated with a given `HRReference`. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`HRIsHREvent` (page 2560) Deprecated in Mac OS X v10.4
Gives the HTML Rendering Library an opportunity to handle events. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

Navigating HTML Pages

`HRGoToAnchor` (page 2552) Deprecated in Mac OS X v10.4
Specifies an HTML anchor to scroll into view. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`HRGoToAnchorCFString` (page 2553) Deprecated in Mac OS X v10.4
Specifies an HTML anchor to scroll into view. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`HRGoToCFURL` (page 2553) Deprecated in Mac OS X v10.4
Renders a local file specified as a URL. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`HRGoToData` (page 2554) Deprecated in Mac OS X v10.4
Specifies HTML data for rendering. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`HRGoToFile` (page 2555) Deprecated in Mac OS X v10.4
Specifies an HTML file for rendering. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`HRGoToFSRef` (page 2556) Deprecated in Mac OS X v10.4
Renders a set of HTML data from a specified buffer. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`HRGoToPtr` (page 2557) Deprecated in Mac OS X v10.4
Renders a set of HTML data from a specified buffer. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`HRGoToURL` (page 2558) Deprecated in Mac OS X v10.4
Renders a local file specified as a URL. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

Obtaining Information About Pages

`HRGetBaseURL` (page 2542) Deprecated in Mac OS X v10.4
Obtains the base URL of a given HTML page. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`HRGetBaseURLAsCFString` (page 2543) Deprecated in Mac OS X v10.4
Obtains the base URL of a given HTML page. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`HRGetHTMLFile` (page 2543) Deprecated in Mac OS X v10.4
Obtains a file system specification record for a given HTML page. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`HRGetHTMLFileAsFSRef` (page 2544) Deprecated in Mac OS X v10.4
Obtains an `FSRef` for a given HTML page. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`HRGetHTMLURL` (page 2546) Deprecated in Mac OS X v10.4
Obtains the URL for a given HTML page. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

HRGetHTMLURLAsCFURL (page 2546) Deprecated in Mac OS X v10.4
Obtains the URL for a given HTML page. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

HRGetRootURL (page 2549) Deprecated in Mac OS X v10.4
Obtains the root URL for all relative links on a given HTML page. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

HRGetRootURLAsCFString (page 2549) Deprecated in Mac OS X v10.4
Obtains the root URL for all relative links on a given HTML page. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

HRGetTitle (page 2550) Deprecated in Mac OS X v10.4
Obtains the title of a given HTML page. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

HRGetTitleAsCFString (page 2551) Deprecated in Mac OS X v10.4
Obtains the title of a given HTML page. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

Converting URL and FSSpec Data

HRUtilCreateFullCFURL (page 2577) Deprecated in Mac OS X v10.4
Obtains a full URL from a given set of relative URLs. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

HRUtilCreateFullURL (page 2578) Deprecated in Mac OS X v10.4
Obtains a full URL from a given set of relative URLs. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

HRUtilGetFSRefFromURL (page 2579) Deprecated in Mac OS X v10.4
Obtains a FSRef from a given set of relative URL's. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

HRUtilGetFSSpecFromURL (page 2580) Deprecated in Mac OS X v10.4
Obtains a FSSpec from a given set of relative URL's. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

HRUtilGetURLFromFSRef (page 2581) Deprecated in Mac OS X v10.4
Obtains a full URL from a given FSRef. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

HRUtilGetURLFromFSSpec (page 2581) Deprecated in Mac OS X v10.4
Obtains a full URL from a given FSSpec. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

Working With Universal Procedure Pointers

DisposeHRNewCFURLUPP (page 2534) Deprecated in Mac OS X v10.4
(Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

DisposeHRURLToFSRefUPP (page 2535) Deprecated in Mac OS X v10.4
(Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

DisposeHRWasCFURLVisitedUPP (page 2536) Deprecated in Mac OS X v10.4
(Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`HRRegisterNewCFURLUPP` (page 2562) Deprecated in Mac OS X v10.4
(Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`HRRegisterURLToFSRefUPP` (page 2564) Deprecated in Mac OS X v10.4
(Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`HRRegisterWasCFURLVisitedUPP` (page 2565) Deprecated in Mac OS X v10.4
(Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`HRUnregisterNewCFURLUPP` (page 2574) Deprecated in Mac OS X v10.4
(Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`HRUnregisterURLToFSRefUPP` (page 2575) Deprecated in Mac OS X v10.4
(Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`HRUnregisterWasCFURLVisitedUPP` (page 2576) Deprecated in Mac OS X v10.4
(Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`InvokeHRNewCFURLUPP` (page 2582) Deprecated in Mac OS X v10.4
(Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`InvokeHRNewURLUPP` (page 2583) Deprecated in Mac OS X v10.4
(Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`InvokeHRURLToFSRefUPP` (page 2583) Deprecated in Mac OS X v10.4
(Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`InvokeHRURLToFSSpecUPP` (page 2584) Deprecated in Mac OS X v10.4
(Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`InvokeHRWasCFURLVisitedUPP` (page 2584) Deprecated in Mac OS X v10.4
(Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`InvokeHRWasURLVisitedUPP` (page 2585) Deprecated in Mac OS X v10.4
(Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`NewHRNewCFURLUPP` (page 2585) Deprecated in Mac OS X v10.4
(Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`NewHRURLToFSRefUPP` (page 2586) Deprecated in Mac OS X v10.4
(Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`NewHRWasCFURLVisitedUPP` (page 2587) Deprecated in Mac OS X v10.4
(Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

Miscellaneous

`HRSetEmbeddingControl` (page 2569) Deprecated in Mac OS X v10.4
Sets the `ControlRef` for embedding controls. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

`HRSetWindowRef` (page 2573) Deprecated in Mac OS X v10.4
Sets the controlling window. (Deprecated. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

Functions

DisposeHRNewCFURLUPP

(Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
void DisposeHRNewCFURLUPP (
   HRNewCFURLUPP userUPP
);
```

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability

Available in CarbonLib 1.3 and later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`HTMLRendering.h`

DisposeHRNewURLUPP

Disposes of a previously obtained UPP. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
void DisposeHRNewURLUPP (
   HRNewURLUPP userUPP
);
```

Parameters

userUPP

A Universal Procedure Pointer (UPP) that you previously obtained by calling the function `NewHRNewURLUPP` (page 2586).

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In
HTMLRendering.h

DisposeHRURLToFSRefUPP

(Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
void DisposeHRURLToFSRefUPP (
   HRURLToFSRefUPP userUPP
);
```

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability
Available in CarbonLib 1.3 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
HTMLRendering.h

DisposeHRURLToFSSpecUPP

Disposes of a previously obtained UPP. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
void DisposeHRURLToFSSpecUPP (
   HRURLToFSSpecUPP userUPP
);
```

Parameters

userUPP

A Universal Procedure Pointer (UPP) that you previously obtained by calling the function NewHRURLToFSSpecUPP (page 2587).

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability
Available in CarbonLib 1.1 and later.

Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
HTMLRendering.h

DisposeHRWasCFURLVisitedUPP

(Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
void DisposeHRWasCFURLVisitedUPP (
   HRWasCFURLVisitedUPP userUPP
);
```

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability
Available in CarbonLib 1.3 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
HTMLRendering.h

DisposeHRWasURLVisitedUPP

Disposes of a previously obtained UPP. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
void DisposeHRWasURLVisitedUPP (
   HRWasURLVisitedUPP userUPP
);
```

Parameters

userUPP

A Universal Procedure Pointer (UPP) that you previously obtained by calling the function NewHRWasURLVisitedUPP (page 2588).

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`HTMLRendering.h`

HRActivate

Activates the window associated with a given `HRReference`. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
OSStatus HRActivate (
   HRReference hrRef
);
```

Parameters

hrRef

An `HRReference` value previously obtained by your application.

Return Value
A result code. See "HTML Rendering Library Result Codes" (page 2597).

Discussion
Call this function whenever the window associated with a given `HRReference` becomes active. This allows the HTML Rendering Library to activate scrollbars and handle events as appropriate.

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability
Available in CarbonLib 1.1 and later when running Mac OS 9 or later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`HTMLRendering.h`

HRDeactivate

Deactivates the window associated with a given `HRReference`. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
OSStatus HRDeactivate (
   HRReference hrRef
);
```

Parameters

hrRef

An `HRReference` value previously obtained by your application.

Return Value
A result code. See "HTML Rendering Library Result Codes" (page 2597).

Discussion
Call this function whenever the window associated with a given `HRReference` becomes inactive.

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability
Available in CarbonLib 1.1 and later when running Mac OS 9 or later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`HTMLRendering.h`

HRDisposeReference

Disposes of a previously obtained `HRReference`. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
OSStatus HRDisposeReference (
   HRReference hrRef
);
```

Parameters

hrRef

An `HRReference` value previously obtained by your application.

Return Value
A result code. See "HTML Rendering Library Result Codes" (page 2597).

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability
Available in CarbonLib 1.1 and later when running Mac OS 9 or later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`HTMLRendering.h`

HRDraw

Updates a specified region for a given `HRReference`. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
OSStatus HRDraw (
   HRReference hrRef,
   RgnHandle updateRgnH
);
```

Parameters

hrRef

An `HRReference` value previously obtained by your application.

updateRgnH

A handle of type `RgnHandle`. This handle specifies the region that you want the HTML Rendering Library to update. Specify the region in the `GrafPort's` local coordinates. If you pass `NULL`, the HTML Rendering Library updates the entire rectangle you previously specified with the function `HRSetRenderingRect` (page 2571).

Return Value
A result code. See "HTML Rendering Library Result Codes" (page 2597).

Discussion
Be sure to specify the dimensions of the rendering area by calling the function `HRSetRenderingRect` (page 2571) at least once before you call the `HRDraw` function.

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability
Available in CarbonLib 1.1 and later when running Mac OS 9 or later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`HTMLRendering.h`

HRDrawInPort

Updates a specified region for a given `HRReference`. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
OSStatus HRDrawInPort (
   HRReference hrRef,
   RgnHandle updateRgnH,
   CGrafPtr grafPtr
);
```

Parameters

hrRef

A reference to the renderer object.

updateRgnH

The region to be updated.

grafPtr

A graf pointer to render HTML into.

Return Value

A result code. See "HTML Rendering Library Result Codes" (page 2597).

Discussion

All the drawing will now happen in the specified port. This is the API you want to use to draw in an offscreen port, for example when printing. You could also use this API to draw in an on screen port.

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability

Available in CarbonLib 1.3 and later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`HTMLRendering.h`

HRForceQuickdraw

Forces all images to be drawn with QuickDraw. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
OSStatus HRForceQuickdraw (
   HRReference hrRef,
   Boolean forceQuickdraw
);
```

Parameters

hrRef

An `HRReference` value previously obtained by your application.

forceQuickdraw

Pass a value of `true` in this parameter to force all images to be drawn with QuickDraw.

Return Value
A result code. See "HTML Rendering Library Result Codes" (page 2597).

Discussion
This function may be useful when working with print drivers that require QuickDraw for image capturing.

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability
Available in CarbonLib 1.1 and later when running Mac OS 9 or later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`HTMLRendering.h`

HRFreeMemory

Attempts to release cache memory for use by your application. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
SInt32 HRFreeMemory (
   Size inBytesNeeded
);
```

Parameters

inBytesNeeded

A value indicating how many bytes of cache memory your application desires to obtain.

Return Value
A value indicating how many bytes of cache memory the HTML Rendering Library makes available to your application.

Discussion
Call the `HRFreeMemory` function to flush images and movies from the HTML Rendering Library cache. Be sure to check the function result to determine whether sufficient memory was made available to your application.

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability

Available in CarbonLib 1.1 and later when running Mac OS 9 or later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`HTMLRendering.h`

HRGetBaseURL

Obtains the base URL of a given HTML page. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
OSStatus HRGetBaseURL (
   HRReference hrRef,
   Handle baseURLH
);
```

Parameters

hrRef

A `HRReference` value previously obtained by your application.

baseURLH

A handle. Make sure you allocate a valid handle with the `NewHandle` function. On return, this handle (which may be resized) references a C string containing the root URL of the given file. If the current page does not contain a `BASE` tag, the handle is empty.

Return Value

A result code. See "HTML Rendering Library Result Codes" (page 2597).

Discussion

The URL returned by this function begins with `'file:///'`.

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability

Available in CarbonLib 1.1 and later when running Mac OS 9 or later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In
`HTMLRendering.h`

HRGetBaseURLAsCFString

Obtains the base URL of a given HTML page. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
OSStatus HRGetBaseURLAsCFString (
   HRReference hrRef,
   CFStringRef *baseString
);
```

Parameters

hrRef

A `HRReference` value previously obtained by your application.

baseString

On return, this handle references a `CFString` containing the root URL of the given file.

Return Value
A result code. See "HTML Rendering Library Result Codes" (page 2597).

Discussion
Use these API from a Carbon application instead of using `HRGetRootURL` (page 2549), `HRGetBaseURL` (page 2542), `HRGetHTMLURL` (page 2546), `HRGetTitle` (page 2550)and `HRGetHTMLFile` (page 2543). These APIs are same in behavior with their old counter parts. The only difference is that they take `CFString`, `CFURLRef`, and `FSRef` as parameters.

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability
Available in CarbonLib 1.3 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`HTMLRendering.h`

HRGetHTMLFile

Obtains a file system specification record for a given HTML page. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
OSStatus HRGetHTMLFile (
   HRReference hrRef,
   FSSpec *fsspec
);
```

Parameters

hrRef

An `HRReference` value previously obtained by your application.

fsspec

A pointer to a file system specification record (`FSSpec`). On return, the `FSSpec` contains file specification data for the page specified in the `hrRef` parameter.

Return Value
A result code. See "HTML Rendering Library Result Codes" (page 2597).

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability
Available in CarbonLib 1.1 and later when running Mac OS 9 or later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`HTMLRendering.h`

HRGetHTMLFileAsFSRef

Obtains an `FSRef` for a given HTML page. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
OSStatus HRGetHTMLFileAsFSRef (
   HRReference hrRef,
   FSRef *fref
);
```

Parameters

hrRef

An `HRReference` value previously obtained by your application.

fref

A pointer to an (`FSRef`). On return, the `FSRef` contains file specification data for the page specified in the `hrRef` parameter.

Return Value
A result code. See "HTML Rendering Library Result Codes" (page 2597).

Discussion

Use these API from a Carbon application instead of using `HRGetRootURL` (page 2549), `HRGetBaseURL` (page 2542), `HRGetHTMLURL` (page 2546), `HRGetTitle` (page 2550)and `HRGetHTMLFile` (page 2543). These APIs are same in behavior with their old counter parts. The only difference is that they take `CFString`, `CFURLRef`, and `FSRef` as parameters.

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability

Available in CarbonLib 1.3 and later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`HTMLRendering.h`

HRGetHTMLRenderingLibVersion

Identifies which version of the HTML Rendering Library is available. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
OSStatus HRGetHTMLRenderingLibVersion (
   NumVersion *returnVers
);
```

Parameters

returnVers

A pointer to a `NumVersion` structure. On return, this structure contains a value identifying the installed version of the HTML Rendering Library.

Return Value

A result code. See "HTML Rendering Library Result Codes" (page 2597).

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability

Available in CarbonLib 1.1 and later when running Mac OS 9 or later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In
`HTMLRendering.h`

HRGetHTMLURL

Obtains the URL for a given HTML page. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
OSStatus HRGetHTMLURL (
   HRReference hrRef,
   Handle HTMLURLH
);
```

Parameters

hrRef

An `HRReference` value previously obtained by your application.

HTMLURLH

A handle. Make sure you allocate a valid handle with the `NewHandle` function. On return, this handle (which may be resized) references a C string containing the URL of the given file.

Return Value
A result code. See "HTML Rendering Library Result Codes" (page 2597).

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability
Available in CarbonLib 1.1 and later when running Mac OS 9 or later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`HTMLRendering.h`

HRGetHTMLURLAsCFURL

Obtains the URL for a given HTML page. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
OSStatus HRGetHTMLURLAsCFURL (
   HRReference hrRef,
   CFURLRef *theURL
);
```

Parameters

hrRef

An `HRReference` value previously obtained by your application.

theURL

On return, this references a `CFURL` containing the URL of the given file.

Return Value
A result code. See "HTML Rendering Library Result Codes" (page 2597).

Discussion
Use these API from a Carbon application instead of using `HRGetRootURL` (page 2549), `HRGetBaseURL` (page 2542), `HRGetHTMLURL` (page 2546), `HRGetTitle` (page 2550)and `HRGetHTMLFile` (page 2543). These APIs are same in behavior with their old counter parts. The only difference is that they take `CFString`, `CFURLRef`, and `FSRef` as parameters.

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability
Available in CarbonLib 1.3 and later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In
`HTMLRendering.h`

HRGetRenderedImageSize

Reports the size of the rendered image. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
OSStatus HRGetRenderedImageSize (
   HRReference hrRef,
   Point *renderingSize
);
```

Parameters

hrRef

An `HRReference` value previously obtained by your application.

renderingSize

A pointer to a value of type `Point`. On return, the HTML Rendering Library uses this value to report the size of the entire HTML page; that is, the smallest rectangle that fully encloses the entire rendered page (as drawn without scrollbars).

Return Value
A result code. See "HTML Rendering Library Result Codes" (page 2597).

Discussion
The HTML Rendering Library attempts to limit the width of the rendered page to the width of the rectangle you specify by calling the function `HRSetRenderingRect` (page 2571). The length of the page is calculated based on the resulting width.

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability

Available in CarbonLib 1.1 and later when running Mac OS 9 or later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`HTMLRendering.h`

HRGetRenderedImageSize32

Reports the size of the rendered image. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
OSStatus HRGetRenderedImageSize32 (
   HRReference hrRef,
   UInt32 *height,
   UInt32 *width
);
```

Parameters

hrRef

A reference to the renderer object.

height

On return, the height of the image.

width

On return, the width of the image.

Return Value

A result code. See "HTML Rendering Library Result Codes" (page 2597).

Discussion

Use this API when the rendered image could have coordinates larger than what SInt16 can hold.

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability

Available in CarbonLib 1.3 and later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In
`HTMLRendering.h`

HRGetRootURL

Obtains the root URL for all relative links on a given HTML page. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
OSStatus HRGetRootURL (
   HRReference hrRef,
   Handle rootURLH
);
```

Parameters

hrRef

An `HRReference` value previously obtained by your application.

rootURLH

A handle. Make sure you allocate a valid handle with the `NewHandle` function. On return, this handle (which may be resized) references a C string containing the root URL of the given file.

Return Value
A result code. See "HTML Rendering Library Result Codes" (page 2597).

Discussion
The root URL is normally the URL of the page specified in the `hrRef` parameter, unless the page contains a `<BASE>` tag specifying a different root URL.

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability
Available in CarbonLib 1.1 and later when running Mac OS 9 or later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`HTMLRendering.h`

HRGetRootURLAsCFString

Obtains the root URL for all relative links on a given HTML page. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
OSStatus HRGetRootURLAsCFString (
   HRReference hrRef,
   CFStringRef *rootString
);
```

Parameters

hrRef

An `HRReference` value previously obtained by your application.

rootString

On return, this references a `CFString` containing the root URL of the given file.

Return Value

A result code. See "HTML Rendering Library Result Codes" (page 2597).

Discussion

Use these API from a Carbon application instead of using `HRGetRootURL` (page 2549), `HRGetBaseURL` (page 2542), `HRGetHTMLURL` (page 2546), `HRGetTitle` (page 2550)and `HRGetHTMLFile` (page 2543). These APIs are same in behavior with their old counter parts. The only difference is that they take `CFString`, `CFURLRef`, and `FSRef` as parameters.

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability

Available in CarbonLib 1.3 and later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`HTMLRendering.h`

HRGetTitle

Obtains the title of a given HTML page. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
OSStatus HRGetTitle (
   HRReference hrRef,
   StringPtr title
);
```

Parameters

hrRef

An `HRReference` value previously obtained by your application.

title

A `StringPtr`. On return, the HTML Rendering Library stores the contents of the `TITLE` tag of the given page in the referenced string.

Return Value
A result code. See "HTML Rendering Library Result Codes" (page 2597).

Discussion
This function converts any HTML entities into Mac OS characters before storing them in the string specified in the `title` parameter.

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability
Available in CarbonLib 1.1 and later when running Mac OS 9 or later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`HTMLRendering.h`

HRGetTitleAsCFString

Obtains the title of a given HTML page. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
OSStatus HRGetTitleAsCFString (
   HRReference hrRef,
   CFStringRef *title
);
```

Parameters

hrRef
An `HRReference` value previously obtained by your application.

title
A `CFStringRef`. On return, the HTML Rendering Library stores the contents of the `TITLE` tag of the given page in the referenced string.

Return Value
A result code. See "HTML Rendering Library Result Codes" (page 2597).

Discussion
Use these API from a Carbon application instead of using `HRGetRootURL` (page 2549), `HRGetBaseURL` (page 2542), `HRGetHTMLURL` (page 2546), `HRGetTitle` (page 2550)and `HRGetHTMLFile` (page 2543). These APIs are same in behavior with their old counter parts. The only difference is that they take `CFString`, `CFURLRef`, and `FSRef` as parameters.

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very

quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability
Available in CarbonLib 1.3 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`HTMLRendering.h`

HRGoToAnchor

Specifies an HTML anchor to scroll into view. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
OSStatus HRGoToAnchor (
   HRReference hrRef,
   const char *anchorName
);
```

Parameters

hrRef
An `HRReference` value previously obtained by your application.

anchorName
A pointer to a C string containing the name of an anchor on the current HTML page.

Return Value
A result code. See "HTML Rendering Library Result Codes" (page 2597).

Discussion
Note that anchors are case sensitive.

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability
Available in CarbonLib 1.1 and later when running Mac OS 9 or later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`HTMLRendering.h`

HRGoToAnchorCFString

Specifies an HTML anchor to scroll into view. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
OSStatus HRGoToAnchorCFString (
   HRReference hrRef,
   CFStringRef anchorName
);
```

Parameters

hrRef

A reference to the renderer object.

anchorName

The name of the anchor to display.

Return Value

A result code. See "HTML Rendering Library Result Codes" (page 2597).

Discussion

Use these functions instead of using the functions `HRGoToFile` (page 2555), `HRGoToURL` (page 2558), `HRGoToAnchor` (page 2552)and `HRGoToPtr` (page 2557). These functions are the same in behavior with their old counter parts. The only difference is that they take `FSRef`, `CFURLRef`, `CFString`, and `CFData` as parameters.

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability

Available in CarbonLib 1.3 and later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`HTMLRendering.h`

HRGoToCFURL

Renders a local file specified as a URL. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
OSStatus HRGoToCFURL (
   HRReference hrRef,
   CFURLRef url,
   Boolean addToHistory,
   Boolean forceRefresh
);
```

Parameters

hrRef

A reference to the renderer object.

url

A reference to the URL to render.

addToHistory

Pass `true` if this file URL should be added to the history.

forceRefresh

Pass `true` if the rendering area should be refreshed.

Return Value

A result code. See "HTML Rendering Library Result Codes" (page 2597).

Discussion

Use these functions instead of using the functions `HRGoToFile` (page 2555), `HRGoToURL` (page 2558), `HRGoToAnchor` (page 2552)and `HRGoToPtr` (page 2557). These functions are the same in behavior with their old counter parts. The only difference is that they take `FSRef`, `CFURLRef`, `CFString`, and `CFData` as parameters.

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability

Available in CarbonLib 1.3 and later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`HTMLRendering.h`

HRGoToData

Specifies HTML data for rendering. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
OSStatus HRGoToData (
   HRReference hrRef,
   CFDataRef data,
   Boolean addToHistory,
   Boolean forceRefresh
);
```

Parameters

hrRef

A reference to the renderer object.

data

A reference to data in the memory to render.

addToHistory

Pass `true` if this file URL should be added to the history.

forceRefresh

Pass `true` if the rendering area should be refreshed.

Return Value

A result code. See "HTML Rendering Library Result Codes" (page 2597).

Discussion

Use these functions instead of using the functions `HRGoToFile` (page 2555), `HRGoToURL` (page 2558), `HRGoToAnchor` (page 2552)and `HRGoToPtr` (page 2557). These functions are the same in behavior with their old counter parts. The only difference is that they take `FSRef`, `CFURLRef`, `CFString`, and `CFData` as parameters.

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability

Available in CarbonLib 1.3 and later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`HTMLRendering.h`

HRGoToFile

Specifies an HTML file for rendering. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
OSStatus HRGoToFile (
   HRReference hrRef,
   const FSSpec *fsspec,
   Boolean addToHistory,
   Boolean forceRefresh
);
```

Parameters

hrRef

An `HRReference` value previously obtained by your application.

fsspec

A pointer to a file system specification record (`FSSpec`) for the HTML file that you wish to have rendered by the HTML Rendering Library.

addToHistory

Pass `true` in this parameter if you wish to have this file added to the list of visited links.

forceRefresh

Pass `true` in this parameter if you wish to force a screen update.

Return Value

A result code. See "HTML Rendering Library Result Codes" (page 2597).

Discussion

Note that the HTML Rendering Library calls your `HRNewURLProcPtr` (page 2589) application-defined function (if you use one) with this FSSpec. If you need to pass a URL instead of an FSSpec, use the function `HRGoToURL` (page 2558).

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability

Available in CarbonLib 1.1 and later when running Mac OS 9 or later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`HTMLRendering.h`

HRGoToFSRef

Renders a set of HTML data from a specified buffer. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
OSStatus HRGoToFSRef (
   HRReference hrRef,
   const FSRef *fref,
   Boolean addToHistory,
   Boolean forceRefresh
);
```

Parameters

hrRef

A reference to the renderer object.

fref

A reference to the HTML file that is to be opened and rendered.

addToHistory

Pass `true` if this file URL should be added to the history.

forceRefresh

Pass `true` if the rendering area should be refreshed.

Return Value

A result code. See "HTML Rendering Library Result Codes" (page 2597).

Discussion

Use these functions instead of using the functions `HRGoToFile` (page 2555), `HRGoToURL` (page 2558), `HRGoToAnchor` (page 2552)and `HRGoToPtr` (page 2557). These functions are the same in behavior with their old counter parts. The only difference is that they take `FSRef`, `CFURLRef`, `CFString`, and `CFData` as parameters.

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability

Available in CarbonLib 1.3 and later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`HTMLRendering.h`

HRGoToPtr

Renders a set of HTML data from a specified buffer. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
OSStatus HRGoToPtr (
   HRReference hrRef,
   char *buffer,
   UInt32 bufferSize,
   Boolean addToHistory,
   Boolean forceRefresh
);
```

Parameters

hrRef

An `HRReference` value previously obtained by your application.

buffer

A pointer to a buffer containing HTML data.

bufferSize

A value specifying the number of bytes in the buffer.

addToHistory

Pass `true` in this parameter if you wish to have this data added to the list of visited links.

forceRefresh

Pass `true` in this parameter if you wish to force a screen update.

Return Value

A result code. See "HTML Rendering Library Result Codes" (page 2597).

Discussion

The HTML Rendering Library displays the HTML source data pointed to by the handle. It does not copy the handle, so you should not dispose of it until you are sure you no longer need it. Note that any relative links you specify in the HTML data are relative to your application, as the HTML data has no associated URL.

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability

Available in CarbonLib 1.1 and later when running Mac OS 9 or later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`HTMLRendering.h`

HRGoToURL

Renders a local file specified as a URL. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
OSStatus HRGoToURL (
   HRReference hrRef,
   const char *url,
   Boolean addToHistory,
   Boolean forceRefresh
);
```

Parameters

hrRef

An `HRReference` value previously obtained by your application.

url

A pointer to a C string containing a Universal Resource Locator (URL).

addToHistory

Pass `true` in this parameter if you wish to have this file added to the list of visited links.

forceRefresh

Pass `true` in this parameter if you wish to force a screen update.

Return Value
A result code. See "HTML Rendering Library Result Codes" (page 2597).

Discussion
Use this function for rendering local files. The URL should begin with '`file:///`'. Note that the HTML Rendering Library calls your `HRNewURLProcPtr` (page 2589) application-defined function (if you use one) with this URL. If you need to pass an `FSSpec` instead of a URL, use the function `HRGoToURL` (page 2558).

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability
Available in CarbonLib 1.1 and later when running Mac OS 9 or later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`HTMLRendering.h`

HRHTMLRenderingLibAvailable

Reports whether the HTML Rendering Library is available. (**Deprecated.** Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
pascal Boolean HRHTMLRenderingLibAvailable
```

Return Value
Returns `true` if the HTML Rendering Library is available; returns `false` otherwise.

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`HTMLRendering.h`

HRIsHREvent

Gives the HTML Rendering Library an opportunity to handle events. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
Boolean HRIsHREvent (
   const EventRecord *eventRecord
);
```

Parameters

eventRecord

A pointer to an `EventRecord`.

Return Value

Returns `true` if the HTML Rendering Library handles the specified event; returns `false` otherwise.

Discussion

Call the `HRIsHREvent` function with every event received by your application. This ensures that the HTML Rendering Library has an opportunity to handle user clicks and cursor changes.

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability

Available in CarbonLib 1.1 and later when running Mac OS 9 or later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`HTMLRendering.h`

HRNewReference

Obtains a new `HRReference`. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
OSStatus HRNewReference (
   HRReference *hrRef,
   OSType rendererType,
   GrafPtr grafPtr
);
```

Parameters

hrRef

A pointer to a `HRReference` value. On return, your application uses this value to call other functions in the HTML Rendering Library.

rendererType

A value of type `OSType` identifying the type of HTML renderer available. Currently, the constant `kHRRendererHTML32Type` is the only value you may pass in this parameter.

grafPtr

A pointer to a valid `GrafPort`.

Return Value

A result code. See "HTML Rendering Library Result Codes" (page 2597).

Discussion

When your application has no further need for a particular `HRReference`, dispose of it by calling the function `HRDisposeReference` (page 2538).

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability

Available in CarbonLib 1.1 and later when running Mac OS 9 or later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`HTMLRendering.h`

HRNewReferenceInWindow

Obtains a new `HRReference`. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
OSStatus HRNewReferenceInWindow (
   HRReference *hrRef,
   OSType rendererType,
   WindowRef inWindowRef
);
```

Parameters

hrRef

A pointer to the new reference created and returned by the renderer.

rendererType

The type of the renderer—for example `kHRRendererHTML32Type`. Only this type is supported for now.

inWindowRef

A reference to the window for which rendering area will be specified.

Return Value

A result code. See "HTML Rendering Library Result Codes" (page 2597).

Discussion

All the controls created by the HTML renderer will be embedded in the root control of the window specified by the window reference.

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability

Available in CarbonLib 1.3 and later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`HTMLRendering.h`

HRRegisterNewCFURLUPP

(Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
void HRRegisterNewCFURLUPP (
   HRNewCFURLUPP inURLUPP,
   HRReference hrRef,
   void *inRefCon
);
```

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very

quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability
Available in CarbonLib 1.3 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`HTMLRendering.h`

HRRegisterNewURLUPP

Registers an application-defined function that handles newly visited links. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
void HRRegisterNewURLUPP (
   HRNewURLUPP inNewURLUPP,
   HRReference hrRef,
   void *inRefCon
);
```

Parameters

inNewURLUPP
A Universal Procedure Pointer (UPP). You obtain this UPP by calling the function `NewHRNewURLUPP` (page 2586).

hrRef
An `HRReference` value previously obtained by your application.

inRefCon
An arbitrary value set by your application. This value is passed by your application when you call the `HRRegisterNewURLUPP` function and passed back when the HTML Rendering Library calls your function `HRNewURLProcPtr` (page 2589). You may find this value useful for referring to an object instance or a structure, for example.

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability
Available in CarbonLib 1.1 and later when running Mac OS 9 or later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`HTMLRendering.h`

HRRegisterURLToFSRefUPP

(Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
void HRRegisterURLToFSRefUPP (
   HRURLToFSRefUPP inURLToFSRefUPP,
   HRReference hrRef,
   void *inRefCon
);
```

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability

Available in CarbonLib 1.3 and later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`HTMLRendering.h`

HRRegisterURLToFSSpecUPP

Registers an application-defined function that handles previously visited links. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
void HRRegisterURLToFSSpecUPP (
   HRURLToFSSpecUPP inURLToFSSpecUPP,
   HRReference hrRef,
   void *inRefCon
);
```

Parameters

inURLToFSSpecUPP

A Universal Procedure Pointer (UPP). You obtain this UPP by calling the function `NewHRURLToFSSpecUPP` (page 2587).

hrRef

An `HRReference` value previously obtained by your application.

inRefCon

An arbitrary value set by your application. This value is passed by your application when you call the `HRRegisterURLToFSSpecUPP` function and passed back when the HTML Rendering Library calls your function `HRURLToFSSpecProcPtr` (page 2591). You may find this value useful for referring to an object instance or a structure, for example.

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very

quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability
Available in CarbonLib 1.1 and later when running Mac OS 9 or later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`HTMLRendering.h`

HRRegisterWasCFURLVisitedUPP

(Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
void HRRegisterWasCFURLVisitedUPP (
   HRWasCFURLVisitedUPP inWasCFURLVisitedUPP,
   HRReference hrRef,
   void *inRefCon
);
```

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability
Available in CarbonLib 1.3 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`HTMLRendering.h`

HRRegisterWasURLVisitedUPP

Registers an application-defined function that handles previously visited links. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
void HRRegisterWasURLVisitedUPP (
   HRWasURLVisitedUPP inWasURLVisitedUPP,
   HRReference hrRef,
   void *inRefCon
);
```

Parameters

inWasURLVisitedUPP

A Universal Procedure Pointer (UPP). You obtain this UPP by calling the function `NewHRWasURLVisitedUPP` (page 2588).

hrRef

An `HRReference` value previously obtained by your application.

inRefCon

An arbitrary value set by your application. This value is passed by your application when you call the function and passed back when the HTML Rendering Library calls your `MyHRWasURLVisitedProc` function. You may find this value useful for referring to an object instance or a structure, for example.

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability

Available in CarbonLib 1.1 and later when running Mac OS 9 or later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`HTMLRendering.h`

HRScreenConfigurationChanged

Informs the HTML Rendering Library that the screen depth has changed. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
void HRScreenConfigurationChanged (
   void
);
```

Discussion

Call the `HRScreenConfigurationChanged` function every time the screen depth changes or the user's monitor configuration changes. This allows the HTML Rendering Library to redraw correctly.

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability
Available in CarbonLib 1.1 and later when running Mac OS 9 or later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`HTMLRendering.h`

HRScrollToImageLocation32

Scrolls to a specified view of the HTML rendering area. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
OSStatus HRScrollToImageLocation32 (
   HRReference hrRef,
   SInt32 h,
   SInt32 v
);
```

Parameters

hrRef
A reference to the renderer object.

h
The horizontal location.

v
The vertical location.

Return Value
A result code. See "HTML Rendering Library Result Codes" (page 2597).

Discussion
Use this API when specifying a location to scroll to. Location is specified in image space.

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability
Available in CarbonLib 1.3 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`HTMLRendering.h`

HRScrollToLocation

Scrolls to a specified view of the HTML rendering area. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
OSStatus HRScrollToLocation (
   HRReference hrRef,
   Point *location
);
```

Parameters

hrRef

An `HRReference` value previously obtained by your application.

location

A pointer to a value of type `Point`. Your application uses this value to specify the upper left corner of the visible portion of the rendering area.

Return Value

A result code. See "HTML Rendering Library Result Codes" (page 2597).

Discussion

The HTML Rendering Library limits scrolling to the available rendering area.

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability

Available in CarbonLib 1.1 and later when running Mac OS 9 or later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`HTMLRendering.h`

HRSetDrawBorder

Specifies whether to draw a border around the rendering area. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
OSStatus HRSetDrawBorder (
   HRReference hrRef,
   Boolean drawBorder
);
```

Parameters

hrRef

An `HRReference` value previously obtained by your application.

drawBorder

If you pass `true`, the HTML Rendering Library draws a border around the rendering area. If you pass `false`, the HTML Rendering Library draws the rendering area without a border. The default setting is `false`.

Return Value
A result code. See "HTML Rendering Library Result Codes" (page 2597).

Discussion
You may use this optional function to specify whether a border is drawn around the rendering area. This might be useful when you specify a rendering area in a window containing other elements, for example. The default setting is to render without a border.

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability
Available in CarbonLib 1.1 and later when running Mac OS 9 or later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`HTMLRendering.h`

HRSetEmbeddingControl

Sets the `ControlRef` for embedding controls. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
OSStatus HRSetEmbeddingControl (
   HRReference hrRef,
   ControlRef controlRef
);
```

Parameters

hrRef

A reference to the renderer object.

controlRef

All the future controls created by renderer are embedded in this `controlRef`.

Return Value
A result code. See "HTML Rendering Library Result Codes" (page 2597).

Discussion
Use this API to tell the HTML Renderer to embed all the controls it has created so far and the new controls it creates after this call in the given control. This is particularly useful if you wish to have HTML displayed within your dialog. For example, Software Update needs this.

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability

Available in CarbonLib 1.3 and later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`HTMLRendering.h`

HRSetGrafPtr

Sets a new `GrafPort` for a given `HRReference`. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
OSStatus HRSetGrafPtr (
   HRReference hrRef,
   GrafPtr grafPtr
);
```

Parameters

hrRef

An `HRReference` value previously obtained by your application.

grafPtr

A pointer to a valid `GrafPort`. This value replaces any `GrafPort` previously specified for the given `HRReference`.

Return Value

A result code. See "HTML Rendering Library Result Codes" (page 2597).

Discussion

Call this function whenever the `GrafPort` changes for a given `HRReference`, as occurs during printing, for example. Be sure to call the `HRSetGrafPtr` function again to reset the original `GrafPort` when necessary.

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability

Available in CarbonLib 1.1 and later when running Mac OS 9 or later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In
`HTMLRendering.h`

HRSetGrowboxCutout

Specifies whether to allow for a size box when drawing scrollbars. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
OSStatus HRSetGrowboxCutout (
   HRReference hrRef,
   Boolean allowCutout
);
```

Parameters

hrRef

An `HRReference` value previously obtained by your application.

allowCutout

If you pass `true` in this parameter, the HTML Rendering Library draws scroll bars that provide space for a size box. If you pass `false` in this parameter, the HTML Rendering Library draws scroll bars that extend all the way to the bottom right corner of the rendering area. The default setting is `false`.

Return Value
A result code. See "HTML Rendering Library Result Codes" (page 2597).

Discussion
You may find it useful to specify a cutout for a size box if the HTML rendering area you specify extends to the lower right corner of a window. The default setting is to draw scrollbars without leaving room for a size box.

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability
Available in CarbonLib 1.1 and later when running Mac OS 9 or later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`HTMLRendering.h`

HRSetRenderingRect

Specifies the boundaries of the HTML rendering area. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
OSStatus HRSetRenderingRect (
   HRReference hrRef,
   const Rect *renderingRect
);
```

Parameters

hrRef

An `HRReference` value previously obtained by your application.

renderingRect

A pointer to a value of type `Rect`. This value specifies the boundaries of the HTML rendering area. Specify the boundaries in the `GrafPort's` port coordinates.

Return Value

A result code. See "HTML Rendering Library Result Codes" (page 2597).

Discussion

The HTML Rendering Library draws all elements, including scroll bars, inside the area you specify with the function `HRSetRenderingRect` (page 2571).

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability

Available in CarbonLib 1.1 and later when running Mac OS 9 or later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`HTMLRendering.h`

HRSetScrollbarState

Specifies how scrollbars are drawn. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
OSStatus HRSetScrollbarState (
   HRReference hrRef,
   HRScrollbarState hScrollbarState,
   HRScrollbarState vScrollbarState
);
```

Parameters

hrRef

An `HRReference` value previously obtained by your application.

hScrollbarState

A value of type `HRScrollbarState`. See Scrollbar State (page 2596) for a list of the constants you may pass in this parameter.

vScrollbarState
A value of type `HRScrollbarState`. See Scrollbar State (page 2596) for a list of the constants you may pass in this parameter.

Return Value
A result code. See "HTML Rendering Library Result Codes" (page 2597).

Discussion
You can specify one of three values for scrollbars:

- `eHRScrollbarOn` tells the HTML Rendering Library to draw scrollbars at all times. If the data does not fill the current view, the scroll bars are inactive.
- `eHRScrollbarOff` tells the HTML Rendering Library never to draw scrollbars. You may find this option useful if the HTML rendering area you specify does not extend to the edge of a window.
- `eHRScrollbarAuto` tells the HTML Rendering Library to draw scrollbars as needed. This is the default setting.

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability
Available in CarbonLib 1.1 and later when running Mac OS 9 or later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`HTMLRendering.h`

HRSetWindowRef

Sets the controlling window. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
OSStatus HRSetWindowRef (
   HRReference hrRef,
   WindowRef windowRef
);
```

Parameters

hrRef
A reference to the renderer object.

windowRef
A new reference to the window to be attached to the `hrRef`.

Return Value
A result code. See "HTML Rendering Library Result Codes" (page 2597).

Discussion
All the controls created by the HTML renderer will be moved in the root control of the window specified by the window reference. All the drawing will now happen in the specified window.

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability
Available in CarbonLib 1.3 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`HTMLRendering.h`

HRUnregisterNewCFURLUPP

(Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
void HRUnregisterNewCFURLUPP (
   HRReference hrRef
);
```

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability
Available in CarbonLib 1.3 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`HTMLRendering.h`

HRUnregisterNewURLUPP

Unregisters a previously registered application-defined function. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
void HRUnregisterNewURLUPP (
   HRReference hrRef
);
```

Parameters

hrRef

An `HRReference` value. You pass the same `HRReference` value that you passed to register the application-defined function.

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability

Available in CarbonLib 1.1 and later when running Mac OS 9 or later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`HTMLRendering.h`

HRUnregisterURLToFSRefUPP

(Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
void HRUnregisterURLToFSRefUPP (
   HRReference hrRef
);
```

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability

Available in CarbonLib 1.3 and later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`HTMLRendering.h`

HRUnregisterURLToFSSpecUPP

Unregisters a previously registered application-defined function. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
void HRUnregisterURLToFSSpecUPP (
   HRReference hrRef
);
```

Parameters

hrRef

An `HRReference` value. You pass the same `HRReference` value that you passed to register the application-defined function.

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability

Available in CarbonLib 1.1 and later when running Mac OS 9 or later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`HTMLRendering.h`

HRUnregisterWasCFURLVisitedUPP

(Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
void HRUnregisterWasCFURLVisitedUPP (
   HRReference hrRef
);
```

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability

Available in CarbonLib 1.3 and later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`HTMLRendering.h`

HRUnregisterWasURLVisitedUPP

Unregisters a previously registered application-defined function. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
void HRUnregisterWasURLVisitedUPP (
   HRReference hrRef
);
```

Parameters

hrRef

An `HRReference` value. You pass the same `HRReference` value that you passed to register the application-defined function.

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability

Available in CarbonLib 1.1 and later when running Mac OS 9 or later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`HTMLRendering.h`

HRUtilCreateFullCFURL

Obtains a full URL from a given set of relative URLs. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
OSStatus HRUtilCreateFullCFURL (
   CFStringRef rootString,
   CFStringRef linkString,
   CFURLRef *url
);
```

Parameters

rootString

A `CFStringRef` that refers to a `CFString` containing the root URL that the HTML Rendering Library will use to create the full URL. The root URL typically identifies an HTML source page.

linkString

A `CFStringRef` that refers to a `CFString` containing the link URL that the HTML Rendering Library will use to create the full URL. The link URL typically identifies a link being clicked on.

url

A `CFURLRef`. On return, this handle references a full URL created from the specified root URL and the specified link URL.

Return Value

A result code. See "HTML Rendering Library Result Codes" (page 2597).

Discussion

Use these API from a Carbon application instead of using `HRUtilCreateFullURL` (page 2578), `HRUtilGetFSSpecFromURL` (page 2580), and `HRUtilGetURLFromFSSpec` (page 2581). These APIs are same in behavior with their old counter parts. The only difference is that they take `CFURLRef`, and `FSRef` as parameters.

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability

Available in CarbonLib 1.3 and later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`HTMLRendering.h`

HRUtilCreateFullURL

Obtains a full URL from a given set of relative URLs. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
OSStatus HRUtilCreateFullURL (
   const char *rootURL,
   const char *linkURL,
   Handle fullURLH
);
```

Parameters

rootURL

A pointer to a C string containing the root URL that the HTML Rendering Library will use to create the full URL. The root URL typically identifies an HTML source page.

linkURL

A pointer to a C string containing the link URL that the HTML Rendering Library will use to create the full URL. The link URL typically identifies a link being clicked on.

fullURLH

A handle. On return, this handle references a full URL created from the specified root URL and the specified link URL. You must allocate this handle with the function `NewHandle` before calling `HRUtilCreateFullURL`. The HTML Rendering Library resizes the handle and terminates the C string with a `NULL` character.

Return Value

A result code. See "HTML Rendering Library Result Codes" (page 2597).

Discussion

This function allows you to obtain a full URL to use when a relative URL is inappropriate. This might be useful when you need to pass a URL to another application, for example.

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability

Available in CarbonLib 1.1 and later when running Mac OS 9 or later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`HTMLRendering.h`

HRUtilGetFSRefFromURL

Obtains a `FSRef` from a given set of relative URL's. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
OSStatus HRUtilGetFSRefFromURL (
   CFStringRef rootString,
   CFStringRef linkString,
   FSRef *destRef
);
```

Parameters

rootString

A reference to a `CFString` containing the root URL that the HTML Rendering Library will use to create the full URL. The root URL typically identifies an HTML source page.

linkString

A reference to a `CFString` containing the link URL that the HTML Rendering Library will use to create the full URL. The link URL typically identifies a link being clicked on.

destRef

A pointer to a `FSRef`. On return, this points to the `FSRef` of the full URL calculated from the specified root url and the specified link URL.

Return Value

A result code. See "HTML Rendering Library Result Codes" (page 2597).

Discussion

Use these API from a Carbon application instead of using `HRUtilCreateFullURL` (page 2578), `HRUtilGetFSSpecFromURL` (page 2580), and `HRUtilGetURLFromFSSpec` (page 2581). These APIs are same in behavior with their old counter parts. The only difference is that they take `CFURLRef`, and `FSRef` as parameters.

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very

quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability
Available in CarbonLib 1.3 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`HTMLRendering.h`

HRUtilGetFSSpecFromURL

Obtains a `FSSpec` from a given set of relative URL's. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
OSStatus HRUtilGetFSSpecFromURL (
   const char *rootURL,
   const char *linkURL,
   FSSpec *destSpec
);
```

Parameters

rootURL
A pointer to a C string containing the root URL that the HTML Rendering Library will use to create the full URL. The root URL typically identifies an HTML source page.

linkURL
A pointer to a C string containing the link URL that the HTML Rendering Library will use to create the full URL. The link URL typically identifies a link being clicked on.

destSpec
A pointer to a file system specification record (`FSSpec`). On return, this points to the `FSSPec` of the full URL calculated from the specified root url and the specified link URL.

Return Value
A result code. See "HTML Rendering Library Result Codes" (page 2597).

Discussion
This function allows you to obtain an `FSSpec` to use when a relative URL is inappropriate. This might be useful for Apple events, for example

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability
Available in CarbonLib 1.1 and later when running Mac OS 9 or later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
HTMLRendering.h

HRUtilGetURLFromFSRef

Obtains a full URL from a given FSRef. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
OSStatus HRUtilGetURLFromFSRef (
   const FSRef *fileRef,
   CFURLRef *url
);
```

Parameters

fileRef

A pointer to an existing FSRef.

url

A pointer to a CFURLRef. On return, the CFURL contains the URL of the given FSSpec.

Return Value
A result code. See "HTML Rendering Library Result Codes" (page 2597).

Discussion
Use these API from a Carbon application instead of using HRUtilCreateFullURL (page 2578), HRUtilGetFSSpecFromURL (page 2580), and HRUtilGetURLFromFSSpec (page 2581). These APIs are same in behavior with their old counter parts. The only difference is that they take CFURLRef, and FSRef as parameters.

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability
Available in CarbonLib 1.3 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
HTMLRendering.h

HRUtilGetURLFromFSSpec

Obtains a full URL from a given FSSpec. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
OSStatus HRUtilGetURLFromFSSpec (
   const FSSpec *fsspec,
   Handle urlHandle
);
```

Parameters

fsspec

A pointer to an existing file system specification record (`FSSpec`).

urlHandle

A handle to a C string. On return, this C string contains the URL of the given `FSSpec`. You must allocate this handle with the function `NewHandle` before calling `HRUtilGetURLFromFSSpec`. The HTML Rendering Library resizes the handle and terminates the C string with a `NULL` character.

Return Value

A result code. See "HTML Rendering Library Result Codes" (page 2597).

Discussion

This function allows you to obtain a URL from a given `FSSpec`. This might be useful when you have previously obtained an `FSSpec` and need to pass it to an application that requires URL data, for example.

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability

Available in CarbonLib 1.1 and later when running Mac OS 9 or later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`HTMLRendering.h`

InvokeHRNewCFURLUPP

(Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
OSStatus InvokeHRNewCFURLUPP (
   CFURLRef url,
   CFStringRef targetString,
   Boolean addToHistory,
   void *refCon,
   HRNewCFURLUPP userUPP
);
```

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very

quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability
Available in CarbonLib 1.3 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`HTMLRendering.h`

InvokeHRNewURLUPP

(Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
OSStatus InvokeHRNewURLUPP (
   const char *url,
   const char *targetFrame,
   Boolean addToHistory,
   void *refCon,
   HRNewURLUPP userUPP
);
```

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`HTMLRendering.h`

InvokeHRURLToFSRefUPP

(Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
OSStatus InvokeHRURLToFSRefUPP (
   CFStringRef rootString,
   CFStringRef linkString,
   FSRef *fref,
   URLSourceType urlSourceType,
   void *refCon,
   HRURLToFSRefUPP userUPP
);
```

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability
Available in CarbonLib 1.3 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`HTMLRendering.h`

InvokeHRURLToFSSpecUPP

(Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
OSStatus InvokeHRURLToFSSpecUPP (
   const char *rootURL,
   const char *linkURL,
   FSSpec *fsspec,
   URLSourceType urlSourceType,
   void *refCon,
   HRURLToFSSpecUPP userUPP
);
```

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`HTMLRendering.h`

InvokeHRWasCFURLVisitedUPP

(Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
Boolean InvokeHRWasCFURLVisitedUPP (
   CFURLRef url,
   void *refCon,
   HRWasCFURLVisitedUPP userUPP
);
```

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability
Available in CarbonLib 1.3 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
HTMLRendering.h

InvokeHRWasURLVisitedUPP

(Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
Boolean InvokeHRWasURLVisitedUPP (
   const char *url,
   void *refCon,
   HRWasURLVisitedUPP userUPP
);
```

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
HTMLRendering.h

NewHRNewCFURLUPP

(Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
HRNewCFURLUPP NewHRNewCFURLUPP (
   HRNewCFURLProcPtr userRoutine
);
```

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability
Available in CarbonLib 1.3 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
HTMLRendering.h

NewHRNewURLUPP

Obtains a UPP for an application-defined function that handles newly visited links. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
HRNewURLUPP NewHRNewURLUPP (
   HRNewURLProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your application-defined function that handles newly visited links. For more information, see `HRNewURLProcPtr` (page 2589).

Return Value

A Universal Procedure Pointer. You pass this pointer to the function `HRRegisterNewURLUPP` (page 2563). See the description of the `HRNewURLUPP` data type.

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability

Available in CarbonLib 1.1 and later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In

`HTMLRendering.h`

NewHRURLToFSRefUPP

(Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
HRURLToFSRefUPP NewHRURLToFSRefUPP (
   HRURLToFSRefProcPtr userRoutine
);
```

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability

Available in CarbonLib 1.3 and later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.4.

Declared In
HTMLRendering.h

NewHRURLToFSSpecUPP

Obtains a UPP for an application-defined function that intercepts URL's. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
HRURLToFSSpecUPP NewHRURLToFSSpecUPP (
   HRURLToFSSpecProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your application-defined function that intercepts URL's. For more information, see the function HRURLToFSSpecProcPtr (page 2591).

Return Value
A Universal Procedure Pointer. You pass this pointer to the function HRRegisterURLToFSSpecUPP (page 2564). See the description of the HRURLToFSSpecUPP data type.

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
HTMLRendering.h

NewHRWasCFURLVisitedUPP

(Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
HRWasCFURLVisitedUPP NewHRWasCFURLVisitedUPP (
   HRWasCFURLVisitedProcPtr userRoutine
);
```

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability
Available in CarbonLib 1.3 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`HTMLRendering.h`

NewHRWasURLVisitedUPP

Obtains a UPP for an application-defined function that handles previously visited links. (Deprecated in Mac OS X v10.4. Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
HRWasURLVisitedUPP NewHRWasURLVisitedUPP (
   HRWasURLVisitedProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your application-defined function that handles visited links. For more information, see the function `HRWasURLVisitedProcPtr` (page 2593).

Return Value
A Universal Procedure Pointer. You pass this pointer to the function `HRRegisterWasURLVisitedUPP` (page 2565). See the description of the `HRWasURLVisitedUPP` data type.

Special Considerations

Because the Web Kit takes a completely different approach to displaying HTML and web content from that used by the HTML Rendering Library, you cannot make a one-to-one substitution of Web Kit methods for HTML Rendering Library functions. However, the basic features of the Web Kit can be implemented very quickly, and Web Kit offers much greater capability than the HTML Rendering Library. Although the Web Kit is an Objective-C interface, you can call it from a Carbon application. See Accessing the Web Kit From Carbon Applications for details.

Availability
Available in CarbonLib 1.1 and later.
Available in Mac OS X 10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`HTMLRendering.h`

Callbacks

HRNewCFURLProcPtr

(**Deprecated.** Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
typedef OSStatus (*HRNewCFURLProcPtr)
(
    CFURLRef url,
    CFStringRef targetString,
    Boolean addToHistory,
    void * refCon
);
```

If you name your function `MyHRNewCFURLProc`, you would declare it like this:

```
OSStatus MyHRNewCFURLProc (
    CFURLRef url,
    CFStringRef targetString,
    Boolean addToHistory,
    void * refCon
);
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HTMLRendering.h`

HRNewURLProcPtr

An application-defined function that tracks newly visited links. (**Deprecated.** Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
typedef OSStatus (*HRNewURLProcPtr) (
    const char * url,
    const char * targetFrame,
    Boolean addToHistory,
    void * refCon
);
```

If you name your function `MyHRNewURLProc`, you would declare it like this:

```
OSStatus MyHRNewURLProc (
    const char * url,
    const char * targetFrame,
    Boolean addToHistory,
    void * refCon
);
```

Parameters

`url`
A pointer to a C string containing the URL of the link.

`targetFrame`
A pointer to a C string containing the name of the target frame.

`addToHistory`
The HTML Rendering Library passes `true` in this parameter to indicate that you should add this link to a link-tracking history. It is up to your application to do any link-tracking.

refCon

An arbitrary value set by your application. This value is passed by your application when you call the function `HRRegisterNewURLUPP` (page 2563) and passed back when the HTML Rendering Library calls your `MyNewURLProc` function. You may find this value useful for referring to an object instance or a structure, for example.

Return Value
A result code. See "HTML Rendering Library Result Codes" (page 2597).

Discussion
You may find this function useful for maintaining a history list, for example. The sequence of steps required to implement an application-defined function to track visited links is as follows:

1. Obtain a UPP for your application-defined function by calling the function `NewHRNewURLUPP` (page 2586).
2. Register your application-defined function by passing the UPP to the function `HRRegisterNewURLUPP` (page 2563).
3. When the HTML Rendering Library calls your application-defined function, take note of the URL being visited.

When you are done using your application-defined function:

1. Unregister your application-defined function by calling the function `HRUnregisterNewURLUPP` (page 2574).
2. Dispose of the UPP by calling the function `DisposeHRNewURLUPP` (page 2534).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HTMLRendering.h`

HRURLToFSRefProcPtr

(**Deprecated.** Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
typedef OSStatus (*HRURLToFSRefProcPtr)
(
    CFStringRef rootString,
    CFStringRef linkString,
    FSRef * fref,
    URLSourceType urlSourceType,
    void * refCon
);
```

If you name your function `MyHRURLToFSRefProc`, you would declare it like this:

```
OSStatus MyHRURLToFSRefProc (
    CFStringRef rootString,
    CFStringRef linkString,
    FSRef * fref,
    URLSourceType urlSourceType,
```

```
    void * refCon
);
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HTMLRendering.h`

HRURLToFSSpecProcPtr

Converts URL data to a file system specification. (**Deprecated.** Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
typedef OSStatus (*HRURLToFSSpecProcPtr)
(
    const char * rootURL,
    const char * linkURL,
    FSSpec * fsspec,
    URLSourceType urlSourceType,
    void * refCon
);
```

If you name your function `MyHRURLToFSSpecProc`, you would declare it like this:

```
OSStatus MyHRURLToFSSpecProc (
    const char * rootURL,
    const char * linkURL,
    FSSpec * fsspec,
    URLSourceType urlSourceType,
    void * refCon
);
```

Parameters

rootURL
A pointer to a C string containing the root URL of the file to be loaded.

linkURL
A pointer to a C string containing the link URL of the file to be loaded.

fsspec
A pointer to a file system specification record (`FSSpec`) that you use to specify the file to which the HTML Rendering Library redirects the user.

urlSourceType

The HTML Rendering Library passes one of the following constants to indicate the type of file being searched for:

- `kHRLookingForHTMLSource` indicates that the file is an HTML source document.
- `kHRLookingForImage` indicates that the file is an image.
- `kHRLookingForEmbedded` indicates that the file is an embedded object, such as a QuickTime movie.
- `kHRLookingForImageMap` indicates that the file is an HTML image map.
- `kHRLookingForFrame` indicates that the file is an HTML frameset.

refCon

An arbitrary value set by your application. This value is passed by your application when you call the function and passed back when the HTML Rendering Library calls your `MyHRURLToFSSpecProc` function. You may find this value useful for referring to an object instance or a structure, for example.

Return Value
A result code. See "HTML Rendering Library Result Codes" (page 2597).

Discussion
This function may be useful if you want to redirect certain URL's to application-specific files, for example. The sequence of steps required to implement an application-defined function to intercept URL's is as follows:

1. Obtain a UPP for your application-defined function by calling the function `NewHRURLToFSSpecUPP` (page 2587).
2. Register your application-defined function by passing the UPP to the function `HRRegisterURLToFSSpecUPP` (page 2564).
3. Respond when the HTML Rendering Library calls your application-defined function.

When you are done with your application-defined function:

1. Unregister your application-defined function by calling the function `HRUnregisterURLToFSSpecUPP` (page 2576).
2. Dispose of the UPP by calling the function `DisposeHRURLToFSSpecUPP` (page 2535).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HTMLRendering.h`

HRWasCFURLVisitedProcPtr

(**Deprecated.** Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
typedef Boolean (*HRWasCFURLVisitedProcPtr)
(
    CFURLRef url,
    void * refCon
);
```

If you name your function `MyHRWasCFURLVisitedProc`, you would declare it like this:

```
Boolean MyHRWasCFURLVisitedProc (
    CFURLRef url,
    void * refCon
);
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HTMLRendering.h`

HRWasURLVisitedProcPtr

Keeps track of whether a given URL has been previously visited. (**Deprecated.** Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
typedef Boolean (*HRWasURLVisitedProcPtr)
(
    const char * url,
    void * refCon
);
```

If you name your function `MyHRWasURLVisitedProc`, you would declare it like this:

```
Boolean MyHRWasURLVisitedProc (
    const char * url,
    void * refCon
);
```

Parameters

`url`

A pointer to a C string containing the URL of the link.

`refCon`

An arbitrary value set by your application. This value is passed by your application when you call the function and passed back when the HTML Rendering Library calls your `MyHRWasURLVisitedProc` function. You may find this value useful for referring to an object instance or a structure, for example.

Return Value
If the given URL was previously visited, your application-defined function should return `true`. If the given URL was not previously visited, your application-defined function should return `false`.

Discussion
The sequence of steps required to implement an application-defined function to handle previously visited links is as follows:

1. Obtain a UPP for your application-defined function by calling the function `NewHRWasURLVisitedUPP` (page 2588).
2. Register your application-defined function by passing the UPP to the function `HRRegisterWasURLVisitedUPP` (page 2565).
3. Respond when the HTML Rendering Library calls your application-defined function.

When you are done using your application-defined function:

1. Unregister your application-defined function by calling the function `HRUnregisterWasURLVisitedUPP` (page 2577).
2. Dispose of the UPP by calling the function `DisposeHRWasURLVisitedUPP` (page 2536).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HTMLRendering.h`

Data Types

HRNewCFURLUPP

(**Deprecated.** Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
typedef HRNewCFURLProcPtr HRNewCFURLUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HTMLRendering.h`

HRNewURLUPP

(**Deprecated.** Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
typedef HRNewURLProcPtr HRNewURLUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HTMLRendering.h`

HRReference

(**Deprecated.** Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
typedef struct OpaqueHRReference * HRReference;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HTMLRendering.h`

HRURLToFSRefUPP

(**Deprecated.** Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
typedef HRURLToFSRefProcPtr HRURLToFSRefUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HTMLRendering.h`

HRURLToFSSpecUPP

(**Deprecated.** Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
typedef HRURLToFSSpecProcPtr HRURLToFSSpecUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HTMLRendering.h`

HRWasCFURLVisitedUPP

(**Deprecated.** Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
typedef HRWasCFURLVisitedProcPtr HRWasCFURLVisitedUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HTMLRendering.h`

HRWasURLVisitedUPP

(**Deprecated.** Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
typedef HRWasURLVisitedProcPtr HRWasURLVisitedUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`HTMLRendering.h`

Constants

Scrollbar State

(**Deprecated.** Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
typedef SInt16 HRScrollbarState;
enum {
    eHRScrollbarOn = 0,
    eHRScrollbarOff = 1,
    eHRScrollbarAuto = 2
};
```

Constants

`eHRScrollbarOn`
Tells the HTML Rendering Library to draw scrollbars at all times. If the data does not fill the current view, the scroll bars are inactive.

Available in Mac OS X v10.0 and later.

Declared in `HTMLRendering.h`.

`eHRScrollbarOff`
Tells the HTML Rendering Library never to draw scrollbars. You may find this option useful if the HTML rendering area you specify does not extend to the edge of a window.

Available in Mac OS X v10.0 and later.

Declared in `HTMLRendering.h`.

`eHRScrollbarAuto`
Tells the HTML Rendering Library to draw scrollbars as needed. This is the default setting.

Available in Mac OS X v10.0 and later.

Declared in `HTMLRendering.h`.

Renderer HTML Type

(**Deprecated.** Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
enum {
    kHRRendererHTML32Type = 'ht32'
};
```

URL Source Type

(**Deprecated.** Use Web Kit instead; see *WebKit Objective-C Programming Guide*.Use Web Kit instead; see *WebKit Objective-C Programming Guide*.)

```
typedef UInt16 URLSourceType;
enum {
    kHRLookingForHTMLSource = 1,
    kHRLookingForImage = 2,
    kHRLookingForEmbedded = 3,
    kHRLookingForImageMap = 4,
    kHRLookingForFrame = 5
};
```

Constants

`kHRLookingForHTMLSource`
Indicates that the file is an HTML source document.
Available in Mac OS X v10.0 and later.
Declared in `HTMLRendering.h`.

`kHRLookingForImage`
Indicates that the file is an image.
Available in Mac OS X v10.0 and later.
Declared in `HTMLRendering.h`.

`kHRLookingForEmbedded`
Indicates that the file is an embedded object, such as a QuickTime movie.
Available in Mac OS X v10.0 and later.
Declared in `HTMLRendering.h`.

`kHRLookingForImageMap`
Indicates that the file is an HTML image map.
Available in Mac OS X v10.0 and later.
Declared in `HTMLRendering.h`.

`kHRLookingForFrame`
Indicates that the file is an HTML frameset.
Available in Mac OS X v10.0 and later.
Declared in `HTMLRendering.h`.

Result Codes

The most common result codes returned by HTML Rendering Library are listed in the table below. HTML Rendering Library may also return codes `noErr` (0), and `paramErr` (-50).

Result Code	Value	Description
`hrHTMLRenderingLibNotInstalledErr`	-5360	The HTML Rendering Library is not available. Available in Mac OS X v10.0 and later.
`hrMiscellaneousExceptionErr`	-5361	An unexpected exception occurred. Available in Mac OS X v10.0 and later.
`hrUnableToResizeHandleErr`	-5362	Unable to resize a handle to the required size. Available in Mac OS X v10.0 and later.

Multilingual Text Engine Reference

Framework: Carbon/Carbon.h

Declared in MacTextEditor.h

Overview

Multilingual Text Engine (MLTE) is an API that allows your application to provide Carbon-compliant Unicode text editing. This document is relevant for anyone who is writing an application that needs to display static Unicode text or provide Unicode-compliant text editing fields. You can also use MLTE if your application provides text editing support within a full-size window. For more information about basic text processing and using MLTE, see the document "Handling Unicode Text Editing with MLTE."

Carbon fully supports the Multilingual Text Engine, and Apple recommends that you adopt Multilingual Text Engine as a replacement for TextEdit.

Functions by Task

Displaying Static Text

`TXNDrawUnicodeTextBox` (page 2639)
Draws a Unicode string in the specified rectangle.

`TXNDrawCFStringTextBox` (page 2637)
Draws text from a Core Foundation string (`CFString`) in the specified rectangle.

Initializing and Terminating MLTE

`TXNInitTextension` (page 2662)
Initializes MLTE.

`TXNVersionInformation` (page 2698)
Gets the version number of MLTE and the set of features in this version.

Working With MLTE Objects

`TXNCreateObject` (page 2630)
Creates a new MLTE text object which is an opaque structure that handles text formatting at the document level.

`TXNGetAccessibilityHIObject` (page 2645)
Obtains an `HIObjectRef` representing the MLTE object for accessibility purposes.

`TXNDeleteObject` (page 2632)
Deletes a previously allocated text object.

`TXNDataSize` (page 2632)
Reports the amount of memory used to hold the text in a given text object.

`TXNAttachObjectToWindowRef` (page 2620)
Attaches a text object to a window.

`TXNGetWindowRef` (page 2660)
Returns a reference to the window to which the specified text object is attached.

Responding to Events

`TXNGetEventTarget` (page 2651)
Obtains the current event target for a TXNObject.

`TXNSetEventTarget` (page 2686)
Sets a Carbon Event target for MLTE Carbon Event handlers.

`TXNGetCommandEventSupport` (page 2647)
Obtains the command event support that is currently set for an MLTE object.

`TXNSetCommandEventSupport` (page 2681)
Enables and disables support for menu commands in MLTE.

`TXNAdjustCursor` (page 2619)
Obtains the current cursor position and draws the cursor in a form appropriate to the content over which it is placed.

`TXNClick` (page 2626)
Processes a mouse-down event in a window's content area.

`TXNEchoMode` (page 2640)
Determines whether a specified character is drawn instead of the glyph associated with the input character.

`TXNFocus` (page 2644)
Changes the focus of a text object.

`TXNForceUpdate` (page 2644)
Forces an update of the view rectangle and the scroll bars.

`TXNGetSleepTicks` (page 2658)
Reports the appropriate amount of time to allot to background processes, depending on the state of the window.

`TXNIdle` (page 2662)
Does idle time processing, such as flashing the cursor.

`TXNKeyDown` (page 2666)
Processes a key-down event.

`TXNUpdate` (page 2698)
Redraws everything in a frame in response to an update event.

`TXNScroll` (page 2679)
Scrolls the text within a view rectangle of the specified text object.

`TXNRegisterScrollInfoProc` (page 2675)
Installs or uninstalls a scrolling callback function on a text object.

Working With HITextView

`HITextViewCreate` (page 2610)
Creates an HITextView that is initially invisible.

`HITextViewGetTXNObject` (page 2611)
Obtains the text object associated with an HITextView.

`HITextViewCopyBackgroundColor` (page 2610)
Obtains the background color of the view.

`HITextViewSetBackgroundColor` (page 2612)
Sets the background color of the view.

Editing Data

`TXNDrawObject` (page 2638)
Draws a text object in the last window set by your application.

`TXNClear` (page 2624)
Deletes the current selection.

`TXNCopy` (page 2628)
Copies the current selection to the private MLTE scrap.

`TXNCut` (page 2631)
Deletes the current selection and copies it to the private MLTE scrap.

`TXNIsScrapPastable` (page 2665)
Tests whether the Clipboard contains data that is supported by MLTE.

`TXNPaste` (page 2671)
Pastes the contents of the private MLTE scrap into the text object.

`TXNGetData` (page 2649)
Copies a range of data.

`TXNGetDataEncoded` (page 2650)
Copies the text in a specified range, and if necessary, translates the text to match your application's preferred encoding.

`TXNSetData` (page 2683)
Replaces a range of data (text, graphics, and so forth).

`TXNCanRedoAction` (page 2622)
Indicates whether an action can be redone.

`TXNRedo` (page 2675)
Redoes the last command.

`TXNCanUndoAction` (page 2624)
Indicates whether an action can be undone.

`TXNUndo` (page 2697)
Undoes the last command.

Managing Fonts and Font Menus

`TXNCountRunsInRange` (page 2629)
Obtains a count of the style runs in a range of data.

`TXNGetIndexedRunInfoFromRange` (page 2654)
Gets information about a run in a range of data.

`TXNSetTypeAttributes` (page 2694)
Sets text attributes (such as size and style) for the current selection or the text defined by a range you specify.

`TXNGetContinuousTypeAttributes` (page 2648)
Checks to see if the attributes of the current selection are continuous.

`TXNDisposeFontMenuObject` (page 2633) Deprecated in Mac OS X v10.5
Disposes of a Font menu object.

`TXNDoFontMenuSelection` (page 2633) Deprecated in Mac OS X v10.5
Changes the font of the current selection.

`TXNGetFontMenuHandle` (page 2653) Deprecated in Mac OS X v10.5
Gets the Font menu handle that belongs to a Font menu object.

`TXNNewFontMenuObject` (page 2666) Deprecated in Mac OS X v10.5
Creates a new Font menu object.

`TXNPrepareFontMenu` (page 2672) Deprecated in Mac OS X v10.5
Prepares a Font menu for display.

`TXNGetFontDefaults` (page 2652) Deprecated in Mac OS X v10.4
Makes a copy of the font descriptions for a given text object.

`TXNSetFontDefaults` (page 2688) Deprecated in Mac OS X v10.4
Specifies the font descriptions for each script used in a text object.

Managing Layout and Formatting

`TXNRecalcTextLayout` (page 2675)
Recalculates the text layout based on new view and destination rectangles.

`TXNSetTXNObjectControls` (page 2693)
Sets formatting and privileges attributes (such as justification, line direction, tab values, and read-only status) that apply to the entire text object.

`TXNGetTXNObjectControls` (page 2659)
Gets the current formatting and privileges attributes (such as justification, line direction, tab values, and read-only status) for a text object.

`TXNGetLineCount` (page 2656)
Gets the total number of lines in a text object.

`TXNGetLineMetrics` (page 2656)
Gets information about line width and height for a specified line of data in a text object.

`TXNHIPointToOffset` (page 2661)
Gets the offset value that corresponds to a point in local coordinates.

`TXNOffsetToHIPoint` (page 2670)
Obtains the local coordinates of the point that corresponds to a specified offset of a text object.

Managing Selections

`TXNGetSelection` (page 2658)
Gets the absolute offsets of the current selection.

`TXNIsSelectionEmpty` (page 2665)
Determines whether the current selection is empty.

`TXNSelectAll` (page 2680)
Selects all data in the frame of a text object.

`TXNSetSelection` (page 2692)
Specifies the selection range or the position of the insertion point.

`TXNShowSelection` (page 2696)
Scrolls the current selection into view.

Controlling the Frame and Window

`TXNGetViewRect` (page 2660)
Gets the rectangle that describes the current view of the document.

`TXNGrowWindow` (page 2661)
Adjusts the size of a window in response to mouse-down events in the size region of the window.

`TXNResizeFrame` (page 2676)
Resizes the view and destination rectangles.

`TXNSetBackground` (page 2681)
Sets the background on which the text object's data is drawn.

`TXNSetFrameBounds` (page 2688)
Changes the boundaries of a text object's frame.

`TXNZoomWindow` (page 2700)
Increases the size of the data displayed in a window in response to a click in the zoom box.

`TXNSetHIRectBounds` (page 2689)
Sets the view rectangle and/or the destination rectangle.

`TXNGetHIRect` (page 2653)
Obtains the values for the current view, destination, or text rectangle.

`TXNSetScrollbarState` (page 2691)
Sets the state of the scroll bars so they are drawn correctly in response to activate events.

Searching

`TXNFind` (page 2641)
Finds a piece of text or a graphics, sound, or movie object.

Managing Files

`TXNFlattenObjectToCFDataRef` (page 2643)
Flattens a text object so it can be saved to disk or embedded with other data.

`TXNRevert` (page 2677)
Reverts to the last saved version of a document.

`TXNReadFromCFURL` (page 2673)
Reads data from a `CFURLRef` into a TXNObject.

`TXNCopyTypeIdentifiersForRange` (page 2628)
Obtains an array of universal type identifiers for a TXNObject.

`TXNWriteRangeToCFURL` (page 2699)
Writes a range of a text object to a file or to a special file bundle.

Printing

`TXNPageSetup` (page 2671)
Displays the Page Setup dialog for the current default printer and manages changes, such as reformatting the text, in response to page layout changes.

`TXNPrint` (page 2673)
Prints the document so it is formatted to fit the page size selected for the printer.

Supporting Drag and Drop

`TXNDragReceiver` (page 2634)
Handles dragged data in a text object for which a custom drag handler is already in place.

`TXNDragTracker` (page 2635)
Handles tracking a drag event in a text object for which a custom drag handler is already in place.

Keeping Track of User Actions

`TXNGetChangeCount` (page 2647)
Retrieves the number of times a document has been changed.

`TXNGetCountForActionType` (page 2649)
Gets the number of times a given type of action has occurred.

`TXNClearCountForActionType` (page 2626)
Sets the counter for the specified action type to zero.

`TXNBeginActionGroup` (page 2621)
Starts an action group.

`TXNEndActionGroup` (page 2641)
Ends an action group.

`TXNSetActionNameMapper` (page 2680)
Sets a callback that MLTE uses to obtain the localized string representing an action or an action group.

Managing Spell Checking As You Type

`TXNGetSpellCheckAsYouType` (page 2659)
Determines whether the "Spell Check as You Type" feature is enabled.

`TXNSetSpellCheckAsYouType` (page 2693)
Enables and disables the "Spell Check as You Type" feature.

Working with the Contextual Menu

`TXNSetContextualMenuSetup` (page 2682)
Provides a callback function that is called before MLTE displays its contextual menu.

Working With UPP Pointers for MLTE Callback Functions

`NewTXNActionNameMapperUPP` (page 2616)
Creates a new universal procedure pointer (UPP) to an action name mapper callback function.

`InvokeTXNActionNameMapperUPP` (page 2613)
Calls your action name mapper callback function.

`DisposeTXNActionNameMapperUPP` (page 2608)
Disposes of the universal procedure pointer (UPP) to your action name mapper callback function.

`NewTXNContextualMenuSetupUPP` (page 2616)
Creates a new universal procedure pointer (UPP) to a contextual menu setup callback function.

`InvokeTXNContextualMenuSetupUPP` (page 2614)
Calls your contextual menu setup callback function.

`DisposeTXNContextualMenuSetupUPP` (page 2608)
Disposes of the universal procedure pointer (UPP) to your contextual menu setup callback function.

`NewTXNFindUPP` (page 2617)
Creates a new universal procedure pointer (UPP) to a find callback function that uses your criteria for matching.

`InvokeTXNFindUPP` (page 2614)
Calls your find callback function.

`DisposeTXNFindUPP` (page 2609)
Disposes of the universal procedure pointer (UPP) to your find callback function.

`NewTXNScrollInfoUPP` (page 2617)
Creates a new universal procedure pointer (UPP) to a scrolling callback function.

`InvokeTXNScrollInfoUPP` (page 2615)
Calls your scrolling callback function.

`DisposeTXNScrollInfoUPP` (page 2609)
Disposes of the universal procedure pointer (UPP) to your scrolling callback function.

Not Recommended

This section lists functions that are not recommended and you should no longer use. The Carbon Porting Notes for each function provide information on what you should do in place of using the function.

`TXNActivate` (page 2618) Deprecated in Mac OS X v10.3
Sets the state of the scroll bars so they are drawn correctly in response to activate events. (Deprecated. Use `TXNSetScrollbarState` (page 2691) instead.)

`TXNAttachObjectToWindow` (page 2620) Deprecated in Mac OS X v10.3
Attaches a text object to a window. (Deprecated. Use `TXNAttachObjectToWindowRef` (page 2620) instead.)

`TXNConvertFromPublicScrap` (page 2627) Deprecated in Mac OS X v10.3
Converts the Clipboard content to the private MLTE scrap. (Deprecated. This function isn't needed in Mac OS X.)

`TXNConvertToPublicScrap` (page 2627) Deprecated in Mac OS X v10.3
Converts the private MLTE scrap content to the Clipboard. (Deprecated. This function isn't needed in Mac OS X.)

`TXNDraw` (page 2636) Deprecated in Mac OS X v10.3
Redraws the text area, including any scroll bars associated with the text frame. (Deprecated. Use the `TXNDrawObject` (page 2638).)

`TXNGetRectBounds` (page 2657) Deprecated in Mac OS X v10.3
Obtains the values for the current view, destination, and text rectangles. (Deprecated. Use `TXNGetHIRect` (page 2653) instead.)

`TXNIsObjectAttachedToSpecificWindow` (page 2663) Deprecated in Mac OS X v10.3
Determines whether a text object is attached to a specified window. (Deprecated. Use `TXNGetWindowRef` (page 2660) instead.)

`TXNIsObjectAttachedToWindow` (page 2664) Deprecated in Mac OS X v10.3
Checks to see if a text object is attached to a window. (Deprecated. Use `TXNGetWindowRef` (page 2660) instead.)

`TXNNewObject` (page 2667) Deprecated in Mac OS X v10.3
Creates a new MLTE text object which is an opaque structure that handles text formatting at the document level. (Deprecated. Use `TXNCreateObject` (page 2630) instead.)

`TXNOffsetToPoint` (page 2670) Deprecated in Mac OS X v10.3
Gets the local coordinates of the point that corresponds to a specified offset of a text object. (Deprecated. Use `TXNOffsetToHIPoint` (page 2670) instead.)

`TXNPointToOffset` (page 2672) Deprecated in Mac OS X v10.3
Gets the offset value that corresponds to a point in local coordinates. (Deprecated. Use `TXNHIPointToOffset` (page 2661) instead.)

`TXNSetDataFromFile` (page 2685) Deprecated in Mac OS X v10.3
Replaces a range of data with the contents of a file. (Deprecated. Use `TXNSetDataFromCFURLRef` (page 2684) instead.)

`TXNSetRectBounds` (page 2690) Deprecated in Mac OS X v10.3
Set the view rectangle and/or the destination rectangle. (Deprecated. Use `TXNSetHIRectBounds` (page 2689) instead.)

`TXNTerminateTextension` (page 2696) Deprecated in Mac OS X v10.3
Closes the MLTE library. (Deprecated. This function is no longer needed.)

`TXNSetViewRect` (page 2695) Deprecated in Mac OS X v10.2
Sets the rectangle that describes the current view into the document; changes the amount of text that is viewable. (Deprecated. Use `TXNSetFrameBounds` (page 2688) or `TXNSetRectBounds` (page 2690) instead.)

`DisposeTXNActionKeyMapperUPP` (page 2608) Deprecated in Mac OS X v10.4
Disposes of the universal procedure pointer (UPP) to your action key mapping callback function. (Deprecated. Use `TXNActionNameMapperProcPtr` instead.)

`InvokeTXNActionKeyMapperUPP` (page 2613) Deprecated in Mac OS X v10.4
Calls your action key mapping callback function. (Deprecated. Use `TXNActionNameMapperProcPtr` instead.)

`NewTXNActionKeyMapperUPP` (page 2616) Deprecated in Mac OS X v10.4
Creates a new universal procedure pointer (UPP) to a callback function that uses your criteria for mapping actions. (Deprecated. Use `TXNActionNameMapperProcPtr` instead.)

`TXNCanRedo` (page 2622) Deprecated in Mac OS X v10.4
Returns whether the most recently undone action is redoable and indicates the type of action that can be redone. (Deprecated. Use `TXNCanRedoAction` (page 2622) instead.)

`TXNCanUndo` (page 2623) Deprecated in Mac OS X v10.4
Returns whether the most recent action is undoable and provides a value that indicates the type of action than can be undone. (Deprecated. Use `TXNCanUndoAction` (page 2624) instead.)

`TXNClearActionChangeCount` (page 2625) Deprecated in Mac OS X v10.4
Resets the specified action counters to zero. (Deprecated. Use `TXNClearCountForActionType` (page 2626) instead.)

`TXNGetActionChangeCount` (page 2646) Deprecated in Mac OS X v10.4
Retrieves the number of times the specified action or actions have occurred since the count was initialized or cleared. (Deprecated. Use `TXNGetCountForActionType` (page 2649) instead.)

`TXNSave` (page 2677) Deprecated in Mac OS X v10.4
Saves the contents of the document as the file type you specify. (Deprecated. Use `TXNWriteRangeToCFURL` (page 2699) instead.)

`TXNSetDataFromCFURLRef` (page 2684) Deprecated in Mac OS X v10.4
Replaces a range of data with the contents of a file. (Deprecated. Use `TXNReadFromCFURL` (page 2673) instead.)

Unsupported Functions

This section lists functions that are not supported and cannot be called in Mac OS X.

`TXNTSMCheck` (page 2697)
Checks to see if the Text Services Manager (TSM) is active.

Functions

DisposeTXNActionKeyMapperUPP

Disposes of the universal procedure pointer (UPP) to your action key mapping callback function. (Deprecated in Mac OS X v10.4. Use `TXNActionNameMapperProcPtr` instead.)

Not recommended.

```
void DisposeTXNActionKeyMapperUPP (
   TXNActionKeyMapperUPP userUPP
);
```

Discussion
See the callback `TXNActionKeyMapperProcPtr` (page 2700) for more information.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`MacTextEditor.h`

DisposeTXNActionNameMapperUPP

Disposes of the universal procedure pointer (UPP) to your action name mapper callback function.

```
void DisposeTXNActionNameMapperUPP (
   TXNActionNameMapperUPP userUPP
);
```

Parameters

userUPP
The `TXNActionNameMapperUPP` that is to be disposed of.

Discussion
See the callback `TXNActionNameMapperProcPtr` (page 2702) for more information.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`MacTextEditor.h`

DisposeTXNContextualMenuSetupUPP

Disposes of the universal procedure pointer (UPP) to your contextual menu setup callback function.

```
void DisposeTXNContextualMenuSetupUPP (
   TXNContextualMenuSetupUPP userUPP
);
```

Parameters

userUPP

The `TXNContextualMenuSetupUPP` that is to be disposed of.

Discussion
See the callback `TXNContextualMenuSetupProcPtr` (page 2702) for more information.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`MacTextEditor.h`

DisposeTXNFindUPP

Disposes of the universal procedure pointer (UPP) to your find callback function.

```
void DisposeTXNFindUPP (
   TXNFindUPP userUPP
);
```

Discussion
See the callback `TXNFindProcPtr` (page 2703) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacTextEditor.h`

DisposeTXNScrollInfoUPP

Disposes of the universal procedure pointer (UPP) to your scrolling callback function.

```
void DisposeTXNScrollInfoUPP (
   TXNScrollInfoUPP userUPP
);
```

Discussion
See the callback `TXNScrollInfoProcPtr` (page 2705) for more information.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`MacTextEditor.h`

HITextViewCopyBackgroundColor

Obtains the background color of the view.

```
OSStatus HITextViewCopyBackgroundColor (
   HIViewRef inTextView,
   CGColorRef *outColor
);
```

Parameters

inTextView

The HITextView associated with the text object whose background color you want to copy.

outColor

A `CGColorRef` representing the color or pattern that is used for drawing the background of the text view. If the returned `CGColorRef` is not `NULL`, it is retained on return. You are responsible for releasing this `CGColorRef` when you are no longer referencing it. If the returned value is `NULL`, the background is not drawn.

Return Value

A result code. See "MLTE Result Codes" (page 2773).

Availability

Available in Mac OS X v10.4 and later.

Not available to 64-bit applications.

Declared In

`HITextViews.h`

HITextViewCreate

Creates an HITextView that is initially invisible.

```
OSStatus HITextViewCreate (
   const HIRect *inBoundsRect,
   OptionBits inOptions,
   TXNFrameOptions inTXNFrameOptions,
   HIViewRef *outTextView
);
```

Parameters

inBoundsRect

The bounding box of the view. Pass `NULL`, if you want to initialize the bounds of the view to `0`.

inOptions

Reserved for future use; you must pass `0`.

inTXNFrameOptions

The frame options you want to set for the text view.

outTextView

On output, points to the newly-created text view.

Return Value

A result code. See "MLTE Result Codes" (page 2773).

Discussion
An HITextView is an MLTE text view that can be embedded in the HIView hierarchy. The view can be embedded in an HIScrollView if you want scroll bars and can also be used in a composited window. For more information on HIView, see the document *Introducing HIView*, available from the Apple Developer Documentation website.

When you call the function `HITextViewCreate` to create a text view, an MLTE text object (`TXNObject`) is allocated and attached to the text view. You can extract the text object by calling the function `HITextViewGetTXNObject`. You can supply the extracted text object as a parameter to many of the MLTE functions that take a text object as a parameter. However, not all MLTE functions that take a text object can operate on an MLTE object that comes from an HITextView. In general, you cannot use MLTE functions that may alter the geometry of the object or explicitly invoke drawing. If you do, the function returns the result code `kTXNDisabledFunctionalityErr`.

The following MLTE functions return an error if you pass a text object that comes from an HITextView:

- `TXNAttachObjectToWindowRef`
- `TXNGetWindowRef`
- `TXNDrawObject`
- `TXNSetScrollbarState`
- `TXNGrowWindow`
- `TXNZoomWindow`
- `TXNResizeFrame`
- `TXNSetFrameBounds`
- `TXNSetViewRect`
- `TXNDraw`
- `TXNFocus`
- `TXNUpdate`
- `TXNForceUpdate`
- `TXNPageSetup`
- `TXNPrint`
- `TXNIdle`

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In
`HITextViews.h`

HITextViewGetTXNObject

Obtains the text object associated with an HITextView.

```
TXNObject HITextViewGetTXNObject (
   HIViewRef inTextView
);
```

Parameters

inTextView

The HITextView associated with the text object you want to retrieve.

Return Value
Returns the text object associated with the given view.

Discussion
You can supply the extracted text object as a parameter to many of the MLTE functions that take a text object as a parameter. However, not all MLTE functions that take a text object can operate on an MLTE object that comes from an HITextView. In general, you cannot use MLTE functions that may alter the geometry of the object or explicitly invoke drawing. If you do, the function returns the result code `kTXNDisabledFunctionalityErr`. See `HITextViewCreate` (page 2610) for a list of the functions that return an error if you pass a text object that comes from an HITextView.

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Related Sample Code
QTMetaData

Declared In
`HITextViews.h`

HITextViewSetBackgroundColor

Sets the background color of the view.

```
OSStatus HITextViewSetBackgroundColor (
   HIViewRef inTextView,
   CGColorRef inColor
);
```

Parameters

inTextView

The HITextView whose background color is to be set.

inColor

A `CGColorRef` representing the color or pattern that is to fill the background of the text view. The `CGColorRef` is retained by this function. If the text view already contains a background color, it is released prior to the new color being retained. If `inColor` is `NULL`, the background of the text view will not draw.

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Discussion
This function allows you to provide alpha.

Availability
Available in Mac OS X v10.4 and later.

Not available to 64-bit applications.

Declared In
HITextViews.h

InvokeTXNActionKeyMapperUPP

Calls your action key mapping callback function. (Deprecated in Mac OS X v10.4. Use `TXNActionNameMapperProcPtr` instead.)

Not recommended.

```
CFStringRef InvokeTXNActionKeyMapperUPP (
   TXNActionKey actionKey,
   UInt32 commandID,
   TXNActionKeyMapperUPP userUPP
);
```

Return Value
See the Base Services documentation for a description of the `CFStringRef` data type.

Discussion
See the callback `TXNActionKeyMapperProcPtr` (page 2700) for more information.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
MacTextEditor.h

InvokeTXNActionNameMapperUPP

Calls your action name mapper callback function.

```
CFStringRef InvokeTXNActionNameMapperUPP (
   CFStringRef actionName,
   UInt32 commandID,
   void *inUserData,
   TXNActionNameMapperUPP userUPP
);
```

Parameters

actionName
The action name.

commandID
The command ID of the menu item that is to be mapped.

iUserData
A pointer to user-defined data that will be passed to your action name mapper callback.

userUPP
The callback function that is to be called. For more information, see `NewTXNActionNameMapperUPP` (page 2616).

Return Value
A `CFStringRef`.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`MacTextEditor.h`

InvokeTXNContextualMenuSetupUPP

Calls your contextual menu setup callback function.

```
void InvokeTXNContextualMenuSetupUPP (
   MenuRef iContextualMenu,
   TXNObject object,
   void *inUserData,
   TXNContextualMenuSetupUPP userUPP
);
```

Parameters

iContextualMenu
The contextual menu.

object
The TXNObject for which the contextual menu is to be displayed.

iUserData
A pointer to user-defined data that will be passed to your contextual menu setup callback.

userUPP
The callback function that is to be called. For more information, see `NewTXNContextualMenuSetupUPP` (page 2616).

Availability
Available in Mac OS X v10.4 and later.

Declared In
`MacTextEditor.h`

InvokeTXNFindUPP

Calls your find callback function.

```
OSStatus InvokeTXNFindUPP (
   const TXNMatchTextRecord *matchData,
   TXNDataType iDataType,
   TXNMatchOptions iMatchOptions,
   const void *iSearchTextPtr,
   TextEncoding encoding,
   TXNOffset absStartOffset,
   ByteCount searchTextLength,
   TXNOffset *oStartMatch,
   TXNOffset *oEndMatch,
   Boolean *ofound,
   URefCon refCon,
   TXNFindUPP userUPP
);
```

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Discussion
You can call an invoke function rather than calling your routine directly if you want to support code portability across compiler targets. See the callback `TXNFindProcPtr` (page 2703) for parameter descriptions and other information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacTextEditor.h`

InvokeTXNScrollInfoUPP

Calls your scrolling callback function.

```
void InvokeTXNScrollInfoUPP (
   SInt32 iValue,
   SInt32 iMaximumValue,
   TXNScrollBarOrientation iScrollBarOrientation,
   SRefCon iRefCon,
   TXNScrollInfoUPP userUPP
);
```

Discussion
You can call an invoke function rather than calling your routine directly if you want to support code portability across compiler targets. See the callback `TXNScrollInfoProcPtr` (page 2705) for parameter descriptions and other information.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`MacTextEditor.h`

NewTXNActionKeyMapperUPP

Creates a new universal procedure pointer (UPP) to a callback function that uses your criteria for mapping actions. (Deprecated in Mac OS X v10.4. Use `TXNActionNameMapperProcPtr` instead.)

Not recommended.

```
TXNActionKeyMapperUPP NewTXNActionKeyMapperUPP (
   TXNActionKeyMapperProcPtr userRoutine
);
```

Return Value
A universal procedure pointer.

Discussion
See the callback `TXNActionKeyMapperProcPtr` (page 2700) for more information.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`MacTextEditor.h`

NewTXNActionNameMapperUPP

Creates a new universal procedure pointer (UPP) to an action name mapper callback function.

```
TXNActionNameMapperUPP NewTXNActionNameMapperUPP (
   TXNActionNameMapperProcPtr userRoutine
);
```

Parameters

userRoutine
The action name mapper callback function for which a UPP is to be created.

Return Value
A universal procedure pointer.

Discussion
See the callback `TXNActionNameMapperProcPtr` (page 2702) for more information.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`MacTextEditor.h`

NewTXNContextualMenuSetupUPP

Creates a new universal procedure pointer (UPP) to a contextual menu setup callback function.

```
TXNContextualMenuSetupUPP NewTXNContextualMenuSetupUPP (
   TXNContextualMenuSetupProcPtr userRoutine
);
```

Parameters

userRoutine

The contextual menu setup callback function for which a UPP is to be created.

Return Value
A universal procedure pointer.

Discussion
For more information, see the callback `TXNContextualMenuSetupProcPtr` (page 2702).

Availability
Available in Mac OS X v10.4 and later.

Declared In
`MacTextEditor.h`

NewTXNFindUPP

Creates a new universal procedure pointer (UPP) to a find callback function that uses your criteria for matching.

```
TXNFindUPP NewTXNFindUPP (
   TXNFindProcPtr userRoutine
);
```

Return Value
A universal procedure pointer.

Discussion
See the callback `TXNFindProcPtr` (page 2703) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacTextEditor.h`

NewTXNScrollInfoUPP

Creates a new universal procedure pointer (UPP) to a scrolling callback function.

```
TXNScrollInfoUPP NewTXNScrollInfoUPP (
   TXNScrollInfoProcPtr userRoutine
);
```

Return Value
A universal procedure pointer.

Discussion
See the callback `TXNScrollInfoProcPtr` (page 2705) for more information.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`MacTextEditor.h`

TXNActivate

Sets the state of the scroll bars so they are drawn correctly in response to activate events. (Deprecated in Mac OS X v10.3. Use `TXNSetScrollbarState` (page 2691) instead.)

Not recommended.

```
OSStatus TXNActivate (
   TXNObject iTXNObject,
   TXNFrameID iTXNFrameID,
   TXNScrollBarState iActiveState
);
```

Parameters

iTXNObject
The text object that identifies the document to be activated.

iTXNFrameID
The frame ID of the document that is to be activated. You obtain a frame ID from `TXNNewObject` (page 2667) when you create a text object.

iActiveState
A value that indicates the state of the scroll bars. See Scroll Bar States (page 2760) for a description of possible values. If you pass the `kScrollBarsAlwaysActive` constant, the scroll bars are always active, whether or not the frame's text area currently has keyboard focus. Passing `kScrollBarsAlwaysActive` can be useful for a window such as a dialog box that may contain multiple text areas, each of which may have a scrollable frame. If you pass `kScrollBarsSyncWithFocus`, MLTE synchronizes the activity state of the scroll bars with the focus state of the frame. Therefore, only when the frame has keyboard focus does it have active scroll bars. A value of `kScrollBarsSyncWithFocus` is the default and is typically recommended if you have only one frame per window.

Return Value
A result code. See "MLTE Result Codes" (page 2773). `TXNActivate` returns a parameter error if you pass an invalid text object or frame ID.

Discussion
You typically call `TXNActivate` in response to an activate event. If the text object was previously inactive, `TXNActivate` removes any visual indication of its prior inactive state (such as a dimmed or framed selection area or inactive scroll bars). Before you call the `TXNActivate` function, you should make sure that the window belongs to your application.

The `TXNActivate` function does not change the keyboard focus. This means your application can have a text area that is not focused, but in which the scroll bars are active. This lets application users scroll the inactive text without changing the focus from another text area.

If you want to display a text area that has both keyboard focus and active scroll bars, you must call the `TXNFocus` (page 2644) function immediately before you call the `TXNActivate` function. Note that MLTE does not retain information about keyboard focus. So if, for example, you set the keyboard focus on a text area and the window containing the text area becomes deactivated, you must call the `TXNFocus` (page 2644) function when the window becomes activated again.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.3.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNAdjustCursor

Obtains the current cursor position and draws the cursor in a form appropriate to the content over which it is placed.

```
void TXNAdjustCursor (
   TXNObject iTXNObject,
   RgnHandle ioCursorRgn
);
```

Parameters

iTXNObject
The text object that identifies the text area for which MLTE should adjust the cursor.

ioCursorRgn
A handle to a region created by your application. Pass `NULL` if you do not want `TXNAdjustCursor` to provide information about the cursor position to your application. If you do want to obtain the cursor's current position, pass a valid region handle in this parameter. If you pass a valid region handle and the cursor is over the text area or its scroll bars, on return `TXNAdjustCursor` sets the region to a 2-pixel by 2-pixel square, centered on the cursor's hot spot. If the cursor is not over the text area or its scroll bars, or if you have passed `NULL`, `TXNAdjustCursor` does not adjust the input value.

Discussion
If the cursor is over a text area, `TXNAdjustCursor` sets the cursor to an I-beam. If the cursor is over graphics, a sound file, a movie, a scroll bar, or outside of a window, `TXNAdjustCursor` sets the cursor to an arrow. Before you call the `TXNAdjustCursor` function, you should make sure that the window belongs to your application.

You can pass the region handle returned by the `TXNAdjustCursor` function in the `ioCursorRgn` parameter to the `WaitNextEvent` function; this ensures that you receive mouse-moved events if the cursor moves outside that region. If you then receive a mouse-moved event, you can call `TXNAdjustCursor` again to ensure that the cursor type is appropriate to its new position. Alternately, to ensure that the cursor is adjusted correctly, you can simply call `TXNAdjustCursor` with every event that your application receives.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNAttachObjectToWindow

Attaches a text object to a window. (Deprecated in Mac OS X v10.3. Use `TXNAttachObjectToWindowRef` (page 2620) instead.)

Not recommended.

```
OSStatus TXNAttachObjectToWindow (
   TXNObject iTXNObject,
   GWorldPtr iWindow,
   Boolean iIsActualWindow
);
```

Parameters

`iTXNObject`

The text object with which you want to associate the window.

`iWindow`

A pointer to the graphics port to which the object should be attached. The graphics port may be a window (`WindowRef`) or a generic graphics port (`CGrafPtr`, `GWorldPtr`). If it is a window, note that you must typecast the window reference to a `GWorldPtr` data type.

`iIsActualWindow`

A `Boolean` value. Pass `true` if the `iWindow` parameter you passed refers to a Window Manager window (`WindowRef`), not a generic graphics port. Pass `false` if the `iWindow` parameter you passed does not refer to a window. If you pass `false`, MLTE never calls window-specific functions such as `InvalRect` or `BeginUpdate` for this text object, and it is your program's responsibility to handle any window-related functionality.

Return Value

A result code. See "MLTE Result Codes" (page 2773). `TXNAttachObjectToWindow` returns `paramErr` if the text object that you pass is invalid.

Discussion

You may create a text object without an associated window pointer by passing `NULL` in the `iWindow` parameter of the `TXNNewObject` (page 2667) function. However, if you do so, you must call the `TXNAttachObjectToWindow` function to associate a window with that object before you call any other MLTE function.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.3.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNAttachObjectToWindowRef

Attaches a text object to a window.

```
OSStatus TXNAttachObjectToWindowRef (
   TXNObject iTXNObject,
   WindowRef iWindowRef
);
```

Parameters

iTXNObject

The text object with which you want to associate the window. You can call the function `TXNCreateObject` to allocate a text object.

iWindowRef

A `WindowRef` that specifies the window to which you want to attach the text object.

Return Value
A result code. See "MLTE Result Codes" (page 2773). Returns `paramErr` if the text object that you pass is invalid.

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNBeginActionGroup

Starts an action group.

```
OSStatus TXNBeginActionGroup (
   TXNObject iTXNObject,
   CFStringRef iActionGroupName
);
```

Parameters

iTXNObject

The text object for which an action group is to be started.

iActionGroupName

A client-supplied string that is to be used to describe the action group.

Return Value
A result code. See "MLTE Result Codes" (page 2773). The error `kTXNOperationNotAllowedErr` is returned if an undo action group has already been started but has not yet terminated.

Discussion
Every supported edit action after `TXNBeginActionGroup` is called is added to the group until`TXNCanUndoAction` (page 2624) is called. When MLTE receives an undo or redo command, it treats all actions added to the group as a single operation to undo or redo. Nesting of groups is not allowed. Calling `TXNBeginActionGroup` twice without calling `TXNCanUndoAction` in between results in an error. If an action group is active, `TXNCanUndoAction` (page 2624) and `TXNCanRedoAction` (page 2622) return `false`.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNCanRedo

Returns whether the most recently undone action is redoable and indicates the type of action that can be redone. (Deprecated in Mac OS X v10.4. Use `TXNCanRedoAction` (page 2622) instead.)

Not recommended.

```
Boolean TXNCanRedo (
   TXNObject iTXNObject,
   TXNActionKey *oTXNActionKey
);
```

Parameters

iTXNObject

The text object for the document you want to examine.

oTXNActionKey

A pointer to a value of type `TXNActionKey`. On return, this value specifies the action that can be redone. See Action Constants (page 2719) for a description of possible values. You can use this information to customize the Redo menu item for the specific action to be redone. For example, if the value obtained by `TXNCanRedo` is `kTXNTypingAction`, you can map that value to a string that reads "Redo Typing" on a system localized for U.S. English. MLTE does not perform the mapping your program is responsible for mapping the key to the appropriate localized string you want displayed to the user. Pass `NULL` if you do not want to obtain this information.

Return Value

A `Boolean` value. If `true`, the last command is redoable; otherwise the last command cannot be redone.

Discussion

You can call the `TXNCanRedo` function to determine whether the Redo item in the Edit menu should be enabled.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNCanRedoAction

Indicates whether an action can be redone.

```
Boolean TXNCanRedoAction (
   TXNObject iTXNObject,
   CFStringRef *oActionName
);
```

Parameters

iTXNObject

The text object having an action that is to be queried.

oActionName

On input, a pointer a `CFStringRef` that, on return, contains the name of the action that can be redone, if there is one. The returned string is either a string defined by MLTE or the string that you passed to `TXNBeginActionGroup` (page 2621) to create a new action group. You are responsible for retaining and releasing the string. Pass `NULL` if you don't want to receive the name of the action.

Return Value
A Boolean whose value is `true` if the last action can be redone; you should enable the Redo item in the Edit menu, if there is one. If this function returns `false`, the last action cannot be redone and you should not enable the Redo item in the Edit menu.

Discussion
This function tells the client whether the current item on the undo stack is redoable and is usually used to determine whether the Redo item in the Edit menu should be enabled. This function optionally obtains the action name that should be used in the Redo item. When the current undo item is an action group, the string used to name the group is returned. For information on action groups, see `TXNBeginActionGroup` (page 2621) and `TXNEndActionGroup` (page 2641).

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNCanUndo

Returns whether the most recent action is undoable and provides a value that indicates the type of action than can be undone. (Deprecated in Mac OS X v10.4. Use `TXNCanUndoAction` (page 2624) instead.)

Not recommended.

```
Boolean TXNCanUndo (
   TXNObject iTXNObject,
   TXNActionKey *oTXNActionKey
);
```

Parameters

iTXNObject

The text object for the document you want to examine.

oTXNActionKey

A pointer to a value of type `TXNActionKey`. On return, this value identifies the action that can be undone. See Action Constants (page 2719) for a description of possible values. You can use this information to customize the Undo menu item for the specific action to be undone. For example, if the value obtained by `TXNCanUndo` is `kTXNTypingAction`, you can map that value to a string that reads "Undo Typing" on a system localized for U.S. English. MLTE does not perform such a mapping your program is responsible for mapping the key to the appropriate localized string you want displayed to the user. Pass `NULL` if you do not wish to obtain this information.

Return Value
A `Boolean` value. If `true`, the last command is undoable; otherwise the last command cannot be undone.

Discussion
You can call `TXNCanUndo` to determine whether the Undo item in the Edit menu should be enabled.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNCanUndoAction

Indicates whether an action can be undone.

```
Boolean TXNCanUndoAction (
   TXNObject iTXNObject,
   CFStringRef *oActionName
);
```

Parameters

iTXNObject

The text object having an action that is to be queried.

oActionName

On input, a pointer a `CFStringRef` that, on return contains the name of the action that can be undone, if there is one. The returned string is either a string defined by MLTE or the string that you passed to `TXNBeginActionGroup` (page 2621) to create a new action group. You are responsible for retaining and releasing the string. Pass `NULL` if you don't want to receive the name of the action.

Return Value
A Boolean whose value is `true` if the last action can be undone; you should enable the Undo item in the Edit menu, if there is one. If this function returns `false`, the last action cannot be undone and you should not enable the Undo item in the Edit menu.

Discussion
This function is usually used to determine whether the Undo item in the Edit menu should be enabled and to obtain the action name that should be used in that item. When the last action is an action group, the string used to name the group is returned.

If you have asked MLTE to handle updating for the Redo and Undo commands in the Edit menu, you should call `TXNSetActionNameMapper` (page 2680) after calling `TXNCanUndoAction` so that MLTE can call back to get the correct strings for those items.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNClear

Deletes the current selection.

```
OSStatus TXNClear (
   TXNObject iTXNObject
);
```

Parameters

iTXNObject

The text object for the current text area.

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Discussion
To remove a selected object from a text area, a user can either press the Delete key or choose Clear from the Edit menu. Before you call the `TXNClear` function, you can use the `TXNIsSelectionEmpty` (page 2665) function to determine whether any text is selected. Unlike the function `TXNCut` (page 2631), the `TXNClear` function does not add the deleted selection to the private MLTE scrap.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNClearActionChangeCount

Resets the specified action counters to zero. (Deprecated in Mac OS X v10.4. Use `TXNClearCountForActionType` (page 2626) instead.)

Not recommended.

```
OSStatus TXNClearActionChangeCount (
   TXNObject iTXNObject,
   TXNCountOptions iOptions
);
```

Parameters

iTXNObject

The text object whose action counter you want to reset.

iOptions

The `TXNCountOptions` to use when resetting the count. See Action Count Masks (page 2721) for information on the options you can supply.

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Discussion
You can use this function to clear the action counters as needed for your application.

Availability
Available in Mac OS X v10.1 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
MacTextEditor.h

TXNClearCountForActionType

Sets the counter for the specified action type to zero.

```
OSStatus TXNClearCountForActionType (
   TXNObject iTXNObject,
   CFStringRef iActionTypeName
);
```

Parameters

iTXNObject

The text object having one or more counters that are to be set to zero.

iActionTypeName

The action type for which the counter is to be set to zero. This parameter can be a string that was passed to `TXNBeginActionGroup` (page 2621) or one of the constants described in Action Constants (page 2719).

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
MacTextEditor.h

TXNClick

Processes a mouse-down event in a window's content area.

```
void TXNClick (
   TXNObject iTXNObject,
   const EventRecord *iEvent
);
```

Parameters

iTXNObject

The text object in which the mouse-down event occurred.

iEvent

A pointer to the event record that contains the mouse-down event to process.

Discussion
When you pass the event to the `TXNClick` function, it responds to the user's action by scrolling, selecting text, playing a sound or movie, handling a drag–and-drop operation, or responding to a double- or triple-click, as appropriate. Before you call `TXNClick`, you should make sure that the front window belongs to your application.

Availability
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNConvertFromPublicScrap

Converts the Clipboard content to the private MLTE scrap. (Deprecated in Mac OS X v10.3. This function isn't needed in Mac OS X.)

Not recommended.

```
OSStatus TXNConvertFromPublicScrap (
   void
);
```

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Discussion
You should call the `TXNConvertFromPublicScrap` function after another application has modified the contents of the Clipboard. Calling the `TXNConvertFromPublicScrap` function ensures that the contents of the system Clipboard are available to your application. Typically, when you receive a resume event, you call the Scrap Manager function `GetCurrentScrap` to determine whether the Clipboard content has been modified. If so, you should then call `TXNConvertFromPublicScrap`.

Special Considerations

The function `TXNConvertFromPublicScrap` is no longer needed in Mac OS X version 10.2 and later. Calling the function `TXNPaste` automatically handles conversion from public scrap.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.3.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNConvertToPublicScrap

Converts the private MLTE scrap content to the Clipboard. (Deprecated in Mac OS X v10.3. This function isn't needed in Mac OS X.)

Not recommended.

```
OSStatus TXNConvertToPublicScrap (
   void
);
```

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.3.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNCopy

Copies the current selection to the private MLTE scrap.

```
OSStatus TXNCopy (
   TXNObject iTXNObject
);
```

Parameters

iTXNObject
The text object for the current text area.

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Discussion
You can use the `TXNCopy` function to respond to a user-requested copy action. Before you call `TXNCopy`, you can use the `TXNIsSelectionEmpty` (page 2665) function to determine whether any text is selected.

The `TXNCopy` function copies the current selection to the MLTE scrap. In a Carbon application, the Scrap Manager automatically converts your application's private scrap to the Clipboard so it is available to other applications. In a Classic application, you must call the function `TXNConvertToPublicScrap` (page 2627) after you call `TXNCopy`.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNCopyTypeIdentifiersForRange

Obtains an array of universal type identifiers for a TXNObject.

```
OSStatus TXNCopyTypeIdentifiersForRange (
   TXNObject iTXNObject,
   TXNOffset iStartOffset,
   TXNOffset iEndOffset,
   CFArrayRef *oTypeIdentifiersForRange
);
```

Parameters

iTXNObject
The text object.

iStartOffset
The starting offset in `iTXNObject`.

iEndOffset
The ending offset in `iTXNObject`.

oTypeIdentifiersForRange
A pointer to a `CFArrayRef`. On return, the array contains the list of universal type identifiers that MLTE supports. Each entry in the array is a `CFStringRef`.

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Discussion
Some file formats support limited embedding of data when writing to disk, and use attachments, such as Rich Text Format (RTF), instead.

Use this function to get a list of universal type identifiers that can be used when calling `TXNWriteRangeToCFURL` (page 2699) to write the object out to disk with no data loss. Note that support for new document formats could be added in the future.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNCountRunsInRange

Obtains a count of the style runs in a range of data.

```
OSStatus TXNCountRunsInRange (
   TXNObject iTXNObject,
   TXNOffset iStartOffset,
   TXNOffset iEndOffset,
   ItemCount *oRunCount
);
```

Parameters
iTXNObject
The text object for the current text area.

iStartOffset
The beginning offset of the range of data in the document that you want to examine. Note that this offset is a generic counter of elements (such as characters, pictures, and movies), not an offset into memory.

iEndOffset
The ending offset of the range of data in the document that you want to examine. Note that this offset is a generic counter of elements, not an offset into memory.

oRunCount
On return, a pointer to the number of style runs in the specified range.

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Discussion
Given a range of data in a document that is specified by a starting and ending offset, you can use the `TXNCountRunsInRange` function to obtain a count of the changes in text styles, graphics, movies, or sounds in that range. Once you have a run count, you can supply this information to the function `TXNGetIndexedRunInfoFromRange` (page 2654) in order to obtain information about the runs themselves.

Offsets in MLTE are always character offsets.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNCreateObject

Creates a new MLTE text object which is an opaque structure that handles text formatting at the document level.

```
OSStatus TXNCreateObject (
   const HIRect *iFrameRect,
   TXNFrameOptions iFrameOptions,
   TXNObject *oTXNObject
);
```

Parameters

iFrameRect
A pointer to a rectangle used to specify the destination and view rectangles for the text object. Pass `NULL` if you want to use the window port rectangle as the view and destination rectangles when the object is attached later to a window. See the function `TXNAttachObjectToWindowRef`.

iFrameOptions
A value that specifies the options you want the frame to support. See Frame Option Masks (page 2749) for a description of possible values.

oTXNObject
On input, a pointer to a structure of type `TXNObject`. On output, points to the opaque text object data structure allocated by the function. You need to pass this object to most MLTE functions.

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Discussion
For each document, a new text object is allocated by the `TXNCreateObject` function and returned in the `oTXNObject` parameter. The object is allocated only if no errors occur. If there is an error during the allocation process, MLTE frees the text object.

If you are writing a text editing application, you may want to call the `TXNCreateObject` function when the application launches (a new document will be displayed) and whenever the user selects New from the File menu. In addition, many MLTE functions require you to pass a text object.

If you want to create a read-only document, you need to pass the option `kTXNReadOnlyMask` in the `iFrameOptions` parameter. Note that this option puts the text object into a state that does not allow user input. However, your application can put data into the text object by calling the function `TXNSetData`. If

you want the text object set into a more restrictive read-only state that does not allow user input or your application to put data into the text object programmatically, you need to call the function `TXNSetTXNObjectControls`, passing the tag `kTXNIOPrivilegesTag`. If you choose to set the text object into this restrictive state, you will get an error if you try to call the function `TXNSetData` on the text object. (In this case, you can change the text object to a less restrictive state by calling `TXNSetTXNObjectControls`, passing the tag `kTXNNoUserIOTag`.)

Because of how MLTE uses Carbon events internally, the window in which the document is displayed must have the standard event handlers installed. You can install standard event handlers in one of the following ways:

- When you create the window, add the attribute `kWindowStandardHandlerAttribute` to the window. See *Window Manager Reference* for more information.
- Call the Carbon Event Manager function `InstallStandardEventHandler` on the window's event target. See *Handling Carbon Events* for more information.

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNCut

Deletes the current selection and copies it to the private MLTE scrap.

```
OSStatus TXNCut (
   TXNObject iTXNObject
);
```

Parameters

iTXNObject
The text object for the current text area.

Return Value
A result code. See "MLTE Result Codes" (page 2773).`TXNCut` also returns Scrap Manager errors.

Discussion
You can use the `TXNCut` function to respond to a user-requested cut action. Before you call `TXNCut`, you can use the `TXNIsSelectionEmpty` (page 2665) function to determine whether any text is selected. The `TXNCut` function deletes the current selection and then copies it to the private MLTE scrap. In a Carbon application, the Scrap Manager automatically converts your application's private scrap to the Clipboard so it is available to other applications. In a Classic application, you must call the function `TXNConvertToPublicScrap` (page 2627) after you call `TXNCut` to move the selection to the Clipboard.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNDataSize

Reports the amount of memory used to hold the text in a given text object.

```
ByteCount TXNDataSize (
   TXNObject iTXNObject
);
```

Parameters

iTXNObject

The text object that you want to examine.

Return Value

The number of bytes required to hold the characters.

Discussion

You can use this function to determine how large a handle should be if, for example, you copy text. Note that because every individual sound, picture, or movie in a text object is represented by a single character in the text buffer, the `TXNDataSize` function returns a value that does not necessarily represent the true size of any non-text data.

If you are using Unicode and you want to know the number of characters, divide the returned `ByteCount` value by `sizeof(UniChar)` or 2, since MLTE uses the 16-bit Unicode Transformation Format (UTF-16).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNDeleteObject

Deletes a previously allocated text object.

```
void TXNDeleteObject (
   TXNObject iTXNObject
);
```

Parameters

iTXNObject

The text object you want to delete.

Discussion

You should call the `TXNDeleteObject` function when you close the window associated with a text object. The function `TXNDeleteObject` releases the specified text object and all associated data structures from memory. If the object has multiple frames, all frames are deleted.

Version Notes

Multiple frames are not yet implemented in MLTE.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
MacTextEditor.h

TXNDisposeFontMenuObject

Disposes of a Font menu object. (Deprecated in Mac OS X v10.5.)

```
OSStatus TXNDisposeFontMenuObject (
   TXNFontMenuObject iTXNFontMenuObject
);
```

Parameters

iTXNFontMenuObject
The Font menu object which you want to dispose of.

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Discussion
The `TXNDisposeFontMenuObject` function releases the specified Font menu object from memory. Note that `TXNDisposeFontMenuObject` does not dispose of the main Font menu handle that is associated with the Font menu object. You are responsible for disposing of the Font menu handle after calling `TXNDisposeFontMenuObject`. However, `TXNDisposeFontMenuObject` does dispose of any submenus that MLTE creates to support Apple Type Services for Unicode Imaging (ATSUI) fonts.

A good time to call the `TXNDisposeFontMenuObject` function is when your application quits (that is usually when your application no longer needs the Font menu). You can dispose of the Font menu object as part of a "terminate" function you create to do cleanup and termination tasks.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
MacTextEditor.h

TXNDoFontMenuSelection

Changes the font of the current selection. (Deprecated in Mac OS X v10.5.)

```
OSStatus TXNDoFontMenuSelection (
   TXNObject iTXNObject,
   TXNFontMenuObject iTXNFontMenuObject,
   SInt16 iMenuID,
   SInt16 iMenuItem
);
```

Parameters

iTXNObject
The text object that contains the current selection.

iTXNFontMenuObject
The Font menu object that identifies the current Font menu.

menuID

The menu ID of the selected menu. You should supply the high 16 bits of the long word obtained from the Menu Manager function `MenuSelect`. You must pass the menu ID because the Font menu may have hierarchical submenus.

menuItem

A value that identifies the selected menu item. You should supply the low 16 bits of the long word obtained from the Menu Manager function `MenuSelect`.

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Discussion
When you receive a mouse-down event in a menu used in your application, you typically call the Menu Manager function `MenuSelect` to determine which menu and menu item the user has chosen. After calling the `MenuSelect` function, you should check whether the mouse-down event occurred in a menu specific to your application, other than the standard menus such as File, Edit, and Font. If the mouse-down event did not occur in a menu specific to your application, you should pass the IDs of the menu and menu item to the `TXNDoFontMenuSelection` function. If the value you supply in the `iMenuID` parameter identifies the Font menu or one of its submenus, `TXNDoFontMenuSelection` changes the font of the currently selected text to the font that the user selects.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNDragReceiver

Handles dragged data in a text object for which a custom drag handler is already in place.

```
OSErr TXNDragReceiver (
   TXNObject iTXNObject,
   TXNFrameID iTXNFrameID,
   WindowRef iWindow,
   DragReference iDragReference,
   Boolean iDifferentObjectSameWindow
);
```

Parameters

iTXNObject

The text object that is receiving the dragged data.

iTXNFrameID

The frame ID of the text object that is receiving the dragged data. You obtain a `TXNFrameID` when you create a text object with the `TXNNewObject` (page 2667) function.

iWindow

A pointer to the window containing the text object that is receiving the dragged data. You obtain this pointer from the appropriate Drag Manager function.

iDragReference

The drag reference you want MLTE to handle. You obtain a drag reference by calling the appropriate Drag Manager function.

iDifferentObjectSameWindow

A `Boolean` value. Pass `true` if the drag operation is in the same window that it started in, but in a different text object within that window. If there is only one text object in a window, you should always pass `false`. You should also pass `false` if the drag operation has moved into a different window than the one in which it originated. You can determine whether the drag operation has left the originating window by calling the Drag Manager function `GetDragAttributes`.

Return Value
A result code. See "MLTE Result Codes" (page 2773). A Drag Manager result code.

Discussion
You would not typically use the `TXNDragReceiver` function, because MLTE provides basic drag management for you.

However, you might call `TXNDragReceiver` if your application needs to examine the dragged data prior to MLTE handling it, or if you have multiple text objects in a window, or if you have your own drag management infrastructure that you want to use.

You must inform MLTE that you wish to handle some aspect of the drag process by passing the `TXNFrameOptions` value `kTXNDoNotInstallDragProcsMask` in the `iFrameOptions` parameter of `TXNNewObject` (page 2667). If you do so, you are responsible for calling the drag handlers for the drag operation. Then, you should call `TXNDragReceiver` when your drag receiver is called and you want MLTE to take over control of the drag reception process.

When you call `TXNDragReceiver`, MLTE takes over the drag operation and handles everything from that point onward. This includes determining whether the text object is a valid drop target and if the dragged data is an MLTE-supported type, as well as managing the addition of the dragged data to the text object.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNDragTracker

Handles tracking a drag event in a text object for which a custom drag handler is already in place.

```
OSErr TXNDragTracker (
   TXNObject iTXNObject,
   TXNFrameID iTXNFrameID,
   DragTrackingMessage iMessage,
   WindowRef iWindow,
   DragReference iDragReference,
   Boolean iDifferentObjectSameWindow
);
```

Parameters

iTXNObject

The text object in which you need to track a drag event.

iTXNFrameID
The frame ID of the text object in which you need to track a drag event. You obtain a `TXNFrameID` when you create a text object with the `TXNNewObject` (page 2667) function.

iMessage
A drag message obtained from the appropriate Drag Manager function.

iWindow
A pointer to the window containing the text object in which you need to track a drag event. You obtain this pointer from the appropriate Drag Manager function.

iDragReference
The drag reference that you want MLTE to handle. You obtain a drag reference by calling the appropriate Drag Manager function.

iDifferentObjectSameWindow
A value of type `Boolean`. If your application displays more than one text object per window, pass `true` when the drag operation moves out of one object's view rectangle and into another text object's view rectangle.

Return Value
A result code. See "MLTE Result Codes" (page 2773). A Drag Manager result code.

Discussion
You would not typically use the `TXNDragTracker` function, because MLTE provides basic drag management for you.

However, you might call `TXNDragTracker` if your application needs to examine the dragged data prior to MLTE handling it, or if you have multiple text objects in a window, or if you have your own drag management infrastructure that you want to use.

You must inform MLTE that you wish to handle some aspect of the drag process by passing the `TXNFrameOptions` value `kTXNDoNotInstallDragProcsMask` in the `iFrameOptions` parameter of `TXNNewObject` (page 2667). If you do so, you are responsible for calling the drag handlers for the drag operation. Then, you should call `TXNDragTracker` when your drag tracker is called and you want MLTE to take over control of the drag tracking process.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNDraw

Redraws the text area, including any scroll bars associated with the text frame. (Deprecated in Mac OS X v10.3. Use the `TXNDrawObject` (page 2638).)

Not recommended.

```
void TXNDraw (
   TXNObject iTXNObject,
   GWorldPtr iDrawPort
);
```

Parameters

iTXNObject

The text object whose text is to be redrawn.

iDrawPort

A value of type `GWorldPtr`. Pass a valid pointer or `NULL`. If you pass `NULL`, the `TXNDraw` function redraws the text area into the port that is currently associated with the text object. If you pass a valid pointer instead of `NULL`, `TXNDraw` redraws the text area into the specified port, and does not update the selection. You should pass `NULL` if you want to draw on the screen but pass a valid pointer if you want to take a snapshot of the screen to save or print.

Discussion

You can call the `TXNDraw` function in response to an update event for a window that contains multiple text objects or other graphic elements. If necessary, your application is also responsible for calling the Window Manager functions `BeginUpdate` and `EndUpdate` in response to the update event.

If there is nothing in your window except a single MLTE text object, you should call the `TXNUpdate` (page 2698) function to redraw the area instead of calling `TXNDraw`. The `TXNUpdate` f unction draws everything in the frame, and you do not have to call the Window Manager functions `BeginUpdate` and `EndUpdate` yourself.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.3.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNDrawCFStringTextBox

Draws text from a Core Foundation string (`CFString`) in the specified rectangle.

```
OSStatus TXNDrawCFStringTextBox (
   CFStringRef iText,
   Rect *ioBox,
   ATSUStyle iStyle,
   const TXNTextBoxOptionsData *iOptions
);
```

Parameters

iText

A reference to the Core Foundation string you want drawn in the text box. A Core Foundation string is an array of Unicode characters along with a count of the number of characters in the string. A CFString object is stored as efficiently as possible, so the memory required to store the string is often less than that required to store a simple array of Unicode characters.

ioBox

On input, a pointer to the rectangle that specifies the text box in which the text is to be displayed. On return, MLTE updates the rectangle to reflect the minimum bounding rectangle that encloses the text. If you pass the constant `kTXNDontUpdateBoxRectMask` in the `ioOptions` parameter then the rectangle is not updated. If the rectangle already has text displayed in it, you should call the QuickDraw function `EraseRect` before you call the `TXNDrawUnicodeTextBox` function. The drawing is clipped to the rectangle unless you specify a rotation as one of the text options you set with the `ioOptions` parameter.

iStyle

An ATSUI style to use to display the text. This parameter is optional. If you pass `NULL`, MLTE creates an ATSUI style based on the information (text size, type face, color, and so forth) associated with the current graphics port. See the Apple Type Services for Unicode Imaging documentation for a description of the `ATSUStyle` data type.

iOptions

A pointer to a `TXNTextBoxOptionsData` structure associated with this text object. This is optional. If you pass `NULL`, MLTE uses the settings for the current graphics port. You can use the `TXNTextBoxOptionsData` structure to specify a number of options, such as text orientation (vertical or horizontal), justification, alignment, and font substitution for text that cannot be rendered using the specified font. See Text Box Options Masks (page 2770) for a description of the options you can specify.

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Discussion
You can use the `TXNDrawCFStringTextBox` function to display mono-style Unicode text. You do not need to initialize MLTE to use this function because it uses Apple Type Services for Unicode Imaging (ATSUI) directly.

If you display text justified, it is justified in the direction of the display. Horizontal text is justified horizontally, but not vertically. Vertical text is justified vertically, but not horizontally.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNDrawObject

Draws a text object in the last window set by your application.

```
OSStatus TXNDrawObject (
   TXNObject iTXNObject,
   const HIRect *iClipRect,
   TXNDrawItems iDrawItems
);
```

Parameters

iTXNObject

The text object you want to draw. You can call the function `TXNCreateObject` to allocate a text object.

iClipRect

A pointer to an `HIRect` data structure. If the rectangle is `NULL` MLTE uses the view rectangle when drawing. Otherwise, MLTE uses the rectangle that is the intersection of the `iClipRect` rectangle and the view rectangle.

iDrawItems

A Draw Items Masks (page 2737) value that specifies which elements are drawn. Pass `kTXNDrawItemScrollbarsMask` if you want the scroll bars drawn; `kTXNDrawItemTextMask` to render the text; `kTXNDrawItemTextAndSelectionMask` to render the text and the current selection, and `kTXNDrawItemAllMask` to draw the scroll bars, text, and the current selection.

Discussion

This function has no effect for text objects that have the visibility tag set to `false`.

Availability

Available in Mac OS X v10.3 and later.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNDrawUnicodeTextBox

Draws a Unicode string in the specified rectangle.

```
OSStatus TXNDrawUnicodeTextBox (
   const UniChar iText[],
   UniCharCount iLen,
   Rect *ioBox,
   ATSUStyle iStyle,
   const TXNTextBoxOptionsData *iOptions
);
```

Parameters

iText

The Unicode string you want drawn in the text box. The string should be in 16-bit Unicode Transformation Format (UTF-16).

iLen

The number of Unicode characters contained in the Unicode string.

ioBox

On input, a pointer to the rectangle, in local coordinates, that specifies the text box in which the text is to be displayed. On return, MLTE updates the rectangle to reflect the minimum bounding rectangle that encloses the text. If you pass the constant `kTXNDontUpdateBoxRectMask` in the `ioOptions` parameter, the rectangle is not updated. If the rectangle already has text displayed in it, you should call the QuickDraw function `EraseRect` before you call the `TXNDrawUnicodeTextBox` function. The drawing is clipped to the rectangle unless you specify a rotation as one of the text options you set with the `ioOptions` parameter.

iStyle

An ATSUI style to use to display the text. This parameter is optional. If you pass `NULL`, MLTE creates an ATSUI style based on the information (text size, type face, color, and so forth) associated with the current graphics port. See the Apple Type Services for Unicode Imaging documentation for a description of the `ATSUStyle` data type.

ioOptions

A pointer to a `TXNTextBoxOptionsData` structure associated with this text object. This is optional. If you pass `NULL`, MLTE uses the settings for the current graphics port. You can use the `TXNTextBoxOptionsData` structure to specify a number of options, such as text orientation (vertical or horizontal), justification, alignment, and font substitution for text that cannot be rendered using the specified font. See Text Box Options Masks (page 2770) for a description of the options you can specify.

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Discussion
You can use the `TXNDrawUnicodeTextBox` function to display mono-style Unicode text. You do not need to initialize MLTE to use this function because it uses Apple Type Services for Unicode Imaging (ATSUI) directly.

If you display text justified, it is justified in the direction of the display. Horizontal text is justified horizontally, but not vertically. Vertical text is justified vertically, but not horizontally.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNEchoMode

Determines whether a specified character is drawn instead of the glyph associated with the input character.

```
OSStatus TXNEchoMode (
   TXNObject iTXNObject,
   UniChar iEchoCharacter,
   TextEncoding iEncoding,
   Boolean iOn
);
```

Parameters

iTXNObject

The text object for the current text area.

iEchoCharacter

A value that specifies the substitute character.

iEncoding

The text encoding from which the substitute character is drawn. See the Text Encoding Conversion Manager reference documentation for a discussion of the text encoding data type and of possible text encoding values.

iOn

A `Boolean` value. Pass `true` to turn on character substitution. Pass `false` to turn it off. When you enable character substitution for a text object, all characters in the text area have the character specified by the `iEchoCharacter` parameter substituted for the actual glyph when MLTE draws the text.

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Discussion
You can use the `TXNEchoMode` function when you want to hide what the user types, such as a password in a login dialog box.

The substitution character is a `UniChar` data type to facilitate passing any 2-byte character. The encoding parameter actually determines the encoding MLTE uses to locate a font and display a character. Thus if you want to display the diamond character from the Shift-JIS encoding for Mac OS, you would pass the value `0x86A6` for the character, but pass an encoding value that represents Mac OS Japanese encoding.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNEndActionGroup

Ends an action group.

```
OSStatus TXNEndActionGroup (
   TXNObject iTXNObject
);
```

Parameters

iTXNObject
The text object for which an action group is to be ended.

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Discussion
The call is ignored if there is no active action group.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNFind

Finds a piece of text or a graphics, sound, or movie object.

```
OSStatus TXNFind (
   TXNObject iTXNObject,
   const TXNMatchTextRecord *iMatchTextDataPtr,
   TXNDataType iDataType,
   TXNMatchOptions iMatchOptions,
   TXNOffset iStartSearchOffset,
   TXNOffset iEndSearchOffset,
   TXNFindUPP iFindProc,
   SRefCon iRefCon,
   TXNOffset *oStartMatchOffset,
   TXNOffset *oEndMatchOffset
);
```

Parameters

iTXNObject

The text object to be searched.

iMatchTextDataPtr

A pointer to a data structure that contains the text to match, the length of that text, and the text's encoding. Pass `NULL` if you are looking for a graphics, sound, or movie object.

iDataType

The type of data for which you want to search. See Supported Data Types (page 2765) for a description of possible values. If the data type is `kTXNPictureFile`, `kTXNMovieFile`, or `kTXNSoundFile`, the default behavior is to match on any nontext object. If you want to find a specific data type, you can provide a custom find callback or ignore types that do not match what you want to find.

iMatchOptions

The matching rules to use in the find operation. See Search Criteria Masks (page 2762) for a description of possible values.

iStartSearchOffset

The offset at which the search should begin. You can use `kTXNStartOffset` if you want to search from the start of the object's data.

iEndSearchOffset

The offset at which the search should end. You can use `kTXNEndOffset` if you want to search to the end of the object's data.

iFindProc

The custom callback you want used in place of the default matching behavior. You can pass `NULL` if you want to use the default matching behavior.

iRefCon

An signed 32-bit integer. You can use this for whatever your application needs. It is passed to the custom callback specified by `iFindProc`.

oStartMatchOffset

On return, a pointer to the absolute offset that identifies the start of the match. It is set to a value of `kTXNUseCurrentSelection` if there is no match.

oEndMatchOffset

On return, a pointer to the absolute offset that identifies the end of the match. It is set to a value of `kTXNUseCurrentSelection` if there is no match.

Return Value

A result code. See "MLTE Result Codes" (page 2773).

Discussion
By default, text is matched on the basis of a binary comparison. If you set the `iMatchOptions` variable to ignore case, the characters to be searched are duplicated and case neutralized. If the want to search a large amount of text, a case insensitive search can fail due to insufficient memory.

If you set the `iMatchOptions` variable to find an entire word, then once a match is found, the matched text is tested to see if it is a word. If the `kTXNUseEncodingWordRulesBit` is set, then the Script Manager `FindWord` function is called to make this determination. If the text being searched is Unicode text, then the ATSUI word-determining functions are used to test for a word.

If the application is looking for a nontext type, then each nontext type in the document is returned. The `iFindProc` parameter lets you provide a more elaborate search engine (such as a regular expression processor) should you need one.

Offsets in MLTE are always character offsets.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNFlattenObjectToCFDataRef

Flattens a text object so it can be saved to disk or embedded with other data.

```
OSStatus TXNFlattenObjectToCFDataRef (
   TXNObject iTXNObject,
   TXNDataType iTXNDataType,
   CFDataRef *oDataRef
);
```

Parameters

`iTXNObject`
The text object that identifies the document you want to flatten. You can either call the function `TXNCreateObject` to allocate a text object or you can call the function `HITextViewGetTXNObject` (page 2611) to obtain the text object associated with an HITextView.

`iTXNDataType`
A value that specifies the format in which the data is written out.

`oDataRef`
On input, points to a structure of type `CFDataRef`. On output, points to a flattened version of the text object in the format specified by the `iTXNDataType` parameter. You are responsible to retain the returned `CFDataRef`.

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Discussion
This function supports the following data types:

- `kTXNTextData`, for text data.
- `kTXNUnicodeTextData`, for plain UTF-16.

- `kTXNTextAndMultimediaData`, for data in MLTE format.
- `kTXNRichTextFormatData`, for data in RTF format.

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNFocus

Changes the focus of a text object.

```
void TXNFocus (
   TXNObject iTXNObject,
   Boolean iBecomingFocused
);
```

Parameters

iTXNObject
The text object whose focus you want to change.

iBecomingFocused
If you pass `true`, the text object receives focus. This means the current selection or insertion point is active, text input appears at the insertion point, and the keyboard and font are synchronized. (Note that the font and keyboard are synchronized only if keyboard synchronization is enabled. See Keyboard Synchronization Settings (page 2756).) If the scroll bars are not already active, they are activated. If you pass `false`, the text object's current selection or insertion point is inactive.

Discussion
You should use the `TXNActivate` (page 2618) function to make scroll bars active while text input is not focused. This behavior is often desirable for windows with multiple text areas that are scrollable.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNForceUpdate

Forces an update of the view rectangle and the scroll bars.

```
void TXNForceUpdate (
   TXNObject iTXNObject
);
```

Parameters

iTXNObject
The text object you want to update.

Discussion
This function operates similarly to the Window Manager functions `InvalRect` and `InvalRgn`. For example, when the user increases the size of a window that contains text from a text object, the `TXNForceUpdate` function adds the new region (including two rectangles formerly occupied by the scroll bars in the smaller content area) to the update region.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNGetAccessibilityHIObject

Obtains an `HIObjectRef` representing the MLTE object for accessibility purposes.

```
OSStatus TXNGetAccessibilityHIObject (
   TXNObject iTXNObject,
   HIObjectRef *oHIObjectRef
);
```

Parameters

`iTXNObject`
The text object.

`oHIObjectRef`
On input, a pointer an `HIObjectRef` that, on return, represents the text object specified by `iTXNObject` as an accessible object.

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Discussion
For each MLTE object that a view creates, the view needs to call this function to get an `HIObjectRef` that can be used to represent the MLTE object as an accessible object.

After the view gets this `HIObjectRef`, it must add the `HIObjectRef` as a child of itself. The accessibility engine then routes events to MLTE accessible objects automatically.

The view must install Carbon Event handlers for `kEventAccessibleGetAllAttributeNames` and `kEventAccessibleGetNamedAttribute`, using the `HIObjectRef` as the target, to provide information for at least the following required attributes:

- kAXRoleAttribute
- kAXRoleDescriptionAttribute
- kAXWindowAttribute
- kAXPositionAttribute
- kAXSizeAttribute

MLTE also installs handlers for `kEventAccessibleGetAllAttributeNames` and `kEventAccessibleGetNamedAttribute`. Note that these handlers are not called unless the client-installed handlers return `eventNotHandledErr`. These handlers return information for the following attributes:

- kAXEnabledAttribute
- kAXFocusedAttribute
- kAXValueAttribute
- kAXSelectedTextAttribute
- kAXSelectedTextRangeAttribute
- kAXNumberOfCharactersAttribute
- kAXLineForIndexParameterizedAttribute
- kAXRangeForLineParameterizedAttribute
- kAXStringForRangeParameterizedAttribute
- kAXRangeForPositionParameterizedAttribute
- kAXRangeForIndexParameterizedAttribute
- kAXBoundsForRangeParameterizedAttribute
- kAXStyleRangeForIndexParameterizedAttribute

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNGetActionChangeCount

Retrieves the number of times the specified action or actions have occurred since the count was initialized or cleared. (Deprecated in Mac OS X v10.4. Use `TXNGetCountForActionType` (page 2649) instead.)

Not recommended.

```
OSStatus TXNGetActionChangeCount (
   TXNObject iTXNObject,
   TXNCountOptions iOptions,
   ItemCount *oCount
);
```

Parameters

iTXNObject
The text object whose action count you want to retrieve.

iOptions
The `TXNCountOptions` to use when retrieving the count. See Action Count Masks (page 2721) for information on the options you can supply.

oCount
On return, a pointer to the action count.

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Availability
Available in Mac OS X v10.1 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNGetChangeCount

Retrieves the number of times a document has been changed.

```
ItemCount TXNGetChangeCount (
   TXNObject iTXNObject
);
```

Parameters

iTXNObject
The text object whose changes you want to count.

Return Value
The total number of changes since the document was last saved. If the document is new, the number of changes since it was created.

Discussion
The change count increments for every executed command such as Cut or Copy. An uninterrupted sequence of key-down events increments the count by 1. The count is cleared each time the text object is saved. You can use this function to determine if your application should make the Save item in the File menu active.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNGetCommandEventSupport

Obtains the command event support that is currently set for an MLTE object.

```
OSStatus TXNGetCommandEventSupport (
   TXNObject iTXNObject,
   TXNCommandEventSupportOptions *oOptions
);
```

Parameters

iTXNObject
The text object.

oOptions

A pointer to a value of type `TXNCommandEventSupportOptions` that, on return, contains the option settings for the text object specified by `iTXNObject`. For possible values, see Command Event Support Options (page 2729).

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Availability
Available in Mac OS X v10.4 and later.

Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNGetContinuousTypeAttributes

Checks to see if the attributes of the current selection are continuous.

```
OSStatus TXNGetContinuousTypeAttributes (
   TXNObject iTXNObject,
   TXNContinuousFlags *oContinuousFlags,
   ItemCount iCount,
   TXNTypeAttributes ioTypeAttributes[]
);
```

Parameters

iTXNObject

The text object that contains the current selection.

oContinuousFlags

On return, a pointer to a value that specifies whether text attributes are continuous. See Continuous Style Information Masks (page 2731) for a description of possible values. If a particular bit is set and if your application has passed a `tag` value in a `TXNTypeAttributes` structure in the array that corresponds to the bit, then your application should display a check mark next to the appropriate menu item.

ioCount

The number of `TXNTypeAttributes` structures in the `ioTypeAttributes` array.

ioTypeAttributes

An array of `TXNTypeAttributes` structures. The `tag` values in this array indicate the text attributes in which the application is interested. It cannot be `NULL`. If you are using ATSUI and you want to know the ATSUI font ID, you should set the `tag` field to `kATSUFontTag`, which is a constant in `ATSUnicode.h`. If you are using ATSUI and you set the `tag` field to a QuickDraw font ID, then the QuickDraw font to which the ATSUI font maps is returned. Note that this is not always the same font since many ATSUI fonts are not supported by QuickDraw.

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Discussion
You can use this function to determine whether your application should display check marks in the Font, Style, Size, and Color menus. If these are the only attributes in which you are interested, you can use this function on systems that use QuickDraw or Apple Type Services for Unicode Imaging (ATSUI).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNGetCountForActionType

Gets the number of times a given type of action has occurred.

```
OSStatus TXNGetCountForActionType (
   TXNObject iTXNObject,
   CFStringRef iActionTypeName,
   ItemCount *oCount
);
```

Parameters

iTXNObject
The text object to query.

iActionTypeName
The action type that is to be included when retrieving the count. This parameter can be the string that was passed to `TXNBeginActionGroup` (page 2621) or one of the constants described in Action Constants (page 2719).

oCount
On return, the number of times the action specified by `iActionTypeName` has occurred.

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Discussion
Call `TXNClearCountForActionType` (page 2626) to reset the counters.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNGetData

Copies a range of data.

```
OSStatus TXNGetData (
   TXNObject iTXNObject,
   TXNOffset iStartOffset,
   TXNOffset iEndOffset,
   Handle *oDataHandle
);
```

Parameters

iTXNObject

The text object that contains the data you want to copy.

iStartOffset

The absolute offset from which data copying should begin. Make sure the `iStartOffset` and `iEndOffset` parameters do not specify a range that includes text and nontext data (that is, crosses a data type boundary). You can use the `TXNGetSelection` (page 2658) function to get the absolute offsets of the current selection.

iEndOffset

The absolute offset at which data copying should end. You can use the `TXNGetSelection` function to get the absolute offsets of the current selection.

oDataHandle

A pointer to a handle. On return, the handle points to the requested data. `TXNGetData` allocates the handle as necessary. Your application must dispose of the handle.

Return Value

A result code. See "MLTE Result Codes" (page 2773). The `iStartOffset` and `iEndOffset` parameters can specify a range that crosses a style run boundary but not a range that crosses a data type boundary. If the range includes text and nontext data, the `TXNGetData` function returns the error code `kTXNIllegalToCrossDataBoundariesErr`.

Discussion

You first need to use the `TXNCountRunsInRange` (page 2629) function to find the number of data runs in a given range. Then you can examine each run's type and text attributes with the `TXNGetIndexedRunInfoFromRange` (page 2654) function. Finally, use the `TXNGetData` function to examine data for each run of interest to you. The function does not check to see that the copied data aligns on a word boundary; data is simply copied as specified by the offset values.

Offsets in MLTE are always character offsets.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNGetDataEncoded

Copies the text in a specified range, and if necessary, translates the text to match your application's preferred encoding.

```
OSStatus TXNGetDataEncoded (
   TXNObject iTXNObject,
   TXNOffset iStartOffset,
   TXNOffset iEndOffset,
   Handle *oDataHandle,
   TXNDataType iEncoding
);
```

Parameters

iTXNObject

The text object that contains the data you want to copy.

iStartOffset

The absolute offset from which data copying should begin. You can use the `TXNGetSelection` (page 2658) function to get the absolute offsets of the current selection.

iEndOffset

The absolute offset at which data copying should end. You can use the `TXNGetSelection` (page 2658) function to get the absolute offsets of the current selection.

oDataHandle

A pointer to a handle. On return, a handle to the requested data. `TXNGetDataEncoded` allocates the handle as necessary. Your application must dispose of the handle. If there is no text in the range specified, the returned handle is `NULL`, and the function returns `noErr`.

encoding

The type of data to be encoded. See Supported Data Types (page 2765) for a full description of possible values. You should specify either the `kTXNTextData` or `kTXNUnicodeTextData` constant. If the `iEncoding` parameter specifies an encoding different from that used to store the text data internally, the Conversion Manager translates the data to the specified type (text or Unicode). If the `iEncoding` parameter is not recognized, the data is returned in the current encoding. On systems that do not use ATSUI version 1.1 or later, the current encoding is the Mac OS encoding. Otherwise, the current encoding is Unicode.

Return Value

A result code. See "MLTE Result Codes" (page 2773).

Discussion

Offsets in MLTE are always character offsets.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNGetEventTarget

Obtains the current event target for a TXNObject.

```
OSStatus TXNGetEventTarget (
   TXNObject iTXNObject,
   HIObjectRef *oEventTarget
);
```

Parameters

iTXNObject

The text object.

oEventTarget

A pointer to an `HIObjectRef` that, on return, points to the current event target for the TXNObject specified by `iTXNObject`.

Return Value

A result code. See "MLTE Result Codes" (page 2773).

Discussion

This function obtains the current event target for the TXNObject specified by `iTXNObject`. Use this function to obtain the target and then install your own handlers.

Availability

Available in Mac OS X v10.4 and later.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNGetFontDefaults

Makes a copy of the font descriptions for a given text object. (Deprecated in Mac OS X v10.4.)

```
OSStatus TXNGetFontDefaults (
   TXNObject iTXNObject,
   ItemCount *ioCount,
   TXNMacOSPreferredFontDescription oFontDefaults[]
);
```

Parameters

iTXNObject

The text object for the document whose default font settings you want to copy.

ioCount

A pointer to a value of type `ItemCount`. You need to call the `TXNGetFontDefaults` function twice (see Discussion). The first time you call the function, pass `NULL`. On return, the `ioCount` parameter specifies the number of font descriptions associated with the text object. The second time you call `TXNGetFontDefaults`, on return the `ioCount` parameter points to the number of font descriptions in the `iFontDefaults` array.

iFontDefaults

An array of `TXNMacOSPreferredFontDescription` structures to be filled. You need to call the `TXNGetFontDefaults` function twice (see Discussion). The first time you call the function pass `NULL`. The second time you call `TXNGetFontDefaults`, you can initialize the `iFontDefaults` array to have the number of elements specified by the `ioCount` parameter. Then on return, the `iFontDefaults` parameter contains the font descriptions for the text object.

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Discussion
You need to call this function twice: once to get the number of font default descriptions (returned in the `ioCount` parameter), and the second time to get the font default data.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNGetFontMenuHandle

Gets the Font menu handle that belongs to a Font menu object. (Deprecated in Mac OS X v10.5.)

```
OSStatus TXNGetFontMenuHandle (
   TXNFontMenuObject iTXNFontMenuObject,
   MenuRef *oFontMenuHandle
);
```

Parameters

iTXNFontMenuObject
A Font menu object.

fontMenuHandle
A pointer to a menu handle. On return, a pointer to the Font menu created when the Font menu object was created. MLTE makes a copy of the menu handle. Your application should not dispose of the menu handle until it disposes of the Font menu object by calling the `TXNDisposeFontMenuObject` (page 2633).

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNGetHIRect

Obtains the values for the current view, destination, or text rectangle.

```
OSStatus TXNGetHIRect (
   TXNObject iTXNObject,
   TXNRectKey iTXNRectKey,
   HIRect *oRectangle
);
```

Parameters

iTXNObject

The text object for the current text area. You can either call the function `TXNCreateObject` to allocate a text object or you can call the function `HITextViewGetTXNObject` (page 2611) to obtain the text object associated with an HITextView.

iTXNRectKey

A value that specifies the rectangle you want the function to obtain. See `Rectangle Keys` (page 2759) for a list of the constants you can supply.

oRectangle

On output, points to the `HIRect` data structure that contains the coordinates for the requested rectangle.

Return Value

A result code. See "MLTE Result Codes" (page 2773).

Availability

Available in Mac OS X v10.3 and later.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNGetIndexedRunInfoFromRange

Gets information about a run in a range of data.

```
OSStatus TXNGetIndexedRunInfoFromRange (
   TXNObject iTXNObject,
   ItemCount iIndex,
   TXNOffset iStartOffset,
   TXNOffset iEndOffset,
   TXNOffset *oRunStartOffset,
   TXNOffset *oRunEndOffset,
   TXNDataType *oRunDataType,
   ItemCount iTypeAttributeCount,
   TXNTypeAttributes *ioTypeAttributes
);
```

Parameters

iTXNObject

The text object for the current text area.

iIndex

The value that corresponds to the run for which you want to get information. You call the `TXNCountRunsInRange` (page 2629) function to get the number of runs in a range. The `iIndex` parameter is zero-based, so its possible values are from 0 to the number of runs in a range minus 1. Note that the index is relative to the first run in the range specified by the `iStartOffset` and `iEndOffset` parameters, not for the entire document.

iStartOffset

The offset at which you want to start to obtain run information. This value must be the same value that you passed previously to the function `TXNCountRunsInRange`.

iEndOffset

The offset at which you want run information to end. This value must be the same value that you passed previously to the function `TXNCountRunsInRange`.

oRunStartOffset

On return, a pointer to a value that identifies the start of run relative to the beginning of the text, not the beginning of the range you specified in the `iStartOffset` parameter.

oRunEndOffset

On return, a pointer to a value that identifies the end of the run relative to the beginning of the text, not the beginning of the range you specified in the `iStartOffset` parameter.

oRunDataType

On return, a pointer to a value that identifies the type of data in the run. See Supported Data Types (page 2765) for a description of possible values.

iTypeAttributeCount

The number of font attributes.

ioTypeAttributes

A pointer to a structure of type `TXNTypeAttributes`. On input, you specify the attribute (such as size) in the `tag` field and the attribute size in the `size` field. You can pass `NULL` for the `data` field only if the size of the returned value <= 4. Otherwise a pointer to an appropriately sized block of data must be placed in one of the other members of the `ioTypeAttributes` union, such as, `dataPtr`, `atsuFeatures`, `atsuVariations`. On return, the `data` field contains the attribute data. The `data` field is a union that serves either as a 32-bit integer or a 32-bit pointer, depending on the `size` field.

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Discussion
You should first call the `TXNCountRunsInRange` function to get the count. The `TXNTypeAttributes` structure must specify the text attribute in which the application is interested. In other words, the `tag` and `size` fields must be set.

Offsets in MLTE are always character offsets.

If a tag specified in the `ioTypeAttributes` parameter is not recognized by MLTE—that is, the tag isn't a `TXTNTag` value, a `TXNTypeRunAttributes` value, or a valid style run attribute tag (`ATSUAttributeTag`)—then the constant `kTXNAttributeTagInvalidForRunErr` is returned in the `ioTypeAttributes->data.dataValue` field for that particular tag. The function continues to process the remaining tags, but returns the result code `kTXNSomeOrAllTagsInvalidForRunErr`. Both of these values (`kTXNAttributeTagInvalidForRunErr` and `kTXNSomeOrAllTagsInvalidForRunErr`) are used either when a tag is not recognized or a tag is recognized but some error occurred in trying to obtain the attribute.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNGetLineCount

Gets the total number of lines in a text object.

```
OSStatus TXNGetLineCount (
   TXNObject iTXNObject,
   ItemCount *oLineTotal
);
```

Parameters

iTXNObject

The text object that identifies the document whose line count you want to get.

oLineTotal

On return, a pointer to the number of lines in the text object.

Return Value

A result code. See "MLTE Result Codes" (page 2773).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNGetLineMetrics

Gets information about line width and height for a specified line of data in a text object.

```
OSStatus TXNGetLineMetrics (
   TXNObject iTXNObject,
   unsigned long iLineNumber,
   Fixed *oLineWidth,
   Fixed *oLineHeight
);
```

Parameters

iTXNObject

The text object that identifies the document whose line metrics you want to get.

iLineNumber

A value that specifies the line whose metrics you want to retrieve. You should use the `TXNGetLineCount` (page 2656) function to determine how many lines are in the text object so that you specify a valid line number. Line numbers start at 0.

oLineWidth

On return, a pointer to the width of the line, in pixels.

oLineHeight

On return, a pointer to the height of the line, in pixels.

Return Value

A result code. See "MLTE Result Codes" (page 2773).

Discussion

You can use height and width information to adjust the size of the text object's frame.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNGetRectBounds

Obtains the values for the current view, destination, and text rectangles. (Deprecated in Mac OS X v10.3. Use `TXNGetHIRect` (page 2653) instead.)

Not recommended.

```
OSStatus TXNGetRectBounds (
   TXNObject iTXNObject,
   Rect *oViewRect,
   TXNLongRect *oDestinationRect,
   TXNLongRect *oTextRect
);
```

Parameters

iTXNObject
The text object for the current text area.

oViewRect
On output, a pointer to the `Rect` data structure that contains the coordinates for the view rectangle. If you do not want to obtain this structure, pass `NULL`. The view rectangle specifies the area of the text you see. Scroll bars are drawn inside the view rectangle.

oDestinationRect
On output, a pointer to the `TXNLongRect` data structure that contains the coordinates for the destination rectangle. If you do not want to obtain this structure, pass `NULL`. The destination rectangle controls how text is laid out.

oTextRect
On output, a pointer to the `TXNLongRect` data structure that contains the coordinates for the text rectangle. If you do not want to obtain this structure, pass `NULL`. The text rectangle is the smallest rectangle needed to contain the current text. MLTE calculates the text rectangle by measuring each line of text. So this can be slow performance. The width of the text rectangle is the width of the longest line in the text.

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Discussion
You need only to pass pointers for the rectangles for which you want to obtain coordinates.

Availability
Available in Mac OS X v10.1 and later.
Deprecated in Mac OS X v10.3.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNGetSelection

Gets the absolute offsets of the current selection.

```
void TXNGetSelection (
   TXNObject iTXNObject,
   TXNOffset *oStartOffset,
   TXNOffset *oEndOffset
);
```

Parameters

iTXNObject

The text object for the current text area.

oStartOffset

On return, a pointer to the absolute starting offset of the current selection.

oEndOffset

On return, a pointer to the absolute ending offset of current selection.

Discussion

Offsets in MLTE are always character offsets. Each embedded graphics or sound object is counted as one character.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

QTMetaData

Declared In

`MacTextEditor.h`

TXNGetSleepTicks

Reports the appropriate amount of time to allot to background processes, depending on the state of the window.

```
UInt32 TXNGetSleepTicks (
   TXNObject iTXNObject
);
```

Parameters

iTXNObject

The text object that identifies the current document.

Return Value

The appropriate wait time, in ticks. You pass this value to the `WaitNextEvent` function.

Discussion

In a cooperative processing environment, your application must determine how much time to give to background processes. A tick is 1/60 of a second.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
MacTextEditor.h

TXNGetSpellCheckAsYouType

Determines whether the "Spell Check as You Type" feature is enabled.

```
Boolean TXNGetSpellCheckAsYouType (
   TXNObject iTXNObject
);
```

Parameters

iTXNObject
The text object to query.

Return Value
A Boolean whose value is `true` if the "Spell Check as You Type" feature is enabled; otherwise, `false`.

Discussion
Call `TXNSetSpellCheckAsYouType` (page 2693) to enable or disable the "Spell Check as You Type" feature.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
MacTextEditor.h

TXNGetTXNObjectControls

Gets the current formatting and privileges attributes (such as justification, line direction, tab values, and read-only status) for a text object.

```
OSStatus TXNGetTXNObjectControls (
   TXNObject iTXNObject,
   ItemCount iControlCount,
   const TXNControlTag iControlTags[],
   TXNControlData oControlData[]
);
```

Parameters

iTXNObject
The text object that identifies the document to be activated.

iControlCount
The number of items in the `iControlTags` array.

iControlTags
An array of values that specify the kind of formatting information you want returned in the `oControlData` array. See Formatting and Privileges Settings (page 2742) for a description of possible values.

oControlData

An array of TXNControlData (page 2711) unions. On return, the array contains the information that was requested through the iControlTags array. Your application must allocate the oControlData array.

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
MacTextEditor.h

TXNGetViewRect

Gets the rectangle that describes the current view of the document.

```
void TXNGetViewRect (
   TXNObject iTXNObject,
   Rect *oViewRect
);
```

Parameters

iTXNObject

The text object for the current text area.

oViewRect

On return, a pointer to a rectangle that describes the view of the document. The area described by the oViewRect parameter does not include the area that encloses the scroll bars. The coordinates of this rectangle are local to the window.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
MacTextEditor.h

TXNGetWindowRef

Returns a reference to the window to which the specified text object is attached.

```
WindowRef TXNGetWindowRef (
   TXNObject iTXNObject
);
```

Parameters

iTXNObject

The text object for which you want to obtain a window reference. You can call the function TXNCreateObject to allocate a text object.

Return Value
The window to which the text object is attached. Returns NULL if no window is attached.

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNGrowWindow

Adjusts the size of a window in response to mouse-down events in the size region of the window.

```
void TXNGrowWindow (
   TXNObject iTXNObject,
   const EventRecord *iEvent
);
```

Parameters

iTXNObject
The text object for the current text area.

iEvent
A pointer to the event record that contains the mouse-down event you want to apply to the window.

Discussion
The text object must be contained in a window and not a subframe of the window; otherwise the function does not adjust the size of the window. Before you call `TXNGrowWindow`, you should make sure that the front window belongs to your application.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNHIPointToOffset

Gets the offset value that corresponds to a point in local coordinates.

```
OSStatus TXNHIPointToOffset (
   TXNObject iTXNObject,
   const HIPoint *iHIPoint,
   TXNOffset *oOffset
);
```

Parameters

iTXNObject
The text object for which you want to obtain an offset value. You can either call the function `TXNCreateObject` to allocate a text object or you can call the function `HITextViewGetTXNObject` (page 2611) to obtain the text object associated with an HITextView.

iHIPoint
The local coordinates of the point for which you want to obtain the offset value.

oOffset
On output, a pointer to the offset that corresponds to the value of the `iHIPoint` parameter.

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Discussion
Offsets in MLTE are always character offsets.

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNIdle

Does idle time processing, such as flashing the cursor.

```
void TXNIdle (
   TXNObject iTXNObject
);
```

Parameters
iTXNObject
The text object that identifies the current document.

Discussion
Before you call the `TXNIdle` function, you should make sure that the front window belongs to your application.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNInitTextension

Initializes MLTE.

```
OSStatus TXNInitTextension (
   const TXNMacOSPreferredFontDescription iDefaultFonts[],
   ItemCount iCountDefaultFonts,
   TXNInitOptions iUsageFlags
);
```

Parameters

iDefaultFonts

An array of `TXNMacOSPreferredFontDescription` structures. You use this to specify a table of font information that includes the font family ID, point size, style, and script code. The table can be `NULL` or can have an entry for any script for which you would like to designate a default font. To designate that MLTE should use the default settings for a specified script, you need to supply a valid script code value in the `TextEncoding` field of the font description structure and a value of `kTXNUseScriptDefaultValue` in all other fields of the structure. In Mac OS X, the default settings are read from the Theme settings. In Mac OS 9, the default settings are read from the Script Manager.

iCountDefaultFonts

The number of scripts for which you are designating a default font in the `iDefaultFonts` array.

iUsageFlags

A value that specifies whether embedded objects should be supported. You can also specify whether MLTE should use QuickDraw as the imaging system and whether temporary memory should be used. See Initialization Option Masks (page 2754) for a description of possible values.

Return Value

A result code. See "MLTE Result Codes" (page 2773).

Discussion

For systems prior to Mac OS X v10.3, you should call this function along with any other initialization calls you make when your application starts up. If you call this function a second time, it has no effect; the defaults you set up the first time you called the `TXNInitTextension` function are in effect until you call the `TXNTerminateTextension` (page 2696) function.

For Mac OS X v10.3 and later, you do not have to call this function. However, you may want to call this function to set default fonts that are different from the system default font or to enable multimedia support.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNIsObjectAttachedToSpecificWindow

Determines whether a text object is attached to a specified window. (Deprecated in Mac OS X v10.3. Use `TXNGetWindowRef` (page 2660) instead.)

Not recommended.

```
OSStatus TXNIsObjectAttachedToSpecificWindow (
   TXNObject iTXNObject,
   WindowRef iWindow,
   Boolean *oAttached
);
```

Parameters

iTXNObject

The text object that identifies the document you want to check.

iWindow

A reference to the window against which to check attachment.

oAttached

On output, `true` if the text object is attached to the specified window; `false` otherwise.

Return Value

A result code. See "MLTE Result Codes" (page 2773).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.3.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNIsObjectAttachedToWindow

Checks to see if a text object is attached to a window. (Deprecated in Mac OS X v10.3. Use `TXNGetWindowRef` (page 2660) instead.)

Not recommended.

```
Boolean TXNIsObjectAttachedToWindow (
   TXNObject iTXNObject
);
```

Parameters

iTXNObject

The text object that identifies the document you want to check.

Return Value

A `Boolean` value; returns `true` if the object is attached.

Discussion

You can call this before you call the `TXNAttachObjectToWindow` (page 2620) function to make sure the text object is not already attached to a window. If you pass `NULL` in the `iWindow` parameter of `TXNNewObject` (page 2667) you create a text object without an associated window pointer.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.3.

Not available to 64-bit applications.

Declared In
MacTextEditor.h

TXNIsScrapPastable

Tests whether the Clipboard contains data that is supported by MLTE.

```
Boolean TXNIsScrapPastable (
   void
);
```

Return Value
Returns `true` if the data type on the Clipboard is supported.

Discussion
You can call the `TXNIsScrapPastable` function to determine if the Paste item in the Edit menu should be active.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
MacTextEditor.h

TXNIsSelectionEmpty

Determines whether the current selection is empty.

```
Boolean TXNIsSelectionEmpty (
   TXNObject iTXNObject
);
```

Parameters
iTXNObject
The text object for the current text area.

Return Value
A `Boolean` value. It is `true` if the current selection is empty.

Discussion
You can use the `TXNIsSelectionEmpty` function to determine whether your application should enable the Cut, Copy, and Clear items in the Edit menu.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
MacTextEditor.h

TXNKeyDown

Processes a key-down event.

```
void TXNKeyDown (
   TXNObject iTXNObject,
   const EventRecord *iEvent
);
```

Parameters

iTXNObject

The text object that identifies the active document.

iEvent

A pointer to the event record that contains the key-down event you want handled. You cannot pass `NULL`.

Discussion

Before you call the `TXNKeyDown` function, you should make sure that the front window belongs to your application. Text input occurs in the text object's window. This is always the case unless the application has requested text input through a bottom-line window (a small window that appears at the bottom of the screen) or has turned off the Text Services Manager (TSM).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNNewFontMenuObject

Creates a new Font menu object. (Deprecated in Mac OS X v10.5.)

```
OSStatus TXNNewFontMenuObject (
   MenuRef iFontMenuHandle,
   SInt16 iMenuID,
   SInt16 iStartHierMenuID,
   TXNFontMenuObject *oTXNFontMenuObject
);
```

Parameters

iFontMenuHandle

A value of type `MenuRef` obtained by calling the Menu Manager functions `CreateNewMenu`, `NewMenu`, `CreateMenuFromNib`, or `GetMenu`. Before calling `TXNNewFontMenuObject`, initialize the `menuData` field of the handle to specify the menu title, in Pascal string format. You must make sure the string is localized appropriately.

iMenuID

The menu ID of the Font menu.

iStartHierMenuID

The first MenuID to use if any hierarchical menus need to be created. This function calls `SetMenuItemHierarchicalID` to create hierarchical menus, so the value of this parameter must follow the rules for that function. On systems less than Mac OS 8.5, the submenuID must be less than 255. For systems after Mac OS 8.5, the range can be as large as large 32767. However, it is important to remember that `TXNNewFontMenuObject` only uses `iStartHierMenuID` as a starting ID when adding hierarchical menus. Therefore provide plenty of room to increment this value. For example, on a system less than Mac OS 8.5, start at 175. On systems after than Mac OS 8.5, do not use a value more than 32000.

oTXNFontMenuObject

A pointer to a structure of type `TXNFontMenuObject` (page 2712). On return, a new Font menu object.

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Discussion
A good time to call the `TXNNewFontMenuObject` function is when you are preparing to display your menu bar. This function fills the Font menu with font names. Later, you can pass the Font menu object along with a text object to the `TXNDoFontMenuSelection` function which handles all aspects of user interaction with the Font menu.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNNewObject

Creates a new MLTE text object which is an opaque structure that handles text formatting at the document level. (Deprecated in Mac OS X v10.3. Use `TXNCreateObject` (page 2630) instead.)

Not recommended.

```
OSStatus TXNNewObject (
   const FSSpec *iFileSpec,
   WindowRef iWindow,
   const Rect *iFrame,
   TXNFrameOptions iFrameOptions,
   TXNFrameType iFrameType,
   TXNFileType iFileType,
   TXNPermanentTextEncodingType iPermanentEncoding,
   TXNObject *oTXNObject,
   TXNFrameID *oTXNFrameID,
   TXNObjectRefcon iRefCon
);
```

Parameters

iFileSpec

A pointer to a variable of type `FSSpec`. If you pass `NULL` you start with an empty document. Otherwise, the contents of the file to which `iFileSpec` points are read into the object. The referenced file must consist entirely of data that MLTE can read (`'TEXT'`, `'RTF '`, `'utxt'`, or `'txtn'`). If the referenced file contains your application's private data and data that MLTE can read, you should call the `TXNNewObject` function with the `iFileSpec` parameter set to `NULL`. Once `TXNNewObject` creates the text object, your application can read the private data into the text object by calling the `TXNSetDataFromFile` (page 2685) function.

iWindow

A reference to the window in which the document will be displayed. This parameter can be `NULL`. If it is `NULL`, you must attach a window or graphics port to the text object by using the `TXNAttachObjectToWindow` (page 2620) function.

iFrame

A pointer to a variable of type `Rect`. If you pass `NULL`, the window's `portRect` rectangle is used as the frame. If you do not want to fill the entire window, you use the `iFrame` parameter to specify the area to fill.

iFrameOptions

A value that specifies the options you want the frame to support. See Frame Option Masks (page 2749) for a description of possible values.

If you want to create a read-only document, you need to pass the option `kTXNReadOnlyMask`. Note that this option puts the text object into a state that does not allow user input. However, your application can put data into the text object by calling the function `TXNSetData`. If you want the text object set into a more restrictive read-only state that does not allow user input or your application to put data into the text object programmatically, you need to call the function `TXNSetTXNObjectControls`, passing the tag `kTXNIOPrivilegesTag`. If you choose to set the text object into this restrictive state, you will get an error if you try to call the function `TXNSetData` on the text object. (In this case, you can change the text object to a less restrictive state by calling `TXNSetTXNObjectControls`, passing the tag `kTXNNoUserIOTag`.)

iFrameType

A value that specifies the frame type of the text object. See Supported Frame Types (page 2767) for a description of possible values.

iFileType

A value that specifies the file type of the text object. See Supported File Types (page 2766) for a description of possible values. You should specify the primary file type. If you use the `kTXNTextensionFile` constant, files are saved in a custom format. If you want saved files to be plain text files, you should specify the `kTXNTextFile` constant, then use the `iframeOptions` parameter to specify whether the plain text files should be saved with `kTXNSingleStylePerTextDocumentResType` or `kTXNMultipleStylesPerTextDocumentResType` resources.

iPermanentEncoding

A value that specifies the encoding in which the document is saved. See Text Encoding Preferences (page 2772) for a description of possible values.

oTXNObject

A pointer to a structure of type `TXNObject`. On return, this points to the opaque text object data structure allocated by the function. You need to pass this object to most MLTE functions.

oTXNFrameID

On return, a pointer to the unique ID for the text object's frame. However, in MLTE version 1.1 and earlier, the frame ID is always set to 0.

iRefCon

A value of type `TXNObjectRefcon`. You can define how to use this for your application. You can set this to any value and retrieve it later.

Return Value

A result code. See "MLTE Result Codes" (page 2773).

Discussion

For each document, a new text object is allocated by the `TXNNewObject` function and returned in the `oTXNObject` parameter. The object is allocated only if no errors occur, including errors that may occur when reading a file. If there is an error during the allocation process, MLTE frees the text object.

If you are writing a text editing application, you may want to call the `TXNNewObject` function when the application launches (a new document will be displayed) and whenever the user selects New from the File menu.

Many MLTE functions require you to pass a text object; some functions also require the frame ID supplied back to your application in the `oTXNFrameID` parameter of `TXNNewObject`.

Because of how MLTE uses Carbon events internally, the window in which the document is displayed must have the standard event handlers installed. You can do this in one of the following ways:

- When you create the window, add the attribute `kWindowStandardHandlerAttribute` to the window. See *Inside Mac OS X: Window Manager Reference* for more information.
- Call the Carbon Event Manager function `InstallStandardEventHandler` on the window's event target. See *Inside Mac OS X: Handling Carbon Events* for more information.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.3.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNOffsetToHIPoint

Obtains the local coordinates of the point that corresponds to a specified offset of a text object.

```
OSStatus TXNOffsetToHIPoint (
   TXNObject iTXNObject,
   TXNOffset iOffset,
   HIPoint *oHIPoint
);
```

Parameters

iTXNObject

The text object for which you want to obtain the local coordinates of a point. You can either call the function `TXNCreateObject` to allocate a text object or you can call the function `HITextViewGetTXNObject` (page 2611) to obtain the text object associated with an HITextView.

iOffset

The offset value of the point for which you want to obtain the local coordinates.

oPoint

On output, a pointer to the local coordinates of the point that corresponds to the value of the `iOffset` parameter.

Return Value

A result code. See "MLTE Result Codes" (page 2773).

Discussion

Offsets in MLTE are always character offsets.

Availability

Available in Mac OS X v10.3 and later.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNOffsetToPoint

Gets the local coordinates of the point that corresponds to a specified offset of a text object. (Deprecated in Mac OS X v10.3. Use `TXNOffsetToHIPoint` (page 2670) instead.)

Not recommended.

```
OSStatus TXNOffsetToPoint (
   TXNObject iTXNObject,
   TXNOffset iOffset,
   Point *oPoint
);
```

Parameters

iTXNObject

The text object for which you want to obtain the local coordinates of a point.

iOffset

An offset value.

oPoint
On return, a pointer to the local coordinates of the point that corresponds to the value of the `iOffset` parameter.

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Discussion
Offsets in MLTE are always character offsets.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.3.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNPageSetup

Displays the Page Setup dialog for the current default printer and manages changes, such as reformatting the text, in response to page layout changes.

```
OSStatus TXNPageSetup (
   TXNObject iTXNObject
);
```

Parameters
iTXNObject
The text object for the active document.

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNPaste

Pastes the contents of the private MLTE scrap into the text object.

```
OSStatus TXNPaste (
   TXNObject iTXNObject
);
```

Parameters
iTXNObject
The text object that identifies the current document.

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Discussion
Before you call `TXNPaste`, you can call the `TXNIsScrapPastable` (page 2665) function to determine if the current scrap contains data supported by MLTE.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNPointToOffset

Gets the offset value that corresponds to a point in local coordinates. (Deprecated in Mac OS X v10.3. Use `TXNHIPointToOffset` (page 2661) instead.)

Not recommended.

```
OSStatus TXNPointToOffset (
   TXNObject iTXNObject,
   Point iPoint,
   TXNOffset *oOffset
);
```

Parameters

iTXNObject
The text object for which you want to obtain an offset value.

iPoint
The local coordinates of a point.

oOffset
On return, a pointer to the offset that corresponds to the value of the `iPoint` parameter.

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Discussion
Offsets in MLTE are always character offsets.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.3.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNPrepareFontMenu

Prepares a Font menu for display. (Deprecated in Mac OS X v10.5.)

```
OSStatus TXNPrepareFontMenu (
   TXNObject iTXNObject,
   TXNFontMenuObject iTXNFontMenuObject
);
```

Parameters

iTXNObject

The text object that identifies the document with the Font menu you want to prepare. Pass `NULL` to display an inactive menu (dimmed).

iTXNFontMenuObject

A Font menu object.

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Discussion
You should call the `TXNPrepareFontMenu` function just before your application opens the Font menu for your user. If the text object's current selection is a single font, MLTE places a check mark next to the menu item for that font.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNPrint

Prints the document so it is formatted to fit the page size selected for the printer.

```
OSStatus TXNPrint (
   TXNObject iTXNObject
);
```

Parameters

iTXNObject

The text object that identifies the document you want to print.

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNReadFromCFURL

Reads data from a `CFURLRef` into a TXNObject.

```
OSStatus TXNReadFromCFURL (
   TXNObject iTXNObject,
   TXNOffset iStartOffset,
   TXNOffset iEndOffset,
   CFDictionaryRef iDataOptions,
   CFURLRef iFileURL,
   CFDictionaryRef *oDocumentAttributes
);
```

Parameters

iTXNObject

The text object into which data read from `iFileURL` is to be added.

iStartOffset

The offset in `iTXNObject` at which to start placing data read from `iFileURL`.

iEndOffset

The offset in `iTXNObject` at which to stop placing data read from `iFileURL`.

iDataOptions

Options for reading the data. See Data Option Key Value Constants (page 2733) for a list of the supported options. If this parameter is `NULL`, the data is read in using MLTE's native format.

iFileURL

A `CFURLRef` to the data that is to be added to the text object specified by `inTXNObject`.

oDocumentAttributes

A value of type CFDictionary that, on return, contains the document attributes present in the data stream, if the file format supports them; otherwise this parameter is `NULL`. The native MLTE file format and RTF support embedded document attributes. See Document Attribute Keys (page 2735) for a list of supported attributes. If this parameter is `NULL`, no document attributes are written out.

Return Value

A result code. See "MLTE Result Codes" (page 2773).

Discussion

This function reads data from a file or a special file bundle (directory) into a text object. Offset parameters are used to specify whether the new data is inserted, appended or replaces an existing data range in the text object. Clients can specify the document format and encoding of the data using the `iDataOptions` parameter. This functions also returns the document attributes present in the data stream. Document attributes are supported only for the rich text file formats supported by MLTE, which are RTF and MLTE native file format.

If the caller passes a pointer to a `CFDictionaryRef`, this function returns a reference to a dictionary of attributes, if there is one, that the caller is responsible for releasing. In all other cases, this function sets the reference to `NULL`. Here is some sample code:

```
CFDictionaryRef oDocumentAttributes = NULL;
status = TXNReadFromCFURL (....,&oDocumentAttributes);
if (oDocumentAttributes != NULL) ::CFRelease(oDocumentAttributes);
```

Availability

Available in Mac OS X v10.4 and later.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNRecalcTextLayout

Recalculates the text layout based on new view and destination rectangles.

```
void TXNRecalcTextLayout (
   TXNObject iTXNObject
);
```

Parameters

iTXNObject

The text object whose layout you want to recalculate.

Discussion

You can call the function `TXNRecalcTextLayout` if you call the function `TXNSetRectBounds` (page 2690) with the `iUpdate` parameter set to `false`. `TXNRecalcTextLayout` recalculates the text layout as well as where the scroll bars, if any, should be placed. When you call `TXNRecalcTextLayout`, MLTE generates an update event to redraw the text object.

Availability

Available in Mac OS X v10.1 and later.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNRedo

Redoes the last command.

```
void TXNRedo (
   TXNObject iTXNObject
);
```

Parameters

iTXNObject

The text object for the document you want to examine.

Discussion

If the user undoes an action and then undoes it again, the second undo is the same as a redo.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNRegisterScrollInfoProc

Installs or uninstalls a scrolling callback function on a text object.

```
void TXNRegisterScrollInfoProc (
   TXNObject iTXNObject,
   TXNScrollInfoUPP iTXNScrollInfoUPP,
   SRefCon iRefCon
);
```

Parameters

iTXNObject

The text object on which you want to install a callback to scroll text.

iTXNScrollInfoUPP

A universal procedure pointer to the callback function you want MLTE to call whenever a scroll bar for the text object must be updated.

iRefCon

A 32-bit value that is passed to your callback.

Discussion

You can call the function `TXNRegisterScrollInfoProc` to install a text-scrolling callback on a text object. This is useful if your application draws and handles its own scrolling widgets. Once you register your callback (`TXNScrollInfoUPP`), MLTE invokes your callback each time it is necessary to update the values and maximum values of your scrolling widget. For example when the user presses the Return key to add a new line, MLTE calculates a new maximum value for the text object. Your callback is then called with the newly-calculated maximum value. To turn off your callback call the function `TXNRegisterScrollInfoProc` with a value of `NULL` for the `iTXNScrollInfoUPP` parameter.

Availability

Available in Mac OS X v10.2 and later.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNResizeFrame

Resizes the view and destination rectangles.

```
void TXNResizeFrame (
   TXNObject iTXNObject,
   UInt32 iWidth,
   UInt32 iHeight,
   TXNFrameID iTXNFrameID
);
```

Parameters

iTXNObject

The text object for the current text area.

iWidth

The new width of the view and destination rectangles in pixels.

iHeight

The new height of the view and destination rectangles in pixels.

iTXNFrameID

The frame ID of the frame associated with the view and destination rectangles you want to resize. You obtain the frame ID when you call the `TXNNewObject` (page 2667) function.

Discussion
You need to call the function `TXNSetFrameBounds` (page 2688) if you want to reset the frame bounds.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNRevert

Reverts to the last saved version of a document.

```
OSStatus TXNRevert (
   TXNObject iTXNObject
);
```

Parameters
iTXNObject
The text object for the active document.

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Discussion
You can use the `TXNRevert` function with files that contain only text as well as files that were created using MLTE. If the file was not previously saved, the document reverts to an empty document. To revert to data that is embedded in a private file type, use the `TXNSetSelection` (page 2692) function to select all of the current data and then use the `TXNSetDataFromFile` (page 2685) function to read in the old data.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNSave

Saves the contents of the document as the file type you specify. (Deprecated in Mac OS X v10.4. Use `TXNWriteRangeToCFURL` (page 2699) instead.)

Not recommended.

```
OSStatus TXNSave (
   TXNObject iTXNObject,
   TXNFileType iType,
   OSType iResType,
   TXNPermanentTextEncodingType iPermanentEncoding,
   const FSSpec *iFileSpecification,
   SInt16 iDataReference,
   SInt16 iResourceReference
);
```

Parameters

iTXNObject

The text object for the active document.

iType

The file type to which the text object should be saved. The type must be `kTXNTextensionFile`, `kTXNTextFile`, or `kTXNUnicodeTextData`. See Supported File Types (page 2766) for more information on file type constants.

iResType

The type of resource that should be used to save the style information if the file is being saved as plain text. This parameter is ignored for file types that are not plain text.

iPermanentEncoding

The encoding in which to save the document. If the internal encoding used by MLTE does not match the requested encoding type, the text is translated by the Conversion Manager.

iFileSpecification

A pointer to a variable that specifies the location of the file. This parameter is retained and used in calls to the `TXNRevert` (page 2677) function. It is not retained once the text object is deleted or disposed of.

iDataReference

The data fork reference number of the open file.

iResourceReference

The resource fork reference number of the open file. This parameter is ignored if the file type is not `kTXNTextFile`. You can save text without style information by passing `-1` for this parameter.

Return Value

A result code. See "MLTE Result Codes" (page 2773).

Discussion

You must first open the file to which you want to save the document. If you are saving the file as plain text and the application has specified a resource type in which to save style attributes, then you must also open the file's resource fork.

MLTE does not move the marker before writing the file. You must make sure the file marker of the opened file is at the position where you want data to be written. Typically, this is position 0, but you can specify any valid file position. This behavior lets you write private data, followed by data that is written by MLTE, which can subsequently be followed by more private data or even another MLTE file.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNScroll

Scrolls the text within a view rectangle of the specified text object.

```
OSStatus TXNScroll (
   TXNObject iTXNObject,
   TXNScrollUnit iVerticalScrollUnit,
   TXNScrollUnit iHorizontalScrollUnit,
   long *ioVerticalDelta,
   long *ioHorizontalDelta
);
```

Parameters

iTXNObject

The text object whose text you want to scroll.

iVerticalScrollUnit

Specifies the units to use for the `ioVerticalDelta` parameter. Pass `kTXNScrollUnitsInPixels` to specify pixels, pass `kTXNScrollUnitsInLines` to specify a count of lines, and pass `kTXNScrollUnitsInViewRects` to specify the height of the current view rectangle (`viewRect`). Note that scrolling in line units is the slowest because each line must be measured by MLTE before the text scrolls. See Scroll Units (page 2761) for more information.

iHorizontalScrollUnit

Specifies the units to use for the `ioHorizontalDelta` parameter. Pass `kTXNScrollUnitsInPixels` to specify pixels, pass `kTXNScrollUnitsInLines` to specify a count of lines, and pass `kTXNScrollUnitsInViewRects` to specify the height of the current view rectangle (`viewRect`). Note that scrolling in line units is the slowest because each line must be measured by MLTE before the text scrolls. See Scroll Units (page 2761) for more information.

ioVerticalDelta

On input, the number of units by which to scroll in the vertical direction. You specify the units in the `iVerticalScollUnit` parameter. On output, the number of units actually scrolled in the vertical direction. A positive value indicates to move the text in a downward direction.

ioHorizontalDelta

On input, the number of units by which to scroll in the horizontal direction. You specify the units in the `iHorizontalScrollUnit` parameter. On output, the number of units actually scrolled in the horizontal direction. A positive value indicates to move the text to the right.

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Discussion
The function `TXNScroll` moves the text within the view rectangle of the specified text object by the designated number of units. You can use this function to scroll the text in a text object in response to user input in a control other than the standard scroll bars that MLTE supplies.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
MacTextEditor.h

TXNSelectAll

Selects all data in the frame of a text object.

```
void TXNSelectAll (
   TXNObject iTXNObject
);
```

Parameters

iTXNObject
The text object for the current text area.

Discussion
You can check whether your application should enable the Select All menu item by calling the `TXNDataSize` (page 2632) function to check if the text object contains any data.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
MacTextEditor.h

TXNSetActionNameMapper

Sets a callback that MLTE uses to obtain the localized string representing an action or an action group.

```
OSStatus TXNSetActionNameMapper (
   TXNObject iTXNObject,
   TXNActionNameMapperUPP iStringForKeyProc,
   const void *iUserData
);
```

Parameters

iTXNObject
The text object for which the callback is to be set.

iStringForKeyProc
The callback.

iUserData
A pointer to user-defined data that will help you map the action key to a string.

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Discussion
If you have asked MLTE to handle updating for the Redo and Undo commands in the Edit menu, you should call this function so that MLTE can call your callback, which provides the correct string for each of those commands.

When MLTE's handler for `kEventClassCommand`/`kEventCommandUpdateStatus` is called for the Redo or Undo command, MLTE checks to see if a `TXNActionNameMapperProc` has been installed. If a `TXNActionNameMapperProc` is installed, it is called to get the correct string for updating the menu item. The client can used the action name and the command ID to determine the appropriate string.

For information on a `TXNActionNameMapperProc`, see TXNActionNameMapperProcPtr (page 2702).

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNSetBackground

Sets the background on which the text object's data is drawn.

```
OSStatus TXNSetBackground (
   TXNObject iTXNObject,
   const TXNBackground *iBackgroundInfo
);
```

Parameters

iTXNObject
The text object that identifies the document to be activated.

iBackgroundInfo
A pointer to a structure that describes the background.

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Version Notes
MLTE supports only color as the background.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNSetCommandEventSupport

Enables and disables support for menu commands in MLTE.

```
OSStatus TXNSetCommandEventSupport (
   TXNObject iTXNObject,
   TXNCommandEventSupportOptions iOptions
);
```

Parameters

iTXNObject

The TXNObject.

iOptions

The menu commands for which support is to be enabled or disabled. For possible values, see Command Event Support Options (page 2729).

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Discussion
When `TXNAttachObjectToWindow` (page 2620) or `TXNSetEventTarget` (page 2686) is called to associate an MLTE object with an HIObject that can serve as an event target, most handlers are installed and activated immediately.

However, when the handlers for the kEventClassCommand class are installed, they are not activated. Call this function to activate the handlers for this class, which provide support for the menu commands. This approach means that an application can install handlers on top of these and be sure that enabling or disabling the MLTE handlers does not change the order of the handler chain.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNSetContextualMenuSetup

Provides a callback function that is called before MLTE displays its contextual menu.

```
OSStatus TXNSetContextualMenuSetup (
   TXNObject iTXNObject,
   TXNContextualMenuSetupUPP iMenuSetupProc,
   const void *iUserData
);
```

Parameters

iTXNObject

The text object.

iMenuSetupProc

The callback. For more information, see `NewTXNContextualMenuSetupUPP` (page 2616).

iUserData

A pointer to user-defined data that will be passed to the callback specified by the `iMenuSetupProc` parameter.

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Discussion
The callback function specified by `iMenuSetupProc` is called just before MLTE displays its contextual menu. The menu that is passed to the callback contains MLTE-specific items only. The client items and handlers should be installed each time the callback is called.

When the callback is called, MLTE has selected the word the user clicked with the Option key pressed. For convenience, the TXNObject associated with the callback is passed to the callback as well as the data specified by `iUserData`.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNSetData

Replaces a range of data (text, graphics, and so forth).

```
OSStatus TXNSetData (
   TXNObject iTXNObject,
   TXNDataType iDataType,
   const void *iDataPtr,
   ByteCount iDataSize,
   TXNOffset iStartOffset,
   TXNOffset iEndOffset
);
```

Parameters

iTXNObject
The text object that identifies the document in which you want to replace data.

iDataType
The type of the replacement data. See Supported Data Types (page 2765) for a description of possible values.

iDataPtr
A pointer to the data that will replace the data that is in the range specified by the `iStartOffset` and `iEndOffset` parameters.

iDataSize
The size of the data (in bytes) to which `iDataPtr` points.

iStartOffset
The beginning of the range of data to replace. You can use the `TXNGetSelection` (page 2658) function to get the absolute offsets of the current selection.

iEndOffset
The end of the range to replace. You can use the `TXNGetSelection` (page 2658) function to get the absolute offsets of the current selection. If you want to insert text, the ending and starting offsets should be the same value.

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Discussion
If you have a text object that has word wrap disabled, and you want to avoid horizontal scrolling, you can try the following. After you call the function `TXNSetData`, call `TXNSetSelection` (page 2692) with the value of the ending offset set to what it was before you called `TXNSetData`. Then, call the function `TXNShowSelection` (page 2696) to scroll the text back into view.

Offsets in MLTE are always character offsets.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
QTMetaData

Declared In
`MacTextEditor.h`

TXNSetDataFromCFURLRef

Replaces a range of data with the contents of a file. (Deprecated in Mac OS X v10.4. Use `TXNReadFromCFURL` (page 2673) instead.)

Not recommended.

```
OSStatus TXNSetDataFromCFURLRef (
   TXNObject iTXNObject,
   CFURLRef iURL,
   TXNOffset iStartOffset,
   TXNOffset iEndOffset
);
```

Parameters

iTXNObject

The text object that identifies the document in which you want to replace data. You can either call the function `TXNCreateObject` to allocate a text object or you can call the function `HITextViewGetTXNObject` (page 2611) to obtain the text object associated with an HITextView.

iURL

A reference to the Core Foundation URL that specifies the file which contains the data you want to add to the object.

iStartOffset

The starting position at which to insert the file into the document. If you want to replace the current selection, set the `iStartOffset` parameter to `kTXNUseCurrentSelection`. If you want to replace the entire document, set the `iStartOffset` parameter to `0`. Offsets in MLTE are always character offsets.

iEndOffset

The ending position of the range being replaced by the file. You can use the `TXNGetSelection` function to get the absolute offsets of the current selection. Offsets in MLTE are always character offsets.

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Discussion
Your application must open the file and set the `iStartOffset` parameter to the appropriate value before you call the `TXNSetDataFromCFURLRef` function. If you want to embed MLTE data within private data or other MLTE data, you must set the file position to the appropriate position.

In the (now deprecated) function `TXNNewObject` your could pass a file reference and MLTE supported functionality to revert back to the original file reference. When you call the function `TXNSetDataFromCFURLRef`, MLTE saves the `CFURLRef`. If you change the contents of the text object and then call the function `TXNRevert`, the document reverts to the contents specified by the saved `CFURLRef`.

Availability
Available in Mac OS X v10.3 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNSetDataFromFile

Replaces a range of data with the contents of a file. (Deprecated in Mac OS X v10.3. Use `TXNSetDataFromCFURLRef` (page 2684) instead.)

Not recommended.

```
OSStatus TXNSetDataFromFile (
   TXNObject iTXNObject,
   SInt16 iFileRefNum,
   OSType iFileType,
   ByteCount iFileLength,
   TXNOffset iStartOffset,
   TXNOffset iEndOffset
);
```

Parameters

iTXNObject
The text object that identifies the document in which you want to replace data.

iFileRefNum
The file reference obtained when you opened the file.

iFileType
The file type of the file from which you are getting data. MLTE supports `'RTF '` as well as the file types specified by the constants described in Supported File Types (page 2766).

iFileLength
A value that specifies how much data should be read. This parameter is ignored if the file type is the custom file format (represented by the constant `kTXNTextensionFile`) that MLTE supports. This parameter is useful when your application uses MLTE to read data that is embedded in your application's private file. If you want MLTE to deal with the entire file, set the `iFileLength` parameter to a value of `kTXNEndOffset`.

iStartOffset
The starting position at which to insert the file into the document. You can use the `TXNGetSelection` function to get the absolute offsets of the current selection. If you want to replace the entire document, then set the `iStartOffset` parameter to 0.

iEndOffset
The ending position of the range being replaced by the file. You can use the `TXNGetSelection` function to get the absolute offsets of the current selection.

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Discussion
Your application must open the data fork of the file and set the `iStartOffset` parameter to the appropriate value before you call the `TXNSetDataFromFile` function. If you want to embed MLTE data within private data or other MLTE data, you must set the file position to the appropriate position.

Offsets in MLTE are always character offsets.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.3.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNSetEventTarget

Sets a Carbon Event target for MLTE Carbon Event handlers.

```
OSStatus TXNSetEventTarget (
   TXNObject iTXNObject,
   HIObjectRef iEventTarget
);
```

Parameters
iTXNObject
The TXNObject.

iEventTarget
The `HIObjectRef` that is to be set as the event target for all of the Carbon Event handlers of the TXNObject specified by `iTXNObject`.

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Discussion
The default target for a TXNObject that is not contained in an HITextView is the window of the TXNObject. Call this function when you want to override or set the Carbon Event target for a TXNObject.

Note that if the TXNObject already has a default target when this function is called, the handlers are removed from the old target before the new handlers are installed.

When this function returns, these handlers for Carbon Events are installed and active for the `kEventClassTextInput` class:

- `kEventTextInputUpdateActiveInputArea`
- `kEventTextInputUnicodeForKeyEvent`
- `kEventTextInputUnicodeText`

- kEventTextInputOffsetToPos
- kEventTextInputPosToOffset
- kEventTextInputGetSelectedText

When this function returns, these handlers for Carbon Events are installed and active for the kEventClassTSMDocumentAccess class:

- kEventTSMDocumentAccessGetLength
- kEventTSMDocumentAccessGetSelectedRange
- kEventTSMDocumentAccessGetCharactersPtr
- kEventTSMDocumentAccessGetCharactersPtrForLargestBuffer
- kEventTSMDocumentAccessGetCharacters
- kEventTSMDocumentAccessGetFont
- kEventTSMDocumentAccessGetGlyphInfo

When this function returns, these handlers for Carbon Events are installed and active for the kEventClassFont class:

- kEventFontPanelClosed
- kEventFontSelection

When this function returns, these handlers for Carbon Events are installed and inactive by default for the kEventClassCommand class:

- kEventProcessCommand
- kEventCommandUpdateStatus

The kEventClassCommand handlers support the following commands:

- kHICommandUndo
- kHICommandRedo
- kHICommandSelectAll
- kHICommandCut
- kHICommandCopy
- kHICommandPaste
- kHICommandClear
- kHICommandShowSpellingPanel
- kHICommandCheckSpelling
- kHICommandChangeSpelling
- kHICommandCheckSpellingAsYouType
- kHICommandIgnoreSpelling

- `kHICommandLearnWord`

Activate command support by calling `TXNSetCommandEventSupport` (page 2681) with the appropriate options.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNSetFontDefaults

Specifies the font descriptions for each script used in a text object. (Deprecated in Mac OS X v10.4.)

```
OSStatus TXNSetFontDefaults (
   TXNObject iTXNObject,
   ItemCount iCount,
   const TXNMacOSPreferredFontDescription iFontDefaults[]
);
```

Parameters

iTXNObject
The text object for the document whose fonts you want to specify.

iCount
The number of font descriptions in the `iFontDefaults` array.

iFontDefaults
An array of `TXNMacOSPreferredFontDescription` structures that contain the font description you want to use for each script.

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNSetFrameBounds

Changes the boundaries of a text object's frame.

```
void TXNSetFrameBounds (
   TXNObject iTXNObject,
   SInt32 iTop,
   SInt32 iLeft,
   SInt32 iBottom,
   SInt32 iRight,
   TXNFrameID iTXNFrameID
);
```

Parameters

iTXNObject
The text object for the current text area.

iTop
The top boundary of the rectangle.

iLeft
The left boundary of the rectangle.

iBottom
The bottom boundary of the rectangle.

iRight
The right boundary of the rectangle.

iTXNFrameID
The frame ID of the frame you want to move. You obtain a frame ID when you call the `TXNNewObject` (page 2667) function.

Discussion
If you want to change the view and destination rectangles, you should call the `TXNResizeFrame` (page 2676) function.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNSetHIRectBounds

Sets the view rectangle and/or the destination rectangle.

```
void TXNSetHIRectBounds (
   TXNObject iTXNObject,
   const HIRect *iViewRect,
   const HIRect *iDestinationRect,
   Boolean iUpdate
);
```

Parameters

iTXNObject
The text object for the current text area. You can call the function `TXNCreateObject` to allocate a text object.

iViewRect

A pointer to rectangle that contains the new coordinates for the view rectangle. If you do not want to change the view rectangle pass `NULL`. You cannot set the view rectangle for text objects into an HITextView, you can only set the destination rectangle.

iDestinationRect

A pointer to a rectangle that contains the new coordinates for the destination rectangle. If you do not want to change the destination rectangle pass `NULL`.

iUpdate

A value that specifies whether you want the text and scroll bars recalculated and redrawn. Pass `true` to recalculate and redraw; otherwise pass `false`. You must pass `false` for text objects into an HITextView.

Discussion

The view rectangle controls the text you see. The destination rectangle controls how text is laid out. You can specify coordinates for one or both rectangles. Scroll bars are drawn inside the view rectangle.

Availability

Available in Mac OS X v10.3 and later.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNSetRectBounds

Set the view rectangle and/or the destination rectangle. (Deprecated in Mac OS X v10.3. Use `TXNSetHIRectBounds` (page 2689) instead.)

Not recommended.

```
void TXNSetRectBounds (
   TXNObject iTXNObject,
   const Rect *iViewRect,
   const TXNLongRect *iDestinationRect,
   Boolean iUpdate
);
```

Parameters

iTXNObject

The text object for the current text area.

iViewRect

A pointer to a `Rect` data structure that contains the new coordinates for the view rectangle. If you do not want to change the view rectangle pass `NULL`.

iDestinationRect

A pointer to a `TXNLongRect` data structure that contains the new coordinates for the destination rectangle. If you do not want to change the destination rectangle pass `NULL`.

iUpdate

Pass `true` if you want the text and location of the scroll bars recalculated and redrawn; otherwise pass `false`.

Discussion
You can specify coordinates for one or both rectangles. The view rectangle controls the text you see. The destination rectangle controls how text is laid out. Scroll bars are drawn inside the view rectangle.

If you set the `iViewRect` parameter to a location not currently represented by the scroll bar and you pass `NULL` for the *`iDestinationRect`* parameter, it becomes impossible to scroll to the left bounds of the destination rectangle. If you want to position the view rectangle inside the destination rectangle, you should supply a custom scrolling callback. See `TXNScrollInfoProcPtr` (page 2705).

Availability
Available in Mac OS X v10.1 and later.
Deprecated in Mac OS X v10.3.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNSetScrollbarState

Sets the state of the scroll bars so they are drawn correctly in response to activate events.

```
OSStatus TXNSetScrollbarState (
   TXNObject iTXNObject,
   TXNScrollBarState iActiveState
);
```

Parameters

`iTXNObject`
The text object that identifies the document you want activated. You can call the function `TXNCreateObject` to allocate a text object.

`iActiveState`
A value that indicates the state of the scroll bars. See Scroll Bar States (page 2760) for a description of possible values. If you pass the `kScrollBarsAlwaysActive` constant, the scroll bars are always active, whether or not the frame text area currently has keyboard focus. Passing `kScrollBarsAlwaysActive` can be useful for a window such as a dialog that may contain multiple text areas, each of which may have a scrollable frame. If you pass `kScrollBarsSyncWithFocus`, MLTE synchronizes the activity state of the scroll bars with the focus state of the frame. Therefore, only when the frame has keyboard focus does it have active scroll bars. A value of `kScrollBarsSyncWithFocus` is the default and is typically recommended if you have only one frame per window.

Return Value
A result code. See "MLTE Result Codes" (page 2773). This function returns an error if the text object is in an HITextView.

Discussion
This function is a macro that calls the function `TXNActivate` with the `TXNFrameID` parameter set to `0`. You typically call `TXNSetScrollbarState` in response to an activate event. If the text object was previously inactive, `TXNSetScrollbarState` removes any visual indication of its prior inactive state (such as a dimmed or framed selection area or inactive scroll bars). Before you call the `TXNSetScrollbarState` function, you should make sure that the window belongs to your application.

The `TXNSetScrollbarState` function does not change the keyboard focus. This means your application can have a text area that is not focused, but in which the scroll bars are active. This lets application users scroll the inactive text without changing the focus from another text area.

If you want to display a text area that has both keyboard focus and active scroll bars, you must call the `TXNFocus` (page 2644) function immediately before you call the `TXNSetScrollbarState` function. Note that MLTE does not retain information about keyboard focus. So if, for example, you set the keyboard focus on a text area and the window containing the text area becomes deactivated, you must call the `TXNFocus` (page 2644) function when the window becomes activated again.

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNSetSelection

Specifies the selection range or the position of the insertion point.

```
OSStatus TXNSetSelection (
   TXNObject iTXNObject,
   TXNOffset iStartOffset,
   TXNOffset iEndOffset
);
```

Parameters

iTXNObject
The text object that identifies the document for which you want to set the selection range or insertion point position.

iStartOffset
The new starting offset. Offset values are character offsets.

iEndOffset
The new ending offset. Offset values are character offsets.

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Discussion
You can use the `TXNSetSelection` function to highlight an initial default value in a document, such as a data-entry form, or to position the insertion point at the start of the field where you want the user to enter a value. To position the insertion point, specify the same value for the `iStartOffset` and `iEndOffset` parameters.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNSetSpellCheckAsYouType

Enables and disables the "Spell Check as You Type" feature.

```
OSStatus TXNSetSpellCheckAsYouType (
   TXNObject iTXNObject,
   Boolean iActivate
);
```

Parameters

iTXNObject

The text object for which the "Spell Check as You Type" feature is to be enabled or disabled.

iActivate

A Boolean whose value is `true` to enable the "Spell Check as You Type" feature or `false` to disable it.

Return Value

A result code. See "MLTE Result Codes" (page 2773). If `TXNSetCommandEventSupport` (page 2681) has not been called to enable the event handler for spell checking, an error is returned.

Availability

Available in Mac OS X v10.4 and later.

Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNSetTXNObjectControls

Sets formatting and privileges attributes (such as justification, line direction, tab values, and read-only status) that apply to the entire text object.

```
OSStatus TXNSetTXNObjectControls (
   TXNObject iTXNObject,
   Boolean iClearAll,
   ItemCount iControlCount,
   const TXNControlTag iControlTags[],
   const TXNControlData iControlData[]
);
```

Parameters

iTXNObject

The text object that identifies the document for which you want to set formatting and privileges attributes.

iClearAll

A `Boolean` value. If you set this to `true`, all formatting and privileges attributes are reset to their default value. That is, `true` clears existing tags and resets each to its default value.

iControlCount

The number of items in the `iControlTags` array.

iControlTags

An array of values that specifies kind of data that is passed in the `iControlData` parameter. See Formatting and Privileges Settings (page 2742) for a description of possible values. On systems that use Apple Type Services for Unicode Imaging (ATSUI), you can also pass any of the following ATSUI attribute tag constants:

- `kATSULineDirectionTag`
- `kATSULineJustificationFactorTag`
- `kATSULineFlushFactorTag`
- `kATSULineBaselineValuesTag`
- `kATSULineLayoutOptionsTag`
- `kATSUCGContextTag`

See the ATSUI documentation for a description of these ATSUI constants.

iControlData

An array of `TXNControlData` (page 2711) unions that contain the information your application wants to set. The value you supply to the `iControlTags` parameter specifies how the union of type `TXNControlData` is treated. You must make sure that the value you assign to the `iControlData` parameter is the appropriate type implied by the value you passed in the `iControlTags` parameter.

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Discussion
On systems that use Apple Type Services for Unicode Imaging (ATSUI), the ATSUI line control attribute tags can be passed to this function in the `iControlTag` parameter. This is the case for all the ATSUI tags except `kATSULineRotationTag`. ATSUI tags are applied to the entire text object.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNSetTypeAttributes

Sets text attributes (such as size and style) for the current selection or the text defined by a range you specify.

```
OSStatus TXNSetTypeAttributes (
   TXNObject iTXNObject,
   ItemCount iAttrCount,
   const TXNTypeAttributes iAttributes[],
   TXNOffset iStartOffset,
   TXNOffset iEndOffset
);
```

Parameters

iTXNObject

The text object that contains the current selection.

iAttrCount

The number of font attributes in the `iAttributes` array.

iAttributes

An array of `TXNTypeAttributes` structures in which you specify the attributes you want to set. Values passed in the `iAttributes` array that are less than or equal to `sizeof(UInt32)` are passed by value. Values greater than `sizeof(UInt32)` are passed as a pointer. That is, the third field of the `TXNTypeAttributes` structure is a union that serves as either a 32-bit integer or a 32-bit pointer.

iStartOffset

The starting offset at which you want the application to begin setting attributes. If you want to use the current selection, set `iStartOffset` to `kTXNUseCurrentSelection`.

iEndOffset

The offset at which you want the application to stop setting attributes. If you want to use the current selection, set `iEndOffset` to `kTXNUseCurrentSelection`.

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Discussion
You can use this function to clear ATSUI font features and ATSUI font variations. To clear either the features or the variations, you must `OR` the `kTXNClearTheseFontFeatures` constant with the current ATSUI font feature or font variation setting. See Clearance Settings (page 2729) for more information.

Offsets in MLTE are always character offsets.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNSetViewRect

Sets the rectangle that describes the current view into the document; changes the amount of text that is viewable. (Deprecated in Mac OS X v10.2. Use `TXNSetFrameBounds` (page 2688) or `TXNSetRectBounds` (page 2690) instead.)

Not recommended.

```
void TXNSetViewRect (
   TXNObject iTXNObject,
   const Rect *iViewRect
);
```

Parameters

iTXNObject

The text object for the current text area.

iViewRect

On input, points to a rectangle that describes the new view of the document.

Discussion
The `TXNSetViewRect` function does not change where a line of text wraps. Line wrapping is controlled by the `TXNSetFrameBounds` (page 2688) function.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.2.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNShowSelection

Scrolls the current selection into view.

```
void TXNShowSelection (
   TXNObject iTXNObject,
   Boolean iShowEnd
);
```

Parameters

iTXNObject
The text object for the current text area.

iShowEnd
A `Boolean` value. If you set this to `true`, the end of the selection is scrolled into view. Otherwise, the beginning of the selection is scrolled into view.

Discussion
You can use this to scroll text into view after you have called the `TXNFind` (page 2641) function and the matching text is not in the current view of the text object.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNTerminateTextension

Closes the MLTE library. (Deprecated in Mac OS X v10.3. This function is no longer needed.)

Not recommended.

```
void TXNTerminateTextension (
   void
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.3.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNTSMCheck

Checks to see if the Text Services Manager (TSM) is active.

Not Supported

```
Boolean TXNTSMCheck (
    TXNObject iTXNObject,
    EventRecord *iEvent
);
```

Parameters

iTXNObject

The text object that identifies the current document. Pass `NULL` when there is no active text object and you want to check for TSM activity.

iEvent

A pointer to an event record structure. This can be `NULL`.

Return Value

A `Boolean` value. It is `true` if TSM is active and `false` if TSM is not active.

Discussion

The purpose of this function is to ensure input methods have enough time to respond. Call this when the `WaitNextEvent` function returns `false` or there is no active text object. For a Unicode application, if the event is a key-down event, `TXNTSMCheck` will also process the event and set the event to `NULL`.

Declared In

`MacTextEditor.h`

TXNUndo

Undoes the last command.

```
void TXNUndo (
   TXNObject iTXNObject
);
```

Parameters

iTXNObject

The text object for the document you want to examine.

Discussion

The undo stack is 32 levels deep. That is, undoable actions are tracked until the total count is 32. If a user undoes two actions, the Redo command must be used twice to get back to the original state. If more than 32 actions are performed, the oldest actions are forgotten as each new action takes place.

If the user performs a new action after choosing Redo from the Edit menu, the redone action is no longer available to be undone. For example, a user performs the following actions: types some text, cuts some text, pastes some text, types some text; undoes the last typing action, and undoes the paste operation; redoes the paste; types some new text. After the new text has been typed, the undo stack contains the first text that was typed, the cut action, and the new text that was just typed.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
MacTextEditor.h

TXNUpdate

Redraws everything in a frame in response to an update event.

```
void TXNUpdate (
   TXNObject iTXNObject
);
```

Parameters

iTXNObject

The text object that identifies the document to be updated.

Discussion
This function calls the Window Manager `BeginUpdate` and `EndUpdate` functions for the window that you pass to the `TXNNewObject` function. You shouldn't use it for windows that contain something else besides the text object. If the window contains something in addition to the text object, you should use the `TXNDraw` (page 2636) function instead of the `TXNUpdate` function.

Before you call `TXNUpdate`, you should make sure that the window belongs to your application. The `TXNUpdate` function redraws any content that needs updating regardless of the layer in which your window is located.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
MacTextEditor.h

TXNVersionInformation

Gets the version number of MLTE and the set of features in this version.

```
TXNVersionValue TXNVersionInformation (
   TXNFeatureBits *oFeatureFlags
);
```

Parameters

oFeatureFlags

On return, a pointer to a value that indicates the set of features in use in this version. See Frame Option Bits (page 2746) and ATSUI Feature Masks (page 2726) for a description of possible values.

Return Value
A value that specifies the version of MLTE. See the description of the `TXNVersionValue` data type.

Discussion
If the bit `kTXNWillDefaultToATSUIBit` is set, then by default MLTE uses ATSUI to image and measure text and uses Unicode to store characters. If the bit `kTXNWillDefaultToCarbonEventBit` is set, then, by default, MLTE uses Carbon events and Apple events are not supported.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNWriteRangeToCFURL

Writes a range of a text object to a file or to a special file bundle.

```
OSStatus TXNWriteRangeToCFURL (
   TXNObject iTXNObject,
   TXNOffset iStartOffset,
   TXNOffset iEndOffset,
   CFDictionaryRef iDataOptions,
   CFDictionaryRef iDocumentAttributes,
   CFURLRef iFileURL
);
```

Parameters

iTXNObject
The text object having a range that is to be written.

iStartOffset
The offset in `iTXNObject` at which to start writing data to `iFileURL`.

iEndOffset
The offset in `iTXNObject` at which to stop writing data to `iFileURL`.

iDataOptions
A CFDictionaryRef that specifies options for writing out the data. See Data Option Key Value Constants (page 2733) for a list of the supported options. If this parameter is `NULL`, the data is written out using MLTE's native format.

iDocumentAttributes
The document attributes that are to be embedded in the data stream. This parameter is only supported when writing out the data using one of the following formats: RTF and MLTE native format. Only the key / values defined in Document Attribute Keys (page 2735) are written out. The content of the dictionary is ignored for any other format. If the dictionary is `NULL`, no attributes are added to the data stream.

iFileURL
`CFURLRef` for an existing file or directory, whichever is appropriate for the file type. On return, `iFileURL` contains a copy of the data in the given range for the `iTXNObject` with the format and encoding specified by `iDataOptions`.

Return Value
A result code. See "MLTE Result Codes" (page 2773).

Discussion
This function writes a range of a text object to a file or a special file bundle (directory). It supports several document formats and encodings, which can be specified in the data options dictionary. Clients can specify additional document attributes when data is written out using a file format, such as RTF and native MLTE file format) that supports such attributes.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNZoomWindow

Increases the size of the data displayed in a window in response to a click in the zoom box.

```
void TXNZoomWindow (
   TXNObject iTXNObject,
   SInt16 iPart
);
```

Parameters

iTXNObject
The text object for the current text area.

iPart
The location, in global coordinates, of the cursor at the time the user pressed the mouse button. You obtain this value from the Window Manager function `FindWindow`.

Discussion
Before you call the `TXNZoomWindow` function, you should make sure that the window belongs to your application. You should use the `TXNZoomWindow` function only when a text object has a viewable area that occupies the entire window; for example, if you passed `NULL` for the `iFrame` parameter when you called the `TXNNewObject` (page 2667) function to create the text object. You cannot use `TXNZoomWindow` if the text object is contained in a subframe of a window.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

Callbacks

TXNActionKeyMapperProcPtr

Defines a pointer to an action key mapping function that customizes the Redo and Undo menu items with the specific action that can be redone or undone. (**Deprecated.** Use `TXNActionNameMapperProcPtr` (page 2702) instead.)

```
typedef CFStringRef(* TXNActionKeyMapperProcPtr)
(
    TXNActionKey actionKey,
    UInt32 commandID
);
```

You would declare your function like this if you were to name it `MyTXNActionKeyMapperFunction`:

```
CFStringRef MyTXNActionKeyMapperCallback
(
    TXNActionKey actionKey,
    UInt32 commandID
);
```

Parameters

actionKey

A value of type `TXNActionKey` that indicates an editing action taken by the user.

commandID

The command ID of menu command chosen by the user.

Return Value

The localized string you want to map to the `TXNActionKey`.

Discussion

Your callback is invoked each time you ask MLTE to handle command updates for the Edit menu. MLTE calls your callback whenever it receives a Carbon event of class `kEventClassCommand` and event kind `kEventComandUpdateStatus` for the command ID `kHICommandUndo`. In other words, whenever the undo-command item in the menu needs to be updated. Your callback should examine the `actionKey` parameter and return an appropriately localized `CFStringRef` that describes the undo action. MLTE calls the function `SetMenuItemWithCFString` to update the menu item's text. You are responsible for releasing the `CFString`; MLTE does not call the function `CFRelease` on the string.

You provide a pointer to your action key mapping callback function when you build a dictionary to support Carbon events in MLTE. The pointer should be the value associated with the dictionary key `kTXNActionKeyMapperKey`. You then assign the dictionary to a `TXNCarbonEventInfo` (page 2709) data structure. You treat the data structure as an object control. That is, you associate the `TXNCarbonEventInfo` data structure with a text object by calling the function `TXNSetTXNObjectControls` (page 2693), supplying `kTXNUseCarbonEvents` in the `iControlsTags` parameter and the `TXNCarbonEventInfo` data structure in the `iControlData` parameter.

To provide a pointer to your action key mapping callback function, you use the `NewTXNActionKeyMapperUPP` function to create a universal procedure pointer (UPP) of type `TXNActionKeyMapperUPP`. You can do so with code similar to the following:

```
TXNActionKeyMapperUPP MyTXNActionKeyMapperUPP;
MyTXNActionKeyMapperUPP = NewTXNActionKeyMapperUPP
                        (&MyActionKeyMapperFunction);
```

When you are finished with your action key mapper callback function, you should use the `DisposeTXNActionKeyMapperUPP` function to dispose of the UPP associated with it. However, if you plan to use the same action key mapper callback later in your application, you can reuse the same UPP, rather than dispose of it and later create a new UPP.

Availability

Available in Mac OS X v10.0 and later.

Declared In
MacTextEditor.h

TXNActionNameMapperProcPtr

Defines a pointer to an action name mapping function that customizes the Redo and Undo menu items with the specific action that can be redone or undone.

```
typedef CFStringRef TXNActionNameMapperProcPtr (
      CFStringRef actionName,
      UInt32 commandID,
      void * inUserData
);
```

If you name your function `MyTXNActionNameMapperProc`, you would declare it like this:

```
CFStringRef MyTXNActionNameMapperProc (
      CFStringRef actionName,
      UInt32 commandID,
      void * inUserData
);
```

Parameters

actionName
The name of the action.

commandID
The command ID of the menu command.

inUserData
User-defined data that was provided in the inUserData parameter when `InvokeTXNActionNameMapperUPP` (page 2613) was called.

Availability
Available in Mac OS X v10.4 and later.

Declared In
MacTextEditor.h

TXNContextualMenuSetupProcPtr

Defines a pointer to a contextual menu setup function.

```
typedef void TXNContextualMenuSetupProcPtr (
      MenuRef iContextualMenu,
      TXNObject object,
      void * inUserData
);
```

If you name your function `MyTXNContextualMenuSetupProc`, you would declare it like this:

```
void MyTXNContextualMenuSetupProc (
      MenuRef iContextualMenu,
      TXNObject object,
      void * inUserData
```

```
);
```

Parameters

iContextualMenu

The MLTE contextual menu.

TXNObject

The TXNObject for which `MyTXNContextualMenuSetupProc` was called.

inUserData

User-defined data that was passed to `TXNSetContextualMenuSetup` (page 2682).

Availability
Available in Mac OS X v10.4 and later.

Declared In
`MacTextEditor.h`

TXNFindProcPtr

Defines a pointer to a find function that customizes a search tailored to your application's needs.

```
typedef OSStatus (*TXNFindProcPtr) (
    const TXNMatchTextRecord * matchData,
    TXNDataType iDataType,
    TXNMatchOptions iMatchOptions,
    const void * iSearchTextPtr,
    TextEncoding encoding,
    TXNOffset absStartOffset,
    ByteCount searchTextLength,
    TXNOffset * oStartMatch,
    TXNOffset * oEndMatch,
    Boolean * ofound,
    UInt32 refCon
);
```

If you name your function `MyTXNFindProc`, you would declare it like this:

```
OSStatus TXNFindProcPtr (
    const TXNMatchTextRecord * matchData,
    TXNDataType iDataType,
    TXNMatchOptions iMatchOptions,
    const void * iSearchTextPtr,
    TextEncoding encoding,
    TXNOffset absStartOffset,
    ByteCount searchTextLength,
    TXNOffset * oStartMatch,
    TXNOffset * oEndMatch,
    Boolean * ofound,
    UInt32 refCon
);
```

Parameters

matchData

A pointer to a `TXNMatchTextRecord` structure containing the text to match, the length of that text, and the text's encoding. Pass `NULL` if you are looking for a graphics, sound, or movie object.

iDataType

The type of data for which you want to search. See Supported Data Types (page 2765) for a description of possible values.

iMatchOptions

A value that specifies the matching rules to use in the find operation. See Search Criteria Masks (page 2762) for a description of possible values.

iSearchTextPtr

A pointer to the text to search.

encoding

The encoding of the text to search.

absStartOffset

The offset at which the search should begin. The constant `kTXNStartOffset` specifies the start of the object's data.

searchTextLength

The length, in bytes, of the text to search.

oStartMatch

On return, a pointer to the absolute offset that identifies the start of the match. Your function should set this to `kTXNUseCurrentSelection` if there is no match.

oEndMatch

On return, a pointer to the absolute offset that identifies the end of the match. Your function should set this to `kTXNUseCurrentSelection` if there is no match.

ofound

On return, a pointer to a `Boolean` value; `true` if a match is found.

refCon

An unsigned 32-bit integer your application can use as needed.

Return Value

A result code. See "MLTE Result Codes" (page 2773).

Discussion

You pass a pointer to your find callback function as a parameter to the `TXNFind` (page 2641) function. To provide a pointer to your find callback function, you use the `NewTXNFindUPP` (page 2617) function to create a universal procedure pointer (UPP) of type `TXNFindUPP`. You can do so with code similar to the following:

```
 TXNFindUPP MyTXNFindUPP;
MyTXNFindUPP = NewTXNFindUPP (&MyFindCallback)
```

When you are finished with your find callback function, you should use the `DisposeTXNFindUPP` (page 2609) function to dispose of the UPP associated with it. However, if you plan to use the same find callback function in subsequent calls to the `TXNFind` function, you can reuse the same UPP, rather than dispose of it and later create a new UPP.

Availability

Available in Mac OS X v10.0 and later.

Declared In
MacTextEditor.h

TXNScrollInfoProcPtr

Defines a pointer to a customized scrolling function.

```
typedef void TXNScrollInfoProcPtr (
      SInt32 iValue,
      SInt32 iMaximumValue,
      TXNScrollBarOrientation iScrollBarOrientation,
      SInt32 iRefCon
);
```

If you name your function `MyTXNScrollInfoProc`, you would declare it like this:

```
void MyTXNScrollInfoProc (
      SInt32 iValue,
      SInt32 iMaximumValue,
      TXNScrollBarOrientation iScrollBarOrientation,
      SInt32 iRefCon
);
```

Parameters

iValue
The scroll bar value.

iMaximumValue
The scroll bar maximum value.

iScrollBarOrientation
The orientation of the scroll bar. See Scroll Bar Orientation (page 2760) for more information.

iRefCon
An unsigned 32-bit integer your application can use as needed. This is set and used by your application as needed.

Discussion
You can create a callback to handle and draw your own scroll bar. MLTE calls your callback each time the scroll bar value and the maximum value of the scroll bar needs to be updated. You install a text-scrolling callback on an MLTE text object by calling the function `TXNRegisterScrollInfoProc` (page 2675). When you no longer want the text-scrolling callback invoked on the text object, you can call the function `TXNRegisterScrollInfoProc` with the `iTXNScrollInfoUPP` set to `NULL`.

Availability
Available in Mac OS X v10.2 and later.

Declared In
MacTextEditor.h

Data Types

TXNActionNameMapperUPP

Defines a universal procedure pointer to an action name mapper callback function.

```
typedef TXNScrollActionNameProcPtr TXNActionNameMapperUPP;
```

Discussion
See `TXNActionNameMapperProcPtr` (page 2702) for more information.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`MacTextEditor.h`

TXNActionKeyMapperUPP

Defines a universal procedure pointer to an action key mapping callback function.

```
typedef TXNActionKeyMapperProcPtr TXNActionKeyMapperUPP;
```

Discussion
See `TXNActionKeyMapperProcPtr` (page 2700) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacTextEditor.h`

TXNATSUIFeatures

Contains information about ATSUI font features.

```
struct TXNATSUIFeatures {
    ItemCount featureCount;
    ATSUFontFeatureType * featureTypes;
    ATSUFontFeatureSelector * featureSelectors;
};
typedef struct TXNATSUIFeatures TXNATSUIFeatures;
```

Fields

`featureCount`
The number of features described in this structure.

`featureTypes`
A pointer to a variable of type `ATSUFontFeatureType`. The `ATSUFontFeatureType` type represents the attributes of a particular font feature, such as the presence of ligatures in a font.

`featureSelectors`
: A pointer to a variable of type `ATSUFontFeatureSelector`. The `ATSUFontFeatureSelector` type represents the state of a feature (on or off).

Discussion
Used in the `TXNAttributeData` (page 2707) union.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacTextEditor.h`

TXNATSUIVariations

Contains information about ATSUI variations.

```
struct TXNATSUIVariations {
    ItemCount variationCount;
    ATSUFontVariationAxis * variationAxis;
    ATSUFontVariationValue * variationValues;
};
typedef struct TXNATSUIVariations TXNATSUIVariations;
```

Fields
`variationCount`
: The number of variables described in this structure.

`variationAxis`
: A pointer to a variable of type `ATSUFontVariationAxis`. The `ATSUFontVariationAxis` type represents a stylistic attribute and the range of values used to express this attribute for a font.

`variationValues`
: A pointer to a variable of type `ATSUFontVariationValue`. The `ATSUFontVariationValue` type represents the range of values that a font can use for a particular font variation.

Discussion
Used in the `TXNAttributeData` (page 2707) union.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacTextEditor.h`

TXNAttributeData

Contains information about text attributes in a text object.

```
union TXNAttributeData {
    void * dataPtr;
    UInt32 dataValue;
    TXNATSUIFeatures *atsuFeatures;
    TXNATSUIVariations *atsuVariations;
    CFURLRef urlReference;
};
typedef union TXNAttributeData TXNAttributeData;
```

Fields

`dataPtr`
A pointer to attribute data. For example, a pointer to a font name.

`dataValue`
A value that specifies a text attribute. For example, a value that specifies a font style.

`atsuFeatures`
A pointer to a `TXNATSUIFeatures` (page 2706) structure. For example, a structure that contains information about the ligatures in a font.

`atsuVariations`
A pointer to a `TXNATSUIVariations` (page 2707) structure. For example, a structure that contains information about the range of values used to express a particular stylistic attribute.

`urlReference`
A URL that specifies the location of attribute data.

Discussion
Used in the `TXNTypeAttributes` (page 2718) structure. The data contained in the union is determined by the value in the `size` field of the `TXNTypeAttributes` structure.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacTextEditor.h`

TXNBackground

Specifies the background for text and other data in a text object.

```
struct TXNBackground {
    TXNBackgroundType bgType;
    TXNBackgroundData bg;
};
typedef struct TXNBackground TXNBackground;
```

Fields

`bgType`
Defines the type of data. See `TXNBackgroundType` data type.

`bg`
Specifies the data MLTE should use as a background.

Discussion
Used in the `TXNSetBackground` (page 2681) function.

Version Notes
MLTE 1.1 and earlier supports only color.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacTextEditor.h`

TXNBackgroundData

Represents background data used in the `TXNBackground` structure.

```
union TXNBackgroundData {
    RGBColor color;
};
typedef union TXNBackgroundData TXNBackgroundData;
```

Fields

`color`
A value that specifies the background color on which data is displayed. Color is the only background data that is currently supported.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacTextEditor.h`

TXNCarbonEventInfo

Contains information needed for MLTE to support Carbon events.

```
struct TXNCarbonEventInfo {
    Boolean useCarbonEvents;
    UInt8 filler;
    UInt16 flags;
    CFDictionaryRef fDictionary;
};
typedef struct TXNCarbonEventInfo TXNCarbonEventInfo;
```

Fields

`useCarbonEvents`
Pass `true` to specify a Carbon event, otherwise Apple events will be used by the system to handle MLTE events.

`filler`
Padding you don't need to pass anything.

`flags`
There are currently two flags defined: `kTXNNoAppleEventHandlersMask` and `kTXNRestartAppleEventHandlersMask`. When you request Carbon event support for text input events, it's best to specify the `kTXNNoAppleEventHandlersMask` mask.

`fDictionary`

A reference to a Core Foundation dictionary. (A dictionary is a collection of key-value pairs.) For MLTE to support Carbon events, you need to build a dictionary whose keys are strings that represent the events you want handled (such as "TextInput" or "WindowResize") and whose values are event target references associated with the events. See "Dictionary Keys" for a list of the predefined keys you can use to build the dictionary.

Discussion

You set up MLTE to support Carbon events by passing the `TXNCarbonEventInfo` structure when you call the function `TXNSetTXNObjectControls` (page 2693). Note that MLTE does not handle subclass Carbon events dispatched by the standard handler unless you install the standard handler on the window by calling the Carbon Event Manager function `InstallStandardEventHandler` or by writing code that performs the tasks done by `InstallStandardEventHandler`.

MLTE supports four classes of Carbon events:

- `kEventClassTextInput`
- `kEventClassWindow`
- `kEventClassCommand`
- `kEventClassMenu`

The event kinds supported within each class are listed in Table 36-1. You don't need to specify event kinds to MLTE; the table is provided so you can see the kind of events that MLTE supports. When you set up support for Carbon events, MLTE handles all the calls to the Carbon Event Manager that actually install and set up the handlers that take care of the Carbon events for a text object. See *Inside Mac OS X: Handling Carbon Events* and the *Carbon Event Manager Reference* for more information on Carbon events and for the most recent list of event classes and kinds supported by MLTE.

Table 36-1 Event classes and kinds supported by MLTE

Event Class	Event Kind	Means
`kEventClassTextInput`	`kEventUnicodeForKeyEvent`	Text characters produced by a keyboard
	`kEventOffsetToPos`	Map character index to screen position
	`kEventPosToOffset`	Map screen position to character index
	`kEventGetSelectedText`	Determine currently selected text
	`kEventUpdateActiveInputArea`	Manage contents of an inline input session
`kEventClassWindow`	`kEventWindowActivated`	Window activated (brought to front)
	`kEventWindowDeactivated`	Window deactivated (sent behind)
	`kEventWindowDrawContent`	Draw window's contents on screen
	`kEventWindowClickContentRgn`	Mouse click in content region
`kEventClassCommand`	`kEventCommandProcess`	Menu item chosen or a control with a command has been pressed

	kEventCommandUpdateStatus	Determine enabled or disabled status of command
kEventClassMenu	kEventMenuEnableItems	Font selected from a Font menu

Availability
Available in Mac OS X v10.0 and later.

Declared In
MacTextEditor.h

TXNContextualMenuSetupUPP

Defines a universal procedure pointer to a contextual menu setup callback function.

```
typedef TXNContextualMenuSetupProcPtr TXNContextualMenuSetupUPP;
```

Discussion
See TXNContextualMenuSetupProcPtr (page 2702) for more information.

Availability
Available in Mac OS X v10.4 and later.

Declared In
MacTextEditor.h

TXNControlData

Contains information about formatting and other settings that control how a text object is displayed or behaves.

```
union TXNControlData {
    UInt32 uValue;
    SInt32 sValue;
    TXNTab tabValue;
    TXNMargins *marginsPtr;
};
typedef union TXNControlData TXNControlData;
```

Fields

uValue
: A control setting. You should use this field for control settings, except tab and margins, whose value is unsigned.

sValue
: A control setting. You should use this only for control settings, except tab and margins, whose value is signed. There currently are no control settings whose value is signed.

tabValue
: A structure that contains tab distance and tab type settings.

marginsPtr
: A pointer to a TXNMargins (page 2714) structure that specifies the top, left, and right margin settings.

Discussion
The `TXNControlData` data type is a parameter in the `TXNSetTXNObjectControls` (page 2693) and `TXNGetTXNObjectControls` (page 2659) functions. The data contained in the union is determined by the `iControlData` parameter of those functions.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacTextEditor.h`

TXNErrors

Defines result codes. (**Deprecated.** Use `OSStatus` instead.)

```
typedef OSStatus TXNErrors;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacTextEditor.h`

TXNFindUPP

Defines a universal procedure pointer to a find callback function.

```
typedef TXNFindProcPtr TXNFindUPP;
```

Discussion
See `TXNFindProcPtr` (page 2703) for more information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacTextEditor.h`

TXNFontMenuObject

Contains private variables necessary to represent an MLTE Font menu.

```
typedef struct OpaqueTXNFontMenuObject * TXNFontMenuObject;
```

Discussion
Used in the functions `TXNNewFontMenuObject` (page 2666), `TXNPrepareFontMenu` (page 2672), `TXNGetFontMenuHandle` (page 2653), `TXNDoFontMenuSelection` (page 2633), and `TXNDisposeFontMenuObject` (page 2633).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
MacTextEditor.h

TXNFrameID

Represents the text frame to which actions should be applied.

```
typedef UInt32 TXNFrameID;
```

Discussion
Used in the functions TXNNewObject (page 2667), TXNActivate (page 2618), TXNResizeFrame (page 2676), TXNSetFrameBounds (page 2688), TXNDragReceiver (page 2634), and TXNDragTracker (page 2635).

Availability
Available in Mac OS X v10.0 and later.

Declared In
MacTextEditor.h

TXNLongRect

Contains coordinates for a view or text rectangle. (**Deprecated.** No longer needed.)

```
struct TXNLongRect {
    SInt32 top;
    SInt32 left;
    SInt32 bottom;
    SInt32 right;
};
typedef struct TXNLongRect TXNLongRect;
```

Fields

`top`
The top coordinate of the rectangle.

`left`
The left-side coordinate of the rectangle.

`bottom`
The bottom coordinate of the rectangle.

`right`
The right-side coordinate of the rectangle.

Special Considerations

The TXNLongRect data structure is passed as a parameter to the functions TXNSetRectBounds (page 2690) and TXNGetRectBounds (page 2657), which are deprecated. You should instead use the functions TXNSetHIRectBounds (page 2689) and TXNGetHIRect (page 2653), which take an HIRect data structure as a parameter instead of a TXNLongRect data structure

Availability
Available in Mac OS X v10.1 and later.

Declared In
MacTextEditor.h

TXNMacOSPreferredFontDescription

Contains information about the preferred font, font size, and style for a given text encoding.

```
struct TXNMacOSPreferredFontDescription {
    UInt32 fontID;
    Fixed pointSize;
    TextEncoding encoding;
    Style fontStyle;
};
typedef struct TXNMacOSPreferredFontDescription  TXNMacOSPreferredFontDescription;
```

Fields

`fontID`
The ID of the preferred font.

`pointSize`
The point size of the preferred font.

`encoding`
The text encoding of the preferred font.

`fontStyle`
The font style of the preferred font.

Discussion

Used in the functions `TXNInitTextension` (page 2662), `TXNSetFontDefaults` (page 2688), and `TXNGetFontDefaults` (page 2652).

Availability

Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In

`MacTextEditor.h`

TXNMargins

Contains the margin values of a text object.

```
struct TXNMargins {
    SInt16 topMargin;
    SInt16 leftMargin;
    SInt16 bottomMargin;
    SInt16 rightMargin;
};
typedef struct TXNMargins TXNMargins;
```

Fields

`topMargin`
The location of the top margin. Available in MLTE version 1.2 and later.

`leftMargin`
The location of the left margin. Available in MLTE version 1.2 and later.

`bottomMargin`
The location of the bottom margin. This is a placeholder; it is currently not possible to set the bottom margin.

`rightMargin`
The location of the right margin. Available in MLTE version 1.2 and later.

Discussion
This structure is used as a field in the `TXNControlData` (page 2711) union.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacTextEditor.h`

TXNMatchTextRecord

Contains information about the text to be matched in a find operation.

```
struct TXNMatchTextRecord {
    const void * iTextPtr;
    SInt32 iTextToMatchLength;
    TextEncoding iTextEncoding;
};
typedef struct TXNMatchTextRecord TXNMatchTextRecord;
```

Fields
`iTextPtr`
A pointer to the text to be matched.

`iTextToMatchLength`
The length of text to which the `iTextPtr` parameter points.

`iTextEncoding`
The encoding used by the text to be matched.

Discussion
Used in the `TXNFind` (page 2641) function and the callback `TXNFindProcPtr` (page 2703).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacTextEditor.h`

TXNObject

Contains private variables and functions necessary to represent text and handle text formatting at a document level.

```
typedef struct OpaqueTXNObject * TXNObject;
```

Discussion
You obtain a structure of type `TXNObject` from the `TXNNewObject` (page 2667) function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
MacTextEditor.h

TXNObjectRefCon

Contains data specific to your application. (**Deprecated.** Used only in the `TXNNewObject` (page 2667) function, which is deprecated.)

```
typedef void * TXNObjectRefCon;
```

Declared In
MacTextEditor.h

TXNScrollInfoUPP

Defines a universal procedure pointer to a scroll callback function.

```
typedef TXNScrollInfoProcPtr TXNScrollInfoUPP;
```

Discussion
See `TXNScrollInfoProcPtr` (page 2705) for more information.

Availability
Available in Mac OS X v10.2 and later.

Declared In
MacTextEditor.h

TXNTab

Contains tab information for a text object.

```
struct TXNTab {
    SInt16 value;
    TXNTabType tabType;
    UInt8 filler;
};
typedef struct TXNTab TXNTab;
```

Fields

`value`
The distance between tabs.

`tabType`
The type of tab settings, such as right or left. See Tab Types (page 2767) for a description of possible values.

`filler`
An unsigned 8-bit integer that exists only to make the structure exactly 4 bytes in size.

Discussion
Passed in the `iControlData` parameter of the `TXNSetTXNObjectControls` (page 2693) function when the value of the `iControlTags` parameter is `kTXNTabSettingsTag`, or returned in the `oControlData` parameter of the `TXNGetTXNObjectControls` (page 2659) function when the value of the `iControlTags` parameter is `kTXNTabSettingsTag`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacTextEditor.h`

TXNTextBoxOptionsData

Contains information about how text appears in a Unicode text box.

```
struct TXNTextBoxOptionsData {
    TXNTextBoxOptions optionTags;
    Fract flushness;
    Fract justification;
    Fixed rotation;
    void * options;
};
typedef struct TXNTextBoxOptionsData TXNTextBoxOptionsData;
```

Fields

`optionTags`
Specifies the field in this structure at which MLTE should look. See Text Box Options Masks (page 2770) for a description of possible values. You must supply the data associated with this tag in the appropriate field. For example, if you set the value of the `optionTags` field to `kTXNSetJustificationMask`, you must specify the type of justification in the `justification` field.

`flushness`
Indicates whether text should be displayed flush left, flush right, or centered in the text box. You should use one of the line justification constants defined in ATSUnicode.h. The possible values are `kATSUStartAlignment`, `kATSUEndAlignment`, and `kATSUCenterAlignment`.

`justification`
The type of justification to use in the text box. You should use one of the line justification constants defined in ATSUnicode.h. The possible values are `kATSUNoJustification` and `kATSUFullJustification`.

`rotation`
The angle of rotation for text in the text box.

`options`
Reserved for future use. This should be set to `NULL`.

Discussion
Used in the `TXNDrawUnicodeTextBox` (page 2639) and `TXNDrawCFStringTextBox` (page 2637) functions.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`MacTextEditor.h`

TXNTypeAttributes

Contains information about attributes for a text object.

```
struct TXNTypeAttributes {
    TXTNTag tag;
    ByteCount size;
    TXNAttributeData data;
};
typedef struct TXNTypeAttributes TXNTypeAttributes;
```

Fields

`tag`
: A value that specifies the type of information contained in the `data` field. See Font Run Attributes (page 2739) for a description of possible values.

`size`
: The size of the attribute. See Font Run Attribute Sizes (page 2740) for a description of possible values.

`data`
: A union that serves either as a 32-bit integer or a 32-bit pointer, depending on the `size` field.

Discussion
Used in the functions `TXNSetTypeAttributes` (page 2694), `TXNGetContinuousTypeAttributes` (page 2648), and `TXNGetIndexedRunInfoFromRange` (page 2654).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacTextEditor.h`

TXNVersionValue

Specifies the version of MLTE in use.

```
typedef UInt32 TXNVersionValue;
```

Discussion
Returned by the `TXNVersionInformation` (page 2698) function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`MacTextEditor.h`

TXTNTag

Specifies the type of information you want passed in the `data` field of the `TXNTypeAttributes` structure. (**Deprecated.** Use `TXNTypeRunAttributes` (page 2739).)

```
typedef FourCharCode TXTNTag;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
MacTextEditor.h

Constants

Action Constants

Specify constants for actions used when calling TXNCanUndoAction (page 2624) and TXNCanRedoAction (page 2622).

```
const CFStringRef kTXNActionTyping;
const CFStringRef kTXNActionCut;
const CFStringRef kTXNActionPaste;
const CFStringRef kTXNActionClear;
const CFStringRef kTXNActionChangeFont;
const CFStringRef kTXNActionChangeColor;
const CFStringRef kTXNActionChangeSize;
const CFStringRef kTXNActionChangeStyle;
const CFStringRef kTXNActionAlignLeft;
const CFStringRef kTXNActionAlignCenter;
const CFStringRef kTXNActionAlignRight;
const CFStringRef kTXNActionDrop;
const CFStringRef kTXNActionMove;
const CFStringRef kTXNActionChangeFontFeature;
const CFStringRef kTXNActionChangeFontVariation;
const CFStringRef kTXNActionChangeGlyphVariation;
const CFStringRef kTXNActionChangeTextPosition;
const CFStringRef kTXNActionUndoLast;
```

Constants

kTXNActionTyping
A typing action.

kTXNActionCut
A cut action.

kTXNActionPaste
A paste action.

kTXNActionClear
A clear action.

kTXNActionChangeFont
A font change action.

kTXNActionChangeColor
A color change action.

kTXNActionChangeSize
A size change action.

kTXNActionChangeStyle
A change in style action.

kTXNActionAlignLeft
An align left action.

`kTXNActionAlignCenter`
An align center action.

`kTXNActionAlignRight`
An align right action.

`kTXNActionDrop`
A drop action.

`kTXNActionMove`
A move action.

`kTXNActionChangeFontFeature`
A change font feature action.

`kTXNActionChangeFontVariation`
A change in font variation action.

`kTXNActionChangeGlyphVariation`
A change glyph variation action.

`kTXNActionChangeTextPosition`
A change text position action; includes changing the space before and after characters and shifting the text's baseline.

`kTXNActionUndoLast`
Used in undo and redo functions if none of the other constants apply.

Discussion
Use these constants when calling the `TXNCanUndoAction` (page 2624) and `TXNCanRedoAction` (page 2622) functions.

Declared In
`MacTextEditor.h`

Action Count Constants

Represent action types use by `TXNGetCountForActionType` (page 2649) and `TXNClearCountForActionType` (page 2626).

```
const CFStringRef kTXNActionCountOfTextChanges;
const CFStringRef kTXNActionCountOfStyleChanges;
const CFStringRef kTXNActionCountOfAllChanges;
```

Constants
`kTXNActionCountOfTextChanges`
Count of text changes. All text changes other than style changes and custom defined actions are included in this action count. Includes key presses, inline sessions, cut, copy, and paste, and drop. Undo and redo events of these kinds are also included in this action count.

`kTXNActionCountOfStyleChanges`
Count of text style changes. Style changes include changing font, font face, font size, font feature, font variation, font color, glyph variation, and text position. Undo or redo events of these kinds are also included in this action count.

`kTXNActionCountOfAllChanges`
Total count of actions including all text and style changes, as well as custom defined actions.

Declared In
`MacTextEditor.h`

Action Count Bits

Specify actions to be included in an action count when calling `TXNGetActionChangeCount` (page 2646) and `TXNClearActionChangeCount` (page 2625). (**Deprecated.** See Action Count Constants (page 2720).)

```
enum {
    kTXNTextInputCountBit = 0,
    kTXNRunCountBit = 1
};
```

Constants

`kTXNTextInputCountBit`

When this bit is set, general text input events that affect the content of the document are included in the action count. General text input events include key presses, inline sessions, pasting, cutting, dropping, and other editing events. Undo or redo events of text input events are also included in the action count.

Available in Mac OS X v10.1 and later.

Declared in `MacTextEditor.h`.

`kTXNRunCountBit`

When this bit is set, general style changes to the text are included in the action count. Style changes include changes to the text face, font, font size and so forth. Undo and redo events of style changes are also included in the action count.

Available in Mac OS X v10.1 and later.

Declared in `MacTextEditor.h`.

Declared In

`MacTextEditor.h`

Action Count Masks

Set or test action count bits for use with `TXNGetActionChangeCount` (page 2646) and `TXNClearActionChangeCount` (page 2625). (**Deprecated.** See Action Count Constants (page 2720).)

```
typedef OptionBits TXNCountOptions;
enum {
    kTXNTextInputCountMask = 1L << kTXNTextInputCountBit,
    kTXNRunCountMask = 1L << kTXNRunCountBit,
    kTXNAllCountMask = kTXNTextInputCountMask | kTXNRunCountMask
};
```

Constants

`kTXNTextInputCountMask`

Use to set or test for the `kTXNTextInputCountBit`.

Available in Mac OS X v10.1 and later.

Declared in `MacTextEditor.h`.

`kTXNRunCountMask`

Used to set or test for the `kTXNRunCountBit`.

Available in Mac OS X v10.1 and later.

Declared in `MacTextEditor.h`.

`kTXNAllCountMask`
Use to set or text for both `kTXNTextInputCountBit` and `kTXNRunCountBit`.

Available in Mac OS X v10.1 and later.

Declared in `MacTextEditor.h`.

Declared In
`MacTextEditor.h`

Action Types

Specify constants for editing actions taken by the user in versions of Mac OS X prior to Mac OS X v10.4. (**Deprecated.** These constants were used in the `TXNCanUndo` (page 2623) and `TXNCanRedo` (page 2622) functions, which are deprecated in Mac OS X v10.4. Use the `TXNCanUndoAction` (page 2624) and `TXNCanRedoAction` (page 2622) functions instead.)

```
typedef UInt32 TXNActionKey;
enum {
    kTXNTypingAction = 0,
    kTXNCutAction = 1,
    kTXNPasteAction = 2,
    kTXNClearAction = 3,
    kTXNChangeFontAction = 4,
    kTXNChangeFontColorAction = 5,
    kTXNChangeFontSizeAction = 6,
    kTXNChangeStyleAction = 7,
    kTXNAlignLeftAction = 8,
    kTXNAlignCenterAction = 9,
    kTXNAlignRightAction = 10,
    kTXNDropAction = 11,
    kTXNMoveAction = 12,
    kTXNFontFeatureAction = 13,
    kTXNFontVariationAction = 14,
    kTXNUndoLastAction = 1024
};
```

Constants
`kTXNTypingAction`
A typing action.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNCutAction`
A cut action.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNPasteAction`
A paste action.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNClearAction`
A clear action.
Available in Mac OS X v10.0 and later.
Declared in `MacTextEditor.h`.

`kTXNChangeFontAction`
A font change action.
Available in Mac OS X v10.0 and later.
Declared in `MacTextEditor.h`.

`kTXNChangeFontColorAction`
A change in font color action.
Available in Mac OS X v10.0 and later.
Declared in `MacTextEditor.h`.

`kTXNChangeFontSizeAction`
A change in font size action.
Available in Mac OS X v10.0 and later.
Declared in `MacTextEditor.h`.

`kTXNChangeStyleAction`
A change in font style action.
Available in Mac OS X v10.0 and later.
Declared in `MacTextEditor.h`.

`kTXNAlignLeftAction`
An align left action.
Available in Mac OS X v10.0 and later.
Declared in `MacTextEditor.h`.

`kTXNAlignCenterAction`
An align center action.
Available in Mac OS X v10.0 and later.
Declared in `MacTextEditor.h`.

`kTXNAlignRightAction`
An align right action.
Available in Mac OS X v10.0 and later.
Declared in `MacTextEditor.h`.

`kTXNDropAction`
A drop action.
Available in Mac OS X v10.0 and later.
Declared in `MacTextEditor.h`.

`kTXNMoveAction`
A move selection action.
Available in Mac OS X v10.0 and later.
Declared in `MacTextEditor.h`.

`kTXNFontFeatureAction`
A change in font feature action.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNFontVariationAction`
A change in font variation action.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNUndoLastAction`
Use this if none of the other constants apply.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

Declared In
`MacTextEditor.h`

Apple Event Handler Bits

Specify whether Apple events should be used.

```
enum {
    kTXNNoAppleEventHandlersBit = 0,
    kTXNRestartAppleEventHandlersBit = 1
};
```

Constants
`kTXNNoAppleEventHandlersBit`

When this bit is set, Apple event handlers are not used. (**Deprecated.** There is no replacement.)

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNRestartAppleEventHandlersBit`

When this bit is set, Apple event handlers are started up. (**Deprecated.** There is no replacement.)

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

Special Considerations

MLTE does not support Apple events in Mac OS X version 10.1 and later.

Declared In
`MacTextEditor.h`

Apple Event Handler Masks

Set or test Apple event handler bits.

```
enum {
    kTXNNoAppleEventHandlersMask = 1 << kTXNNoAppleEventHandlersBit,
    kTXNRestartAppleEventHandlersMask = 1 << kTXNRestartAppleEventHandlersBit
};
```

Constants

`kTXNNoAppleEventHandlersMask`

Use to set or test for the `kTXNNoAppleEventHandlersBit`. Use this if you want Apple event handlers removed. (**Deprecated.** There is no replacement.)

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNRestartAppleEventHandlersMask`

Used to set or test for the `kTXNRestartAppleEventHandlersBit`. Use this if you want to subsequently restart Apple event handlers after removing your own text handlers. (**Deprecated.** There is no replacement.)

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

Discussion

These constants are currently the only settings for the flags field of `TXNCarbonEventInfo`.

Special Considerations

MLTE does not support Apple events in Mac OS X version 10.1 and later

Declared In

`MacTextEditor.h`

ATSUI Feature Bits

Specify the default imaging system.

```
enum {
    kTXNWillDefaultToATSUIBit = 0,
    kTXNWillDefaultToCarbonEventBit = 1
};
```

Constants

`kTXNWillDefaultToATSUIBit`

When this bit is set, indicates ATSUI is the default imaging system.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNWillDefaultToCarbonEventBit`

When this bit is set, indicates MLTE uses Carbon events by default.

Available in Mac OS X v10.1 and later.

Declared in `MacTextEditor.h`.

Declared In

`MacTextEditor.h`

ATSUI Feature Masks

Specifies new features and represents the default imaging system.

```
typedef OptionBits TXNFeatureBits;
enum {
    kTXNWillDefaultToATSUIMask = 1L << kTXNWillDefaultToATSUIBit,
    kTXNWillDefaultToCarbonEventMask = 1L << kTXNWillDefaultToCarbonEventBit
};
```

Constants

`kTXNWillDefaultToATSUIMask`

Test for ATSUI as the default imaging system.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNWillDefaultToCarbonEventMask`

Test for Carbon events as the default event handling mechanism.

Available in Mac OS X v10.1 and later.

Declared in `MacTextEditor.h`.

Discussion

You can use this to test for bit 0 in the `oFeatureFlags` parameter returned by the `TXNVersionInformation` (page 2698) function.

Declared In

`MacTextEditor.h`

Automatic Indentation Settings

Specify the automatic indentation setting for a text object.

```
enum {
    kTXNAutoIndentOff = false,
    kTXNAutoIndentOn = true
};
```

Constants

`kTXNAutoIndentOff`

Automatic indenting is not enabled.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNAutoIndentOn`

Automatic indenting is enabled. You can enable this feature only if automatic word wrapping is not enabled.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

Discussion

Passed in the `iControlData` parameter of the `TXNSetTXNObjectControls` (page 2693) function when the value of the `iControlTags` parameter is `kTXNAutoIndentStateTag`, or returned in the `oControlData` parameter of the `TXNGetTXNObjectControls` (page 2659) function when the value of the `iControlTags` parameter is `kTXNAutoIndentStateTag`.

Declared In
MacTextEditor.h

Automatic Scrolling Behavior

Specify automatic scrolling behavior for a text object.

```
enum {
    kTXNAutoScrollInsertionIntoView = 0,
    kTXNAutoScrollNever = 1,
    kTXNAutoScrollWhenInsertionVisible = 2
};
typedef UInt32 TXNAutoScrollBehavior;
```

Constants

`kTXNAutoScrollInsertionIntoView`
The default auto scrolling behavior. When text is inserted, the document is scrolled to show the new insertion. This was the only type of autoscrolling prior to Mac OS X v10.4.

Available in Mac OS X v10.4 and later.

Declared in `MacTextEditor.h`.

`kTXNAutoScrollNever`
Never autoscroll, even when dragging the mouse or inserting text. The only way to scroll the document is for the user to use the scrollbar or to scroll programatically.

Available in Mac OS X v10.4 and later.

Declared in `MacTextEditor.h`.

`kTXNAutoScrollWhenInsertionVisible`
Autoscrolling only happens when the insertion offset is currently in the user's view. If the user is looking at the first page of a ten page document and text is inserted at the end of the document, no autoscrolling occurs. However, if the user was looking at page ten and text is inserted there, the document would scroll. This type of autoscrolling is best for implementing terminal or log windows.

Available in Mac OS X v10.4 and later.

Declared in `MacTextEditor.h`.

Declared In
MacTextEditor.h

Carbon Event Dictionary Keys

Specify Carbon events to be handled by MLTE.

```
#define kTXNTextHandlerKey CFSTR("TextInput")
#define kTXNWindowEventHandlerKey CFSTR("WindowEvent")
#define kTXNWindowResizeEventHandlerKey CFSTR("WindowResize")
#define kTXNCommandTargetKey CFSTR("CommandTarget")
#define kTXNCommandUpdateKey CFSTR("CommandUpdate")
#define kTXNFontMenuObjectKey CFSTR("FontMenuObject")
#define kTXNActionNameMapperKey CFSTR("ActionNameMapper")
#define kTXNWheelMouseEventHandlerKey CFSTR("WheelMouseEvent")
#define kTXNTSMDocumentAccessHandlerKey CFSTR("TSMDocumentAccess")
#define kTXNFontPanelEventHandlerKey CFSTR("FontPanel")
#define kTXNFontMenuRefKey CFSTR("FontMenuRef")
#define kTXNActionKeyMapperKey CFSTR("ActionKeyMapper")
```

Constants

`kTXNTextHandlerKey`
Indicates the Carbon event class `kEventClassTextInput` and the event kinds `kEventTextInputUpdateActiveInputArea`, `kEventTextInputUnicodeForKeyEvent`, `kEventTextInputOffsetToPos`, `kEventTextInputPosToOffset`, and `kEventTextInputGetSelectedText`.

`kTXNWindowEventHandlerKey`
Indicates the Carbon event class `kEventClassWindow` and the event kinds `kEventWindowActivated`, `kEventWindowDeactivated`, `kEventWindowDrawContent`, and `kEventWindowClickContentRegion`.

`kTXNWindowResizeEventHandlerKey`
Indicates the Carbon event class `kEventClassWindow` and a window resizing event.

`kTXNCommandTargetKey`
Indicates the Carbon event class `kEventClassCommand` and the event kind `kEventCommandProcess`.

`kTXNCommandUpdateKey`
Indicates the Carbon event class `kEventClassCommand` and the event kind `kEventCommandUpdate`.

`kTXNFontMenuObjectKey`
Indicates the Carbon event class `kEventClassMenu` and the event kind `kEventMenuEnableItems`.

`kTXNActionNameMapperKey`
Indicates an action key mapper callback function. Available in Mac OS X v10.4; use instead of `kTXNActionKeyMapperKey`.

`kTXNWheelMouseEventHandlerKey`
Indicates the handler for wheel mouse events.

`kTXNTSMDocumentAccessHandlerKey`
Indicates the handler for TSM document access events.

`kTXNFontPanelEventHandlerKey`
Indicates the handler for Font Panel events.

`kTXNFontMenuRefKey`
Indicates the Carbon event class `kEventClassMenu`.

`kTXNActionKeyMapperKey`
Indicates an action key mapper callback function. (**Deprecated.** Use `kTXNActionNameMapperKey` instead.)

Declared In
`MacTextEditor.h`

Clearance Settings

Clear formatting and privileges settings.

```
enum {
    kTXNClearThisControl = 0xFFFFFFFF,
    kTXNClearTheseFontFeatures = 0x80000000
};
```

Constants

`kTXNClearThisControl`

Clears control settings. If you want to clear a setting associated with a control tag, you can call the `TXNSetTXNObjectControls` (page 2693) function with the value of the `iControlData` parameter set to `kTXNClearThisControl`. MLTE resets the value of the control specified in the `iControlTag` parameter of the `TXNSetTXNObjectControls` (page 2693) function to the default value for that control.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNClearTheseFontFeatures`

Clears font feature settings. You can use this constant when you call the `TXNSetTypeAttributes` (page 2694) function to clear all of the ATSUI font features or ATSUI font variations.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

Declared In

`MacTextEditor.h`

Command Event Support Options

Specify options for enabling support for command events in a TXNObject.

```
enum {
    kTXNSupportEditCommandProcessing = 1 << 0,
    kTXNSupportEditCommandUpdating = 1 << 1,
    kTXNSupportSpellCheckCommandProcessing = 1 << 2,
    kTXNSupportSpellCheckCommandUpdating = 1 << 3,
    kTXNSupportFontCommandProcessing = 1 << 4,
    kTXNSupportFontCommandUpdating = 1 << 5
};
typedef UInt32 TXNCommandEventSupportOptions;
```

Constants

`kTXNSupportEditCommandProcessing`

Setting this bit when calling `TXNSetTXNObjectControls` (page 2693) enables support for processing the menu item associated with `kHICommandUndo`, `kHICommandRedo`, `kHICommandCut`, `kHICommandCopy`, `kHICommandPaste`, `kHICommandClear`, and `kHICommandSelectAll`. If this bit is not set when `TXNSetTXNObjectControls` (page 2693) is called, support is disabled.

Available in Mac OS X v10.4 and later.

Declared in `MacTextEditor.h`.

`kTXNSupportEditCommandUpdating`

Setting this bit when calling `TXNSetTXNObjectControls` (page 2693) enables support for updating the menu item associated with `kHICommandUndo`, `kHICommandRedo`, `kHICommandCut`, `kHICommandCopy`, `kHICommandPaste`, `kHICommandClear`, and `kHICommandSelectAll`. For Undo, the item is enabled if there are any undoable actions in MLTE's command stack, and, if you have installed an action key mapper proc, it is called to get the appropriate string for the Undo item. For Redo, the item is enabled if there are any redoable actions; if you have installed an action key mapper callback, it is called to get the appropriate string for the Redo item. For Cut and Clear, the item is enabled if there is current selection that is not empty; otherwise, these items are disabled. For Paste, the item is enabled if the clipboard is not empty; it is disabled if the clipboard is empty or contains data that MLTE cannot parse. For Select All, the item is always updated. If this bit is not set when `TXNSetTXNObjectControls` (page 2693) is called, support is disabled.

Available in Mac OS X v10.4 and later.

Declared in `MacTextEditor.h`.

`kTXNSupportSpellCheckCommandProcessing`

Setting this bit when calling `TXNSetTXNObjectControls` (page 2693) enables support for spell checking. The spell checking commands supported are: Show Spelling Panel (`'shsp'`), Check Spelling (`'cksp'`), Change Spelling (`'chsp'`), enable check spelling as you type (`'aspc'`), ignore spelling (`'igsp'`), and learn spelling (`'lrwd'`). If this bit is not set when `TXNSetTXNObjectControls` (page 2693) is called, support is disabled.

Available in Mac OS X v10.4 and later.

Declared in `MacTextEditor.h`.

`kTXNSupportSpellCheckCommandUpdating`

Enables support for updating the menu item associated with a given spell checking command. Once `kTXNSupportSpellCheckCommandUpdating` is enabled, the Show Spelling and Check Spelling items are always enabled. The Change Spelling item is included in a spelling menu only if the current selection is a misspelled word; it is disabled if the current selection is empty or not a misspelled word. The Check Spelling as You Type item is always enabled. It is checked if this feature has been enabled. By default when you turn on spell checking, this item is enabled. If this feature has been disabled, the item is not checked. Ignore Spelling usually does not have a corresponding menu item. If a menu does have this item, Ignore Spelling is disabled if the current selection is empty or is not a misspelled word. It is enabled if the current selection is a misspelled word. Learn Spelling typically does not have a corresponding menu item. If a menu does have this item, Learn Spelling is disabled if the current selection is empty or is not a misspelled word. It is enabled if the current selection is a misspelled word. If this bit is not set when `TXNSetTXNObjectControls` (page 2693) is called, support is disabled.

Available in Mac OS X v10.4 and later.

Declared in `MacTextEditor.h`.

`kTXNSupportFontCommandProcessing`

Setting this bit enables Carbon Font Panel support. Once enabled, MLTE handles the following Carbon Events defined in `FontPanel.h`: `kHICommandShowHideFontPanel` and `kEventFontPanelClosed` to show and hide the Carbon font panel, `kEventFontSelection` event to update the document after the selection of a new font, size, style, color, or any feature settings from the Typography Panel. If this bit is not set when `TXNSetTXNObjectControls` (page 2693) is called, support is disabled.

Available in Mac OS X v10.4 and later.

Declared in `MacTextEditor.h`.

`kTXNSupportFontCommandUpdating`
Setting this bit enables support for updating the selection in Carbon Font Panel when the current selection in an MLTE document is changed. When this bit is set, `kTXNSupportFontCommandProcessing` must also be set. If this bit is not set when `TXNSetTXNObjectControls` (page 2693) is called, support is disabled.

Available in Mac OS X v10.4 and later.

Declared in `MacTextEditor.h`.

Declared In
`MacTextEditor.h`

Continuous Style Information Bits

Specify whether font information is continuous.

```
enum {
    kTXNFontContinuousBit = 0,
    kTXNSizeContinuousBit = 1,
    kTXNStyleContinuousBit = 2,
    kTXNColorContinuousBit = 3
};
```

Constants
`kTXNFontContinuousBit`
When this bit is set, the font is continuous in a text run.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNSizeContinuousBit`
When this bit is set, the font size is continuous in a text run.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNStyleContinuousBit`
When this bit is set, the font style is continuous in a text run.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNColorContinuousBit`
When this bit is set, the font color is continuous in a text run.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

Declared In
`MacTextEditor.h`

Continuous Style Information Masks

Represents continuous style information needed by your application.

```
typedef OptionBits TXNContinuousFlags;
enum {
    kTXNFontContinuousMask = 1L << kTXNFontContinuousBit,
    kTXNSizeContinuousMask = 1L << kTXNSizeContinuousBit,
    kTXNStyleContinuousMask = 1L << kTXNStyleContinuousBit,
    kTXNColorContinuousMask = 1L << kTXNColorContinuousBit
};
```

Constants

`kTXNFontContinuousMask`
Use to test for continuous font information in a text run.
Available in Mac OS X v10.0 and later.
Declared in `MacTextEditor.h`.

`kTXNSizeContinuousMask`
Use to test for continuous size information in a text run.
Available in Mac OS X v10.0 and later.
Declared in `MacTextEditor.h`.

`kTXNStyleContinuousMask`
Use to test for continuous style information in a text run.
Available in Mac OS X v10.0 and later.
Declared in `MacTextEditor.h`.

`kTXNColorContinuousMask`
Use to test for continuous color information in a text run.
Available in Mac OS X v10.0 and later.
Declared in `MacTextEditor.h`.

Discussion
Used in the `TXNGetContinuousTypeAttributes` (page 2648) function.

Declared In
`MacTextEditor.h`

Data Offsets

Specifies offsets to use when manipulating data in a text object. Offsets in MLTE are always character offsets.

```
typedef UInt32 TXNOffset;
enum {
    kTXNUseCurrentSelection = 0xFFFFFFFF,
    kTXNStartOffset = 0,
    kTXNEndOffset = 0x7FFFFFFF
};
```

Constants

`kTXNUseCurrentSelection`
Use the current selection.
Available in Mac OS X v10.0 and later.
Declared in `MacTextEditor.h`.

`kTXNStartOffset`
The first offset of the text in a text object.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNEndOffset`
The last offset of the text in a text object.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

Discussion
These constants can be passed and returned in functions that have the parameter of type `TXNOffset`.

Declared In
`MacTextEditor.h`

Data Option Key Value Constants

Specifies constants used as key values in Data Option dictionaries.

```
const CFStringRef kTXNPlainTextDocumentType
const CFStringRef kTXNMLTEDocumentType
const CFStringRef kTXNRTFDocumentType
const CFStringRef kTXNQuickTimeDocumentType
```

Constants
`kTXNPlainTextDocumentType`
Plain text document.

`kTXNMLTEDocumentType`
Native MLTE document type.

`kTXNRTFDocumentType`
Rich text format (RTF) document type.

`kTXNQuickTimeDocumentType`
Multimedia file that can be opened by QuickTime importers. This document type is only supported for reading data, not for writing data.

Discussion
These constants are passed in Data Option dictionary keys to `TXNReadFromCFURL` (page 2673) and `TXNWriteRangeToCFURL` (page 2699) to specify options that are to be used when reading data into a TXNObject and when writing data from a TXNObject to a file or special file bundle (directory).

Declared In
`MacTextEditor.h`

Data Option Key Constants

Specifies keys for use in dictionaries passed as parameters to `TXNReadFromCFURL` (page 2673) and `TXNWriteRangeToCFURL` (page 2699).

```
const CFStringRef kTXNDataOptionDocumentTypeKey
const CFStringRef kTXNDataOptionCharacterEncodingKey
```

Constants

`kTXNDataOptionDocumentTypeKey`
: `CFString` specifying the document format. The following constants are supported: `kTXNPlainTextDocumentType`, `kTXNMLTEDocumentType`, `kTXNRTFDocumentType`, and `kTXNQuickTimeDocumentType`. For information on these constants, see Data Option Key Value Constants (page 2733).

`kTXNDataOptionCharacterEncodingKey`
: CFNumber of type `kCFNumberSInt32Type` containing the character encoding as specified in `CFString.h` and `CFStringEncodingExt.h`.

Discussion

Data options are used to specify options for reading in and writing out data.

Declared In

`MacTextEditor.h`

Default Font Name

Specifies the default font name.

```
enum {
    kTXNDefaultFontName = 0
};
```

Constants

`kTXNDefaultFontName`
: The default font name.

Discussion

MLTE used these constants in an earlier version in which only a single font was allowed. You can now specify an array of font descriptions by using the `TXNMacOSPreferredFontDescription` (page 2714) structure. See the function `TXNInitTextension` (page 2662) for a description of how to specify defaults for a font.

Declared In

`MacTextEditor.h`

Default Font Size

Specifies the default font size.

```
enum {
    kTXNDefaultFontSize = 0x000C0000
};
```

Constants

`kTXNDefaultFontSize`
: Sets default font size.

  Available in Mac OS X v10.1 and later.

  Not available to 64-bit applications.

  Declared in `MacTextEditor.h`.

Declared In
MacTextEditor.h

Default Font Style

Specifies the default font style.

```
enum {
    kTXNDefaultFontStyle = normal
};
```

Constants
kTXNDefaultFontStyle
Sets default font style.

Available in Mac OS X v10.1 and later.

Not available to 64-bit applications.

Declared in MacTextEditor.h.

Declared In
MacTextEditor.h

Document Attribute Keys

Specify dictionary keys for document attribute dictionaries used by TXNWriteRangeToCFURL (page 2699) and TXNReadFromCFURL (page 2673).

```
const CFStringRef kTXNDocumentAttributeTitleKey;
const CFStringRef kTXNDocumentAttributeCompanyNameKey;
const CFStringRef kTXNDocumentAttributeSubjectKey;
const CFStringRef kTXNDocumentAttributeAuthorKey;
const CFStringRef kTXNDocumentAttributeKeywordsKey;
const CFStringRef kTXNDocumentAttributeCommentKey;
const CFStringRef kTXNDocumentAttributeEditorKey;
const CFStringRef kTXNDocumentAttributeCreationTimeKey;
const CFStringRef kTXNDocumentAttributeModificationTimeKey;
const CFStringRef kTXNDocumentAttributeCopyrightKey;
```

Constants
kTXNDocumentAttributeTitleKey
CFString containing the document's title.

kTXNDocumentAttributeCompanyNameKey
CFString containing the company name.

kTXNDocumentAttributeSubjectKey
CFString containing the document's subject.

kTXNDocumentAttributeAuthorKey
CFString containing the name of the document's author.

kTXNDocumentAttributeKeywordsKey
CFArray of values of type CFString containing keywords.

kTXNDocumentAttributeCommentKey
CFString containing comments.

`kTXNDocumentAttributeEditorKey`
`CFString` containing the name of the person who last edited the document.

`kTXNDocumentAttributeCreationTimeKey`
`CFAbsoluteTime` containing document comments; note that this is not the file system creation date of the file, but of the document, as it is stored in the document.

`kTXNDocumentAttributeModificationTimeKey`
`CFAbsoluteTime` containing the last modification date of the document contents.

`kTXNDocumentAttributeCopyrightKey`
`CFString` containing the copyright of the document.

Discussion
Use these constants when working with document attribute dictionaries that are passed to `TXNWriteRangeToCFURL` (page 2699) and `TXNReadFromCFURL` (page 2673).

When writing data out, document attributes are embedded into the data stream for document formats that support them (i.e. MLTE native format and RTF). When reading data in, document attributes are extracted from the data stream if the document format supports them.

Declared In
`MacTextEditor.h`

Drag and Drop Constants

Specify whether or not drag and drop is enabled.

```
enum {
    kTXNEnableDragAndDrop = false,
    kTXNDisableDragAndDrop = true
};
```

Constants
`kTXNEnableDragAndDrop`
Enables drag and drop when passed as a parameter to the function `TXNSetTXNObjectControls` (page 2693). Indicates drag and drop is disabled when returned from the function `TXNGetTXNObjectControls` (page 2659).

Available in Mac OS X v10.1 and later.

Declared in `MacTextEditor.h`.

`kTXNDisableDragAndDrop`
Disables drag and drop when passed as a parameter to the function `TXNSetTXNObjectControls` (page 2693). Indicates drag and drop is enabled when returned from the function `TXNGetTXNObjectControls` (page 2659).

Available in Mac OS X v10.1 and later.

Declared in `MacTextEditor.h`.

Declared In
`MacTextEditor.h`

Draw Items Bits

Specify which elements of the text object to render.

```
enum {
    kTXNDrawItemScrollbarsBit= 0,
    kTXNDrawItemTextBit= 1,
    kTXNDrawItemTextAndSelectionBit = 2
};
```

Constants

`kTXNDrawItemScrollbarsBit`
Specifies to draw the scroll bars.
Available in Mac OS X v10.3 and later.
Declared in `MacTextEditor.h`.

`kTXNDrawItemTextBit`
Specifies to render the text.
Available in Mac OS X v10.3 and later.
Declared in `MacTextEditor.h`.

`kTXNDrawItemTextAndSelectionBit`
Specifies to render the text and the current selection.
Available in Mac OS X v10.3 and later.
Declared in `MacTextEditor.h`.

Discussion

See `Draw Items Masks` (page 2737).

Declared In

`MacTextEditor.h`

Draw Items Masks

Test for draw-items bits.

```
enum {
    kTXNDrawItemScrollbarsMask = 1UL << kTXNDrawItemScrollbarsBit,
    kTXNDrawItemTextMask = 1UL << kTXNDrawItemTextBit,
    kTXNDrawItemTextAndSelectionMask = 1UL <<
kTXNDrawItemTextAndSelectionBit,
    kTXNDrawItemAllMask =  0xFFFFFFFF
};
typedef OptionBits TXNDrawItems;
```

Constants

`kTXNDrawItemScrollbarsMask`
Use to set or test for the `kTXNDrawItemScrollbarsBit`.
Available in Mac OS X v10.3 and later.
Declared in `MacTextEditor.h`.

`kTXNDrawItemTextMask`
Used to set or test for the `kTXNDrawItemTextBit`.
Available in Mac OS X v10.3 and later.
Declared in `MacTextEditor.h`.

`kTXNDrawItemTextAndSelectionMask`
Use to set or text for the `kTXNDrawItemTextAndSelectionBit`.

Available in Mac OS X v10.3 and later.

Declared in `MacTextEditor.h`.

`kTXNDrawItemAllMask`
Used to set all draw-items bits or test to see whether all draw-items bits are set. Setting all bits specifies to draw the scroll bars, text, and the current selection.

Available in Mac OS X v10.3 and later.

Declared in `MacTextEditor.h`.

Discussion
These constants can be passed as parameters to the function `TXNDrawObject` (page 2638).

Declared In
`MacTextEditor.h`

Font Defaults

Specify a variety of font settings.

```
enum {
    kTXNDontCareTypeSize = 0xFFFFFFFF,
    kTXNDontCareTypeStyle = 0xFF,
    kTXNIncrementTypeSize = 0x00000001,
    kTXNDecrementTypeSize = 0x80000000,
    kTXNUseScriptDefaultValue = -1,
    kTXNNoFontVariations = 0x7FFF
};
```

Constants
`kTXNDontCareTypeSize`
Use the application default font size.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNDontCareTypeStyle`
Use “normal” should as the font style.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNIncrementTypeSize`
Increase the font size should by one point.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNDecrementTypeSize`
Decrease the font size by one point.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

Discussion
These constants can be used as parameters in a variety of functions that control font attributes, such as the `TXNSetFontDefaults` (page 2688) and `TXNSetTypeAttributes` (page 2694) functions.

Declared In
`MacTextEditor.h`

Font Run Attributes

Specifies a font attribute (font family, size, style, and so forth) for a text run in a text object.

```
typedef FourCharCode TXNTypeRunAttributes;
enum {
    kTXNQDFontNameAttribute = 'fntn',
    kTXNQDFontFamilyIDAttribute = 'font',
    kTXNQDFontSizeAttribute = 'size',
    kTXNQDFontStyleAttribute = 'face',
    kTXNQDFontColorAttribute = 'klor',
    kTXNTextEncodingAttribute = 'encd',
    kTXNATSUIFontFeaturesAttribute = 'atfe',
    kTXNATSUIFontVariationsAttribute = 'atva',
    kTXNURLAttribute = 'urla'
    kTXNATSUIStyle  = 'astl'
};
```

Constants

`kTXNQDFontNameAttribute`
Specifies that the `data` field of the `TXNTypeAttributes` (page 2718) structure contains a font name.
Available in Mac OS X v10.0 and later.
Declared in `MacTextEditor.h`.

`kTXNQDFontFamilyIDAttribute`
Specifies that the `data` field of the `TXNTypeAttributes` (page 2718) structure contains a font family ID.
Available in Mac OS X v10.0 and later.
Declared in `MacTextEditor.h`.

`kTXNQDFontSizeAttribute`
Specifies that the `data` field of the `TXNTypeAttributes` (page 2718) structure contains a font size. Obsolete; incorrect font sizes are always returned as a fixed value. (**Deprecated.** Use `kTXNFontSizeAttribute` instead.)
Available in Mac OS X v10.0 and later.
Declared in `MacTextEditor.h`.

`kTXNQDFontStyleAttribute`
Specifies that the `data` field of the `TXNTypeAttributes` (page 2718) structure contains a font style.
Available in Mac OS X v10.0 and later.
Declared in `MacTextEditor.h`.

`kTXNQDFontColorAttribute`
Specifies that the `data` field of the `TXNTypeAttributes` (page 2718) structure contains a font color.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNTextEncodingAttribute`
Specifies that the `data` field of the `TXNTypeAttributes` (page 2718) structure contains a text encoding.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNATSUIFontFeaturesAttribute`
Specifies that the `data` field of the `TXNTypeAttributes` (page 2718) structure contains ATSUI font features.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNURLAttribute`
Specifies that the `data` field of the `TXNTypeAttributes` (page 2718) structure contains a URL.

Available in Mac OS X v10.2 and later.

Declared in `MacTextEditor.h`.

`kTXNATSUIStyle`
Specifies that the `data` field of the `TXNTypeAttributes` (page 2718) structure contains an ATSUI style.

Available in Mac OS X v10.3 and later.

Declared in `MacTextEditor.h`.

Discussion
You pass these constants in the `tag` field of the `TXNTypeAttributes` (page 2718) structure. You can supplement these with the style attributes defined for ATSUI.

Declared In
`MacTextEditor.h`

Font Run Attribute Sizes

Describes the size of a font attribute.

```
typedef ByteCount TXNTypeRunAttributeSizes;
enum {
    kTXNQDFontNameAttributeSize = sizeof(Str255),
    kTXNQDFontFamilyIDAttributeSize = sizeof(SInt16),
    kTXNQDFontSizeAttributeSize = sizeof(SInt16),
    kTXNQDFontStyleAttributeSize = sizeof(Style),
    kTXNQDFontColorAttributeSize = sizeof(RGBColor),
    kTXNTextEncodingAttributeSize = sizeof(TextEncoding),
    kTXNFontSizeAttributeSize = sizeof(Fixed),
    kTXNATSUIStyleSize = sizeof(ATSUStyle)
};
```

Constants

`kTXNQDFontNameAttributeSize`
The size of a QuickDraw font name.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNQDFontFamilyIDAttributeSize`
The size of a font family ID attribute.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNQDFontSizeAttributeSize`
Obsolete don't use. (**Deprecated.** Instead, use `kTXNFontSizeAttributeSize`.)

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNQDFontStyleAttributeSize`
The size of font style attribute.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNQDFontColorAttributeSize`
The size of a font color attribute.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNTextEncodingAttributeSize`
The size of text encoding attribute.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNFontSizeAttributeSize`
The size of the font size attribute. Use this instead of the `kTXNQDFontSizeAttributeSize` constant. Font sizes are always returned as a `Fixed` value.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNATSUIStyleSize`
The size of the ATSUI style attribute.

Available in Mac OS X v10.3 and later.

Declared in `MacTextEditor.h`.

Discussion
You pass these constants in the `size` field of the `TXNTypeAttributes` (page 2718) structure.

Declared In
`MacTextEditor.h`

Formatting and Privileges Settings

Specifies the formatting and privileges information to get from or set for a text object.

```
typedef FourCharCode TXNControlTag;
enum {
    kTXNLineDirectionTag = 'lndr',
    kTXNJustificationTag = 'just',
    kTXNIOPrivilegesTag = 'iopv',
    kTXNSelectionStateTag = 'slst',
    kTXNInlineStateTag = 'inst',
    kTXNWordWrapStateTag = 'wwrs',
    kTXNKeyboardSyncStateTag = 'kbsy',
    kTXNAutoIndentStateTag = 'auin',
    kTXNTabSettingsTag = 'tabs',
    kTXNRefConTag = 'rfcn',
    kTXNMarginsTag = 'marg',
    kTXNFlattenMoviesTag = 'flat',
    kTXNDoFontSubstitution = 'fSub',
    kTXNNoUserIOTag = 'nuio',
    kTXNUseCarbonEvents = 'cbcb',
    kTXNDrawCaretWhenInactiveTag = 'dcrt',
    kTXNDrawSelectionWhenInactiveTag = 'dsln',
    kTXNDisableDragAndDropTag = 'drag',
    kTXNSingleLevelUndoTag = 'undo',
    kTXNVisibilityTag = 'visb'
    kTXNDisableLayoutAndDrawTag = kTXNVisibilityTag,
    kTXNAutoScrollBehaviorTag = ‘sbev’
};
```

Constants

`kTXNLineDirectionTag`
Specifies a setting for the direction text is written on the line. If you pass this constant in the `iControlTags` parameter of the `TXNSetTXNObjectControls` (page 2693) function, you must also pass a value that specifies the line direction in the `iControlData` parameter. See Line Direction Settings (page 2757) for a description of possible values.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNJustificationTag`
Specifies a justification setting. If you pass this constant in the `iControlTags` parameter of the `TXNSetTXNObjectControls` (page 2693) function, you must also pass a value that specifies a justification setting in the `iControlData` parameter. See Justification Settings (page 2755) for a description of possible values.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNIOPrivilegesTag`

Indicates a privileges setting. If you pass this constant in the `iControlTags` parameter of the `TXNSetTXNObjectControls` (page 2693) function, you must also pass a value that specifies a privileges setting in the `iControlData` parameter. See Read and Write Privileges Settings (page 2758) for a description of possible values.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNSelectionStateTag`

Specifies a selection state; that is, whether MLTE displays a cursor and allows selections in a read-only document. If you pass this constant in the `iControlTags` parameter of the `TXNSetTXNObjectControls` (page 2693) function, you must also pass a value that specifies the selection state in the `iControlData` parameter. See Selection State Settings (page 2764) for a description of possible values.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNInlineStateTag`

Specifies an inline state that is, whether text is input through the document's window (inline) or through a small floating window that appears at the bottom of the screen. If you pass this constant in the `iControlTags` parameter of the `TXNSetTXNObjectControls` (page 2693) function, you must also pass a value that specifies the inline state in the `iControlData` parameter. See Inline State Settings (page 2755) for a description of possible values.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNWordWrapStateTag`

Specifies a word-wrap setting. If you pass this constant in the `iControlTags` parameter of the `TXNSetTXNObjectControls` (page 2693) function, you must also pass a value that specifies the line-wrapping state in the `iControlData` parameter. See Line Wrapping Settings (page 2758) for a description of possible values.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNKeyboardSyncStateTag`

Specifies whether to synchronize the keyboard with the font setting. If you pass this constant in the `iControlTags` parameter of the `TXNSetTXNObjectControls` (page 2693) function, you must also pass a value that specifies the keyboard synchronization state in the `iControlData` parameter. See Keyboard Synchronization Settings (page 2756) for a description of possible values.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

`kTXNAutoIndentStateTag`

Specifies an automatic indentation setting. This is available only when word warp is turned off. If you pass this constant in the `iControlTags` parameter of the `TXNSetTXNObjectControls` (page 2693) function, you must also pass a value that specifies the indentation state in the `iControlData` parameter. See Automatic Indentation Settings (page 2726) for a description of possible values.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNTabSettingsTag`

Specifies a tab width setting. If you pass this constant in the `iControlTags` parameter of the `TXNSetTXNObjectControls` (page 2693) function, you must also pass a `TXNTab` (page 2716) structure in the `iControlData` parameter.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNRefConTag`

An application-specific constant you define.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNMarginsTag`

Specifies margin settings. If you pass this constant in the `iControlTags` parameter of the `TXNSetTXNObjectControls` (page 2693) function, you must also pass a pointer to a `TXNMargins` structure in the `iControlData` parameter. You use this structure to specify the top, left, and right margin settings.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNFlattenMoviesTag`

Specifies whether to flattens movies. A flattened movie is self-contained. If you don't flatten a movie, it can't be played unless any external files (such as audio or image files) on which it depends are available. If you pass this constant in the `iControlTags` parameter of the `TXNSetTXNObjectControls` (page 2693) function, you must also pass a `Boolean` value in the `iControlData` parameter that specifies whether to enable (`true`) or disable (`false`) movie flattening.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNDoFontSubstitution`

Specifies a font substitution setting. If you pass this constant in the `iControlTags` parameter of the `TXNSetTXNObjectControls` (page 2693) function, you must also pass a `Boolean` value in the `iControlData` parameter that specifies whether to enable (`true`) or disable (`false`) font substitution. For best performance, don't enable font substitution.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNNoUserIOTag`

Specifies an input setting; that is, whether to prevent input typed by the user, but allows your application to use the `TXNSetData` (page 2683) function. Text objects could have read-only with respect to the application user, but have read-and-write privileges with respect to the application. If you pass this constant in the `iControlTags` parameter of the `TXNSetTXNObjectControls` (page 2693) function, you must also pass a value that specifies a read-write setting in the `iControlData` parameter. See Read and Write Privileges Settings (page 2758) for a description of possible values.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNUseCarbonEvents`

Specifies settings for using Carbon events. (**Deprecated.** Use `TXNGetEventTarget` (page 2651) and `TXNSetEventTarget` (page 2686).)

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNDrawCaretWhenInactiveTag`

Specifies settings for drawing the caret when the text object does not have focus. (**Deprecated.** In Mac OS Xv10.4 and later, MLTE never draws the caret when the text object does not have focus.)

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNDrawSelectionWhenInactiveTag`

Specifies settings for drawing the selection when the text object does not have focus.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNDisableDragAndDropTag`

Specifies settings for drag and drop support.

Available in Mac OS X v10.1 and later.

Declared in `MacTextEditor.h`.

`kTXNSingleLevelUndoTag`

Specifies to use a single level of undo.

Available in Mac OS X v10.2 and later.

Declared in `MacTextEditor.h`.

`kTXNVisibilityTag`

Specifies visibility.

Available in Mac OS X v10.2 and later.

Declared in `MacTextEditor.h`.

`kTXNDisableLayoutAndDrawTag`

Specifies visibility. Equivalent to `kTXNVisibilityTag`. Available in Mac OS X v10.4 and later. Use this tag to disable and re-enable layout and drawing. It optimizes performance when adding data incrementally to a text object.

Available in Mac OS X v10.4 and later.

Declared in `MacTextEditor.h`.

`kTXNAutoScrollBehaviorTag`

Specifies autoscroll behavior. Available in Mac OS X v10.4 and later. For constants that represent the various types of autoscrolling, see Automatic Scrolling Behavior (page 2727).

Available in Mac OS X v10.4 and later.

Declared in `MacTextEditor.h`.

Discussion

You pass formatting and privileges settings in the `iControlTags` parameter of the `TXNSetTXNObjectControls` (page 2693) or `TXNGetTXNObjectControls` (page 2659) functions. If you want to clear a setting associated with a control tag, you can set the value of the `iControlData` parameter to Clearance Settings (page 2729).

Declared In
`MacTextEditor.h`

Frame Option Bits

Specify frame options for a text object.

```
enum {
    kTXNDrawGrowIconBit = 0,
    kTXNShowWindowBit = 1,
    kTXNWantHScrollBarBit = 2,
    kTXNWantVScrollBarBit = 3,
    kTXNNoTSMEverBit = 4,
    kTXNReadOnlyBit = 5,
    kTXNNoKeyboardSyncBit = 6,
    kTXNNoSelectionBit = 7,
    kTXNSaveStylesAsSTYLResourceBit = 8,
    kOutputTextInUnicodeEncodingBit = 9,
    kTXNDoNotInstallDragProcsBit = 10,
    kTXNAlwaysWrapAtViewEdgeBit = 11,
    kTXNDontDrawCaretWhenInactiveBit = 12,
    kTXNDontDrawSelectionWhenInactiveBit = 13,
    kTXNSingleLineOnlyBit = 14,
    kTXNDisableDragAndDropBit = 15,
    kTXNUseQDforImagingBit = 16,
    kTXNMonostyledTextBit = 17
};
```

Constants

`kTXNDrawGrowIconBit`

When this bit is set, it indicates the frame will have a size box. The presence of a size box in the lower right corner of an MLTE pane is only useful for resizing an MLTE pane if the MLTE pane occupies the entire window (a full-window MLTE object). In this case your application would look for a mouse-down event in the size box and call the function `TXNGrowWindow` as appropriate. Note that the size box is not supported as a means of resizing MLTE panes using `TXNGrowWindow` for MLTE pane objects.

Passing the `kTXNDrawGrowIconMask` constant to the function `TXNNewObject` only causes a size box to be drawn in the lower right corner of the MLTE pane. Passing this constant does not create a size box control in the window. The window will not contain an actual size box control. This means the window will not receive events that indicate a mouse-down event in the grow region. For this to happen, when you create the window that contains the MLTE pane, you must create the window to have a size box.

In summary, although you may pass the constants `kTXNDrawGrowIconMask` to the function `TXNNewObject` when you create an MLTE object in a window, this action only causes the visual appearance of a size box in the lower right corner of the MLTE pane. If you want to detect mouse-down events in the size box, you must also provide a size box in the window through the appropriate Window Manager functions or other tools.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNShowWindowBit`

When this bit is set, it indicates MLTE should display a window when a text object is created. If this bit is set, your application no longer needs to call the `ShowWindow` function from the Window Manager; MLTE will do this for you.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNWantHScrollBarBit`

When this bit is set, it indicates the frame should have a horizontal scroll bar.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNWantVScrollBarBit`

When this bit is set, it indicates the frame should have a vertical scroll bar.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNNoTSMEverBit`

When this bit is set, it indicates not to use Text Services Manager. You cannot use this bit when your application accepts Unicode input. (**Deprecated.** You can no longer set this because in Mac OS X, MLTE always uses the Text Services Manager.)

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNReadOnlyBit`

When this bit is set, it indicates the text object is read-only. If you set this bit when you call the function `TXNNewObject`, the text object is put into a state that does not allow user input. However, your application can put data into the text object by calling the function `TXNSetData`. If you want the text object set into a more restrictive read-only state that does not allow user input or your application to put data into the text object programmatically you need to call the function `TXNSetTXNObjectControls`, passing the tag `kTXNIOPrivilegesTag`.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNNoKeyboardSyncBit`

When this bit is set, it indicates no keyboard synchronization.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

`kTXNNoSelectionBit`

When this bit is set, it indicates MLTE should not show the insertion point.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNSaveStylesAsSTYLResourceBit`

When this bit is set, it indicates text style should be saved as a `kTXNMultipleStylesPerTextDocumentResType` resource. You can set this to assure compatibility with SimpleText. If you use `kTXNMultipleStylesPerTextDocumentResType` resources to save style info, your documents can have as many styles as you'd like. However tabs are not saved. If you don't set this bit, plain text files are saved as `kTXNSingleStylePerTextDocumentResType` resources, and only the first style in the document is saved. (Your application is expected to apply all style changes to the entire document.) If you save files with a `kTXNSingleStylePerTextDocumentResType` resource, their output is similar to those output by CodeWarrior, BBEdit, and MPW.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kOutputTextInUnicodeEncodingBit`

When this bit is set, it indicates plain text should be saved as Unicode.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNDoNotInstallDragProcsBit`

When this bit is set, it indicates you want to call your own drag handlers.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNAlwaysWrapAtViewEdgeBit`

When this bit is set, it indicates line wrap at the edge of the view rectangle.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNDontDrawCaretWhenInactiveBit`

When this bit is set, it indicates the caret should not be drawn when the object does not have focus. (**Deprecated.** In Mac OS X v10.4, MLTE never draws the caret when the text object does not have focus.)

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNDontDrawSelectionWhenInactiveBit`

When this bit is set, it indicates the selection should not be drawn when the object does not have focus.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNSingleLineOnlyBit`

When this bit is set, it indicates that the text object will not scroll vertically. Horizontal scrolling will stop when the end of the text is visible, and there will be no limit to the width of the text. In addition, a line break character typed, pasted, or dropped into the text object will be translated into a hyphen (-).

Available in Mac OS X v10.1 and later.

Declared in `MacTextEditor.h`.

`kTXNDisableDragAndDropBit`

When this bit is set, it indicates that drag and drop will not be allowed in the text object.

Available in Mac OS X v10.1 and later.

Declared in `MacTextEditor.h`.

`kTXNUseQDforImagingBit`

When this bit is set, it indicates that QuickDraw will be used for imaging instead of Quartz, which is the default. Available in Mac OS X only. (**Deprecated.** You can no longer set the imaging system to use; MLTE always uses Quartz imaging.)

Available in Mac OS X v10.1 and later.

Declared in `MacTextEditor.h`.

`kTXNMonostyledTextBit`

When this bit is set, it indicates that the text object will have a single style no matter what kind of changes are made to the object.

Available in Mac OS X v10.2 and later.

Declared in `MacTextEditor.h`.

Declared In
`MacTextEditor.h`

Frame Option Masks

Represents information about frame behavior (such as whether there are scroll bars and a size box).

```
typedef OptionBits TXNFrameOptions;
enum {
    kTXNDrawGrowIconMask = 1L << kTXNDrawGrowIconBit,
    kTXNShowWindowMask = 1L << kTXNShowWindowBit,
    kTXNWantHScrollBarMask = 1L << kTXNWantHScrollBarBit,
    kTXNWantVScrollBarMask = 1L << kTXNWantVScrollBarBit,
    kTXNNoTSMEverMask = 1L << kTXNNoTSMEverBit,
    kTXNReadOnlyMask = 1L << kTXNReadOnlyBit,
    kTXNNoKeyboardSyncMask = 1L << kTXNNoKeyboardSyncBit,
    kTXNNoSelectionMask = 1L << kTXNNoSelectionBit,
    kTXNSaveStylesAsSTYLResourceMask = 1L <<
kTXNSaveStylesAsSTYLResourceBit,
    kOutputTextInUnicodeEncodingMask = 1L <<
kOutputTextInUnicodeEncodingBit,
    kTXNDoNotInstallDragProcsMask = 1L << kTXNDoNotInstallDragProcsBit,
    kTXNAlwaysWrapAtViewEdgeMask = 1L << kTXNAlwaysWrapAtViewEdgeBit,
    kTXNDontDrawCaretWhenInactiveMask = 1L <<
kTXNDontDrawCaretWhenInactiveBit,
    kTXNDontDrawSelectionWhenInactiveMask = 1L <<
kTXNDontDrawSelectionWhenInactiveBit,
    kTXNSingleLineOnlyMask = 1L << kTXNSingleLineOnlyBit,
    kTXNDisableDragAndDropMask = 1L << kTXNDisableDragAndDropBit,
    kTXNUseQDforImagingMask = 1L << kTXNUseQDforImagingBit,
    kTXNMonostyledTextMask = 1L << kTXNMonostyledTextBit
};
```

Constants

`kTXNDrawGrowIconMask`

Use to set or test for the `kTXNDrawGrowIconBit` bit.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNShowWindowMask`

Use to set or test for the `kTXNShowWindowBit` bit.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNWantHScrollBarMask`

Use to set or test for the `kTXNWantHScrollBarBit` bit.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNWantVScrollBarMask`

Use to set or test for the `kTXNWantVScrollBarBit` bit.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNNoTSMEverMask`

Use to set or test for the `kTXNNoTSMEverBit` bit. (**Deprecated.** You can no longer set this because in Mac OS X, MLTE always uses the Text Services Manager.)

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNReadOnlyMask`
Use to set or test for the `kTXNReadOnlyBit` bit.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNNoKeyboardSyncMask`
Use to set or test for the `kTXNNoKeyboardSyncBit` bit.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

`kTXNNoSelectionMask`
Use to set or test for the `kTXNNoSelectionBit` bit.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNSaveStylesAsSTYLResourceMask`
Use to set or test for the `kTXNSaveStylesAsSTYLResourceBit` bit.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kOutputTextInUnicodeEncodingMask`
Use to set or test for the `kOutputTextInUnicodeEncodingBit` bit.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNDoNotInstallDragProcsMask`
Use to set or test for the `kTXNDoNotInstallDragProcsBit` bit.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNAlwaysWrapAtViewEdgeMask`
Use to set or test for the `kTXNAlwaysWrapAtViewEdgeBit` bit.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNDontDrawCaretWhenInactiveMask`

Use to set or test for the `kTXNDontDrawCaretWhenInactiveBit` bit. (**Deprecated.** In Mac OS Xv10.4 and later, MLTE never draws the caret when the text object does not have focus.)

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNDontDrawSelectionWhenInactiveMask`
Use to set or test for the `kTXNDontDrawSelectionWhenInactiveBit` bit.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNSingleLineOnlyMask`
Use to set or test for the `kTXNSingleLineOnlyBit` bit.

Available in Mac OS X v10.1 and later.

Declared in `MacTextEditor.h`.

`kTXNDisableDragAndDropMask`
Use to set or test for the `kTXNDisableDragAndDropBit` bit.

Available in Mac OS X v10.1 and later.

Declared in `MacTextEditor.h`.

`kTXNUseQDforImagingMask`
Use to set or test for the `kTXNUseQDforImagingBit` bit. (**Deprecated.** You can no longer set the imaging system; MLTE always uses Quartz imaging.)

Available in Mac OS X v10.1 and later.

Declared in `MacTextEditor.h`.

`kTXNMonostyledTextMask`
Use to set or test for the `kTXNMonostyledTextBit` bit.

Available in Mac OS X v10.2 and later.

Declared in `MacTextEditor.h`.

Discussion
See Frame Option Bits (page 2746).

Declared In
`MacTextEditor.h`

HIObject Class ID

Specifies the HIObject class ID for the HITextView class.

```
#define kHITextViewClassID CFSTR("com.apple.HITextView")
```

Constants
`kHITextViewClassID`
The class ID for the HITextView class.

Discussion
For more information on HIView, see the document *Introducing HIView*, available from the Apple Developer Documentation website.

Declared In
`MacTextEditor.h`

HIObject Control Kind

Specifies the HIObject control kind for the HITextView class.

```
enum {
    kControlKindHITextView  = 'hitx'
};
```

Constants
`kControlKindHITextView`
The control kind for the HITextView class.

Available in Mac OS X v10.3 and later.

Declared in `HITextViews.h`.

Discussion
For more information on HIView, see the document *Introducing HIView*, available from the Apple Developer Documentation website.

Declared In
`MacTextEditor.h`

Initialization Option Bits

Specify initialization options for MLTE.

```
enum {
    kTXNWantMoviesBit = 0,
    kTXNWantSoundBit = 1,
    kTXNWantGraphicsBit = 2,
    kTXNAlwaysUseQuickDrawTextBit = 3,
    kTXNUseTemporaryMemoryBit = 4
};
```

Constants

`kTXNWantMoviesBit`
When this bit is set, it specifies that movie data is supported in a text object.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `MacTextEditor.h`.

`kTXNWantSoundBit`
When this bit is set, it specifies that sound data is supported in a text object.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `MacTextEditor.h`.

`kTXNWantGraphicsBit`
When this bit is set, it specifies that graphics data is supported in a text object.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `MacTextEditor.h`.

`kTXNAlwaysUseQuickDrawTextBit`
When this bit is set, it specifies that MLTE should use QuickDraw for imaging even if ATSUI is available. This is often the best choice for applications that need speed and efficient memory usage.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `MacTextEditor.h`.

`kTXNUseTemporaryMemoryBit`
When this bit is set, it specifies that MLTE should use temporary memory for all memory allocations.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `MacTextEditor.h`.

Declared In
MacTextEditor.h

Initialization Option Masks

Represents MLTE initialization options.

```
typedef OptionBits TXNInitOptions;
enum {
    kTXNWantMoviesMask = 1L << kTXNWantMoviesBit,
    kTXNWantSoundMask = 1L << kTXNWantSoundBit,
    kTXNWantGraphicsMask = 1L << kTXNWantGraphicsBit,
    kTXNAlwaysUseQuickDrawTextMask = 1L << kTXNAlwaysUseQuickDrawTextBit,
    kTXNUseTemporaryMemoryMask = 1L << kTXNUseTemporaryMemoryBit
};
```

Constants

`kTXNWantMoviesMask`
Use to set or test for the `kTXNWantMoviesBit` bit.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `MacTextEditor.h`.

`kTXNWantSoundMask`
Use to set or test for the `kTXNWantSoundBit` bit.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `MacTextEditor.h`.

`kTXNWantGraphicsMask`
Use to set or test for the `kTXNWantGraphicsBit` bit.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `MacTextEditor.h`.

`kTXNAlwaysUseQuickDrawTextMask`
Use to set or test for the `kTXNAlwaysUseQuickDrawTextBit` bit.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `MacTextEditor.h`.

`kTXNUseTemporaryMemoryMask`
Use to set or test for the `kTXNUseTemporaryMemoryBit` bit.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `MacTextEditor.h`.

Discussion
Used in the `iUsageFlags` parameter of the `TXNInitTextension` (page 2662) function.

Declared In
MacTextEditor.h

Inline State Settings

Specify the inline state for a text object.

```
enum {
    kTXNUseInline = false,
    kTXNUseBottomline = true
};
```

Constants

`kTXNUseInline`

Text is entered at the caret insertion point in the text object's window.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNUseBottomline`

Text is entered in a bottom-line window (a small floating window that appears at the bottom of the screen).

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

Discussion

Passed in the `iControlData` parameter of the `TXNSetTXNObjectControls` (page 2693) function when the value of the `iControlTags` parameter is `kTXNInlineStateTag`, or returned in the `oControlData` parameter of the `TXNGetTXNObjectControls` (page 2659) function when the value of the `iControlTags` parameter is `kTXNInlineStateTag`.

Declared In

`MacTextEditor.h`

Justification Settings

Specify the justification setting.

```
enum {
    kTXNFlushDefault = 0,
    kTXNFlushLeft = 1,
    kTXNFlushRight = 2,
    kTXNCenter = 4,
    kTXNFullJust = 8,
    kTXNForceFullJust = 16
};
```

Constants

`kTXNFlushDefault`

Justification is flush according to the line direction.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNFlushLeft`

Justification is flush left.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNFlushRight`
Justification is flush right.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNCenter`
Justification is centered.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNFullJust`
Justification is flush left and right for all lines except the last line in a paragraph.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNForceFullJust`
Justification is flush left and right for all lines.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

Discussion
Passed in the `iControlData` parameter of the `TXNSetTXNObjectControls` (page 2693) function when the value of the `iControlTags` parameter is `kTXNJustificationTag`, or returned in the `oControlData` parameter of the `TXNGetTXNObjectControls` (page 2659) function when the value of the `iControlTags` parameter is `kTXNJustificationTag`.

Declared In
`MacTextEditor.h`

Keyboard Synchronization Settings

Specify the keyboard synchronization setting for a text object.

```
enum {
    kTXNSyncKeyboard = false,
    kTXNNoSyncKeyboard = true
};
```

Constants
`kTXNSyncKeyboard`
Keyboard synchronization is enabled.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

`kTXNNoSyncKeyboard`
Keyboard synchronization is not enabled.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

Discussion
These constants are passed in the `iControlData` parameter of the `TXNSetTXNObjectControls` (page 2693) function when the value of the `iControlTags` parameter is `kTXNKeyboardSyncStateTag`. They are also returned in the `oControlData` parameter of the `TXNGetTXNObjectControls` (page 2659) function when the value of the `iControlTags` parameter is `kTXNKeyboardSyncStateTag`.

Declared In
`MacTextEditor.h`

Layout and Draw Constants

Specify the layout and draw setting for a text object.

```
enum {
    kTXNEnableLayoutAndDraw = false,
    kTXNDisableLayoutAndDraw = true
};
```

Constants
`kTXNEnableLayoutAndDraw`
Layout and draw is enabled.
Available in Mac OS X v10.4 and later.
Declared in `MacTextEditor.h`.

`kTXNDisableLayoutAndDraw`
Layout and draw is disabled.
Available in Mac OS X v10.4 and later.
Declared in `MacTextEditor.h`.

Discussion
These constants are passed in the `iControlData` parameter of the `TXNSetTXNObjectControls` (page 2693) function when the value of the `iControlTags` parameter is `kTXNDisableLayoutAndDrawTag`. They are also are returned in the `oControlData` parameter of the `TXNGetTXNObjectControls` (page 2659) function when the value of the `iControlTags` parameter is `kTXNDisableLayoutAndDrawTag`.

Declared In
`MacTextEditor.h`

Line Direction Settings

Specify the line direction setting.

```
enum {
    kTXNLeftToRight = 0,
    kTXNRightToLeft = 1
};
```

Constants
`kTXNLeftToRight`
Line direction flows from left to right.
Available in Mac OS X v10.0 and later.
Declared in `MacTextEditor.h`.

`kTXNRightToLeft`
Line direction flows from right to left.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

Discussion
Passed in the `iControlData` parameter of the `TXNSetTXNObjectControls` (page 2693) function when the value of the `iControlTags` parameter is `kTXNLineDirectionTag`, or returned in the `oControlData` parameter of the `TXNGetTXNObjectControls` (page 2659) function when the value of the `iControlTags` parameter is `kTXNLineDirectionTag`.

Declared In
`MacTextEditor.h`

Line Wrapping Settings

Specify the line-wrap setting for a text object.

```
enum {
    kTXNAutoWrap = false,
    kTXNNoAutoWrap = true
};
```

Constants
`kTXNAutoWrap`
Automatic line wrapping is enabled.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNNoAutoWrap`
Automatic line wrapping is not enabled.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

Discussion
Passed in the `iControlData` parameter of the `TXNSetTXNObjectControls` (page 2693) function when the value of the `iControlTags` parameter is `kTXNWordWrapStateTag`, or returned in the `oControlData` parameter of the `TXNGetTXNObjectControls` (page 2659) function when the value of the `iControlTags` parameter is `kTXNWordWrapStateTag`.

Declared In
`MacTextEditor.h`

Read and Write Privileges Settings

Specify the privileges setting for a text object.

```
enum {
    kTXNReadWrite = false,
    kTXNReadOnly = true
};
```

Constants

`kTXNReadWrite`
The document has read and write privileges.
Available in Mac OS X v10.0 and later.
Declared in `MacTextEditor.h`.

`kTXNReadOnly`
The document is read-only.
Available in Mac OS X v10.0 and later.
Declared in `MacTextEditor.h`.

Discussion

Passed in the `iControlData` parameter of the `TXNSetTXNObjectControls` (page 2693) function when the value of the `iControlTags` parameter is `kTXNIOPrivilegesTag`, or returned in the `oControlData` parameter of the `TXNGetTXNObjectControls` (page 2659) function when the value of the `iControlTags` parameter is `kTXNIOPrivilegesTag`.

Declared In

`MacTextEditor.h`

Rectangle Keys

Specifies the bounds to use for a text object.

```
typedef UInt32    TXNRectKey;
enum {
    kTXNViewRectKey               = 0,
    kTXNDestinationRectKey        = 1,
    kTXNTextRectKey               = 2,
    kTXNVerticalScrollBarRectKey  = 3,
    kTXNHorizontalScrollBarRectKey = 4
};
```

Constants

`kTXNViewRectKey`
Specifies to use the view rectangle.
Available in Mac OS X v10.3 and later.
Declared in `MacTextEditor.h`.

`kTXNDestinationRectKey`
Specifies to use the destination rectangle.
Available in Mac OS X v10.3 and later.
Declared in `MacTextEditor.h`.

`kTXNTextRectKey`
Specifies to use the text rectangle.
Available in Mac OS X v10.3 and later.
Declared in `MacTextEditor.h`.

`kTXNVerticalScrollBarRectKey`
Specifies to include the vertical scroll bar in the rectangle.

Available in Mac OS X v10.3 and later.

Declared in `MacTextEditor.h`.

`kTXNHorizontalScrollBarRectKey`
Specifies to include the horizontal scroll bar in the rectangle.

Available in Mac OS X v10.3 and later.

Declared in `MacTextEditor.h`.

Discussion
You can pass a rectangle key to the function `TXNGetHIRect` (page 2653).

Declared In
`MacTextEditor.h`

Scroll Bar Orientation

Specifies the orientation of a text window's scroll bar.

```
typedef UInt32 TXNScrollBarOrientation;
enum {
    kTXNHorizontal = 0,
    kTXNVertical = 1
};
```

Constants
`kTXNHorizontal`
Specifies a horizontal scroll bar.

Available in Mac OS X v10.2 and later.

Declared in `MacTextEditor.h`.

`kTXNVertical`
Specifies a vertical scroll bar.

Available in Mac OS X v10.2 and later.

Declared in `MacTextEditor.h`.

Discussion
You use these constants in your `TXNScrollInfoProcPtr` (page 2705) callback function.

Declared In
`MacTextEditor.h`

Scroll Bar States

Represents the scroll bar state for the window attached to a text object.

```
typedef Boolean TXNScrollBarState;
enum {
    kScrollBarsAlwaysActive = true,
    kScrollBarsSyncWithFocus = false
};
```

Constants

`kScrollBarsAlwaysActive`
Indicates that scroll bars should always appear in the active state even then the text area does not have focus.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kScrollBarsSyncWithFocus`
Indicates that scroll bars should be active only if the frame in which they are enclosed is focused.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

Discussion
Used in the `TXNActivate` (page 2618) function.

Declared In
`MacTextEditor.h`

Scroll Units

Specify the unit by which scrolling should occur.

```
typedef UInt32 TXNScrollUnit;
enum {
    kTXNScrollUnitsInPixels = 0,
    kTXNScrollUnitsInLines = 1,
    kTXNScrollUnitsInViewRects = 2
};
```

Constants

`kTXNScrollUnitsInPixels`
Specifies pixels as the scrolling unit.

Available in Mac OS X v10.2 and later.

Declared in `MacTextEditor.h`.

`kTXNScrollUnitsInLines`
Specifies line count as the scrolling unit. Scrolling is slower when you use this unit because each line must be measured by MLTE before the text scrolls.

Available in Mac OS X v10.2 and later.

Declared in `MacTextEditor.h`.

`kTXNScrollUnitsInViewRects`
Specifies the height of the current view rectangle as the scrolling unit.

Available in Mac OS X v10.2 and later.

Declared in `MacTextEditor.h`.

Discussion
These constants are supplied as the `iVerticalScrollUnit` and `iHorizontalScrollUnit` parameters to the `TXNScroll` (page 2679) function. They specifies the units used for each of these parameters.

Declared In
`MacTextEditor.h`

Search Criteria Bits

Specify matching rules to use when searching.

```
enum {
    kTXNIgnoreCaseBit = 0,
    kTXNEntireWordBit = 1,
    kTXNUseEncodingWordRulesBit = 31
};
```

Constants

`kTXNIgnoreCaseBit`
When this bit is set, indicates that case should be ignored.
Available in Mac OS X v10.0 and later.
Declared in `MacTextEditor.h`.

`kTXNEntireWordBit`
When this bit is set, indicates that the entire word must match.
Available in Mac OS X v10.0 and later.
Declared in `MacTextEditor.h`.

`kTXNUseEncodingWordRulesBit`
When this bit is set, indicates that Unicode Utilities should be used to find a word boundary. You need to set this bit if your applications uses 2-byte scripts in which words are not separated by a space.
Available in Mac OS X v10.0 and later.
Declared in `MacTextEditor.h`.

Declared In
`MacTextEditor.h`

Search Criteria Masks

Represents the matching rules to be used in a find operation.

```
typedef OptionBits TXNMatchOptions;
enum {
    kTXNIgnoreCaseMask = 1L << kTXNIgnoreCaseBit,
    kTXNEntireWordMask = 1L << kTXNEntireWordBit,
    kTXNUseEncodingWordRulesMask = 1L << kTXNUseEncodingWordRulesBit
};
```

Constants

`kTXNIgnoreCaseMask`
Use to set or test for the `kTXNIgnoreCaseBit` bit.
Available in Mac OS X v10.0 and later.
Declared in `MacTextEditor.h`.

`kTXNEntireWordMask`
Use to set or test for the `kTXNEntireWordBit` bit.
Available in Mac OS X v10.0 and later.
Declared in `MacTextEditor.h`.

`kTXNUseEncodingWordRulesMask`
Use to set or test for the `kTXNEntireWordBit` bit.
Available in Mac OS X v10.0 and later.
Declared in `MacTextEditor.h`.

Discussion

These constants can be passed in the `iMatchOptions` parameter of the `TXNFind` (page 2641) function or the callback `TXNFindProcPtr` (page 2703).

Declared In

`MacTextEditor.h`

Selection Display Settings

Specify whether the text object should scroll to show the beginning or the end of the selection.

```
enum {
    kTXNShowStart = false,
    kTXNShowEnd = true
};
```

Constants

`kTXNShowStart`
The start of the selection should be shown. The selection scrolls to show the start if necessary.
Available in Mac OS X v10.0 and later.
Declared in `MacTextEditor.h`.

`kTXNShowEnd`
The end of the selection should be shown. The selection scrolls to show the end if necessary.
Available in Mac OS X v10.0 and later.
Declared in `MacTextEditor.h`.

Declared In

`MacTextEditor.h`

Selection State Settings

Specify whether or not MLTE displays a caret and allows selections in text that is read-only.

```
enum {
    kTXNSelectionOn = true,
    kTXNSelectionOff = false
};
```

Constants

`kTXNSelectionOn`

Display a caret and allow a selection in text that is read-only.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNSelectionOff`

Do not display a caret or allow a selection in text that is read-only.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

Discussion

Passed in the `iControlData` parameter of the `TXNSetTXNObjectControls` (page 2693) function when the value of the `iControlTags` parameter is `kTXNSelectionStateTag`, or returned in the `oControlData` parameter of the `TXNGetTXNObjectControls` (page 2659) function when the value of the `iControlTags` parameter is `kTXNSelectionStateTag`.

Declared In

`MacTextEditor.h`

Style Resource Types

Specify the resource type to use to save style information for a plain text document.

```
enum {
    kTXNSingleStylePerTextDocumentResType = 'MPSR',
    kTXNMultipleStylesPerTextDocumentResType = 'styl'
};
```

Constants

`kTXNSingleStylePerTextDocumentResType`

User for a document that contains a single style and should be treated as a BBEdit, MPW, or CodeWarrior document.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNMultipleStylesPerTextDocumentResType`

Use for a document that contains multiple styles and should be treated as a SimpleText document.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

Discussion

Used in the `TXNNewObject` (page 2667) function.

Declared In
MacTextEditor.h

Supported Data Types

Specifies the type of data being requested from or passed to an MLTE function.

```
typedef OSType TXNDataType;
enum {
    kTXNTextData = 'TEXT',
    kTXNPictureData = 'PICT',
    kTXNMovieData = 'moov',
    kTXNSoundData = 'snd ',
    kTXNUnicodeTextData = 'utxt'
};
```

Constants

`kTXNTextData`
Text data.
Available in Mac OS X v10.0 and later.
Declared in `MacTextEditor.h`.

`kTXNPictureData`
Graphics (`PICT`) data.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `MacTextEditor.h`.

`kTXNMovieData`
Movie or sound data.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `MacTextEditor.h`.

`kTXNSoundData`
Sound data.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `MacTextEditor.h`.

`kTXNUnicodeTextData`
Unicode text data.
Available in Mac OS X v10.0 and later.
Declared in `MacTextEditor.h`.

Discussion
Used in the `TXNGetDataEncoded` (page 2650) function.

Declared In
MacTextEditor.h

Supported File Types

Represents a file type.

```
typedef OSType TXNFileType;
enum {
    kTXNTextensionFile = 'txtn',
    kTXNTextFile = 'TEXT',
    kTXNPictureFile = 'PICT',
    kTXNMovieFile = 'MooV',
    kTXNSoundFile = 'sfil',
    kTXNAIFFFile = 'AIFF',
    kTXNUnicodeTextFile = 'utxt'
};
```

Constants

`kTXNTextensionFile`

A file that contains Unicode or Mac OS text. By default, it contains Unicode text. Files are saved in a private format.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNTextFile`

A file that contains plain text data.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNPictureFile`

A file that contains graphics data in `PICT` format.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

`kTXNMovieFile`

A file that contains movie data in `'MooV'` format.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

`kTXNSoundFile`

A file that contains sound data in `'sfil'` format.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

`kTXNAIFFFile`

A file that contains sound data in `'aiff'` format.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNUnicodeTextFile`
A file that contains Unicode text data.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

Declared In
`MacTextEditor.h`

Supported Frame Types

Represents a frame type. (**Deprecated.** No longer needed.)

```
typedef UInt32 TXNFrameType;
enum {
    kTXNTextEditStyleFrameType = 1,
    kTXNPageFrameType = 2,
    kTXNMultipleFrameType = 3
};
```

Constants
`kTXNTextEditStyleFrameType`
A single rectangle that allows text to scroll if the rectangle fills. Although you can pass this as a parameter to the function `TXNNewObject` (page 2667), you should instead use the function `TXNCreateObject` (page 2630), which does not require a frame type.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNPageFrameType`
A single rectangle with a bottom. That is, text moves to a new page if the frame is full. This constant is not supported in Mac OS X version 10.3 and later.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNMultipleFrameType`
Multiple frames. This constant is not supported in Mac OS X version 10.3 and later.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

Special Considerations

This data type is used only by the `TXNNewObject` function, which is deprecated.

Declared In
`MacTextEditor.h`

Tab Types

Defines the tab settings for a text object.

```
typedef SInt8 TXNTabType;
enum {
    kTXNRightTab = -1,
    kTXNLeftTab = 0,
    kTXNCenterTab = 1
};
```

Constants

`kTXNRightTab`
Right tabs are active.
Available in Mac OS X v10.0 and later.
Declared in `MacTextEditor.h`.

`kTXNLeftTab`
Left tabs are active; not available in MLTE version 1.0.
Available in Mac OS X v10.0 and later.
Declared in `MacTextEditor.h`.

`kTXNCenterTab`
Center tabs are active; not available in MLTE version 1.0.
Available in Mac OS X v10.0 and later.
Declared in `MacTextEditor.h`.

Discussion
Used in the `TXNTab` (page 2716) structure.

Declared In
`MacTextEditor.h`

Text Background Types

Represents a background data type used in the `TXNBackground` structure.

```
typedef UInt32 TXNBackgroundType;
enum {
    kTXNBackgroundTypeRGB = 1
};
```

Constants

`kTXNBackgroundTypeRGB`
Indicates color.
Available in Mac OS X v10.0 and later.
Declared in `MacTextEditor.h`.

Discussion
Used in the `TXNBackground` (page 2708) structure. MLTE supports only color as the background type.

Declared In
`MacTextEditor.h`

Text Box Options Bits

Specify how text should be displayed in a Unicode text box.

```
enum {
    kTXNSetFlushnessBit = 0,
    kTXNSetJustificationBit = 1,
    kTXNUseFontFallBackBit = 2,
    kTXNRotateTextBit = 3,
    kTXNUseVerticalTextBit = 4,
    kTXNDontUpdateBoxRectBit = 5,
    kTXNDontDrawTextBit = 6,
    kTXNUseCGContextRefBit = 7,
    kTXNImageWithQDBit = 8,
    kTXNDontWrapTextBit = 9
};
```

Constants

`kTXNSetFlushnessBit`

When this bit is set, indicates MLTE should display text flush according to the line direction.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

`kTXNSetJustificationBit`

When this bit is set, indicates justification. Text is justified in the direction of the display. Horizontal text is justified horizontally, but not vertically. Vertical text is justified vertically, but not horizontally.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

`kTXNUseFontFallBackBit`

When this bit is set, indicates MLTE should use ATSUI transient font matching that searches for a font that has a matching character.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

`kTXNRotateTextBit`

When this bit is set, indicates MLTE should display text rotated clockwise.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

`kTXNUseVerticalTextBit`

When this bit is set, indicates MLTE should display text vertically from top to bottom.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

`kTXNDontUpdateBoxRectBit`
When this bit is set, indicates MLTE should not update the specified rectangle. If you set this bit when you call the `TXNDrawUnicodeTextBox` (page 2639) function, the function does not update the right coordinates of the specified rectangle to accommodate the longest line for text.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

`kTXNDontDrawTextBit`
When this bit is set, indicates MLTE should return the size of the text but should not draw the text box.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

`kTXNUseCGContextRefBit`
When this bit is set, indicates MLTE should the Quartz context (`CGContext`) you provide instead of the temporary `CGContextRef` created internally by MLTE. To do so, you must set the `kTXNUseCGContextRefBit` bit in `TXNTextBoxOptions` and pass a `CGContextRef` in the `options` field of the `TXNTextBoxOptionsData` structure.

Available in Mac OS X v10.1 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

`kTXNImageWithQDBit`
When this bit is set, indicates MLTE should use QuickDraw for imaging text. (**Deprecated.** You can no longer set the imaging system; MLTE always uses Quartz imaging.)

Available in Mac OS X v10.1 and later.

Declared in `MacTextEditor.h`.

`kTXNDontWrapTextBit`
When this bit is set, indicates MLTE should not wrap text.

Available in Mac OS X v10.1 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

Declared In
`MacTextEditor.h`

Text Box Options Masks

Defines how text appears in a text box.

```
typedef OptionBits TXNTextBoxOptions;
enum {
    kTXNSetFlushnessMask = 1L << kTXNSetFlushnessBit,
    kTXNSetJustificationMask = 1L << kTXNSetJustificationBit,
    kTXNUseFontFallBackMask = 1L << kTXNUseFontFallBackBit,
    kTXNRotateTextMask = 1L << kTXNRotateTextBit,
    kTXNUseVerticalTextMask = 1L << kTXNUseVerticalTextBit,
    kTXNDontUpdateBoxRectMask = 1L << kTXNDontUpdateBoxRectBit,
    kTXNDontDrawTextMask = 1L << kTXNDontDrawTextBit,
    kTXNUseCGContextRefMask = 1L << kTXNUseCGContextRefBit,
    kTXNImageWithQDMask = 1L << kTXNImageWithQDBit,
    kTXNDontWrapTextMask = 1L << kTXNDontWrapTextBit
};
```

Constants

`kTXNSetFlushnessMask`
Use to set or test for the `kTXNSetFlushnessBit` bit.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `MacTextEditor.h`.

`kTXNSetJustificationMask`
Use to set or test for the `kTXNSetJustificationBit` bit.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `MacTextEditor.h`.

`kTXNUseFontFallBackMask`
Use to set or test for the `kTXNUseFontFallBackBit` bit.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `MacTextEditor.h`.

`kTXNRotateTextMask`
Use to set or test for the `kTXNRotateTextBit` bit.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `MacTextEditor.h`.

`kTXNUseVerticalTextMask`
Use to set or test for the `kTXNUseVerticalTextBit` bit.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `MacTextEditor.h`.

`kTXNDontUpdateBoxRectMask`
Use to set or test for the `kTXNDontUpdateBoxRectBit` bit.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `MacTextEditor.h`.

`kTXNDontDrawTextMask`

Use to set or test for the `kTXNDontDrawTextBit` bit.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

`kTXNUseCGContextRefMask`

Use to set or test for `kTXNUseCGContextRefBit` bit.

Available in Mac OS X v10.1 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

`kTXNImageWithQDMask`

Use to set or test for `kTXNImageWithQDBit` bit. (**Deprecated.** You can no longer set the imaging system; MLTE always uses Quartz imaging.)

Available in Mac OS X v10.1 and later.

Declared in `MacTextEditor.h`.

`kTXNDontWrapTextMask`

Use to set or test for `kTXNDontWrapTextBit` bit.

Available in Mac OS X v10.1 and later.

Not available to 64-bit applications.

Declared in `MacTextEditor.h`.

Discussion

Used in the `TXNDrawUnicodeTextBox` (page 2639) and `TXNDrawCFStringTextBox` (page 2637) functions.

Declared In

`MacTextEditor.h`

Text Encoding Preferences

Represents how to encode text for your application.

```
typedef UInt32 TXNPermanentTextEncodingType;
enum {
    kTXNSystemDefaultEncoding = 0,
    kTXNMacOSEncoding = 1,
    kTXNUnicodeEncoding = 2
};
```

Constants

`kTXNSystemDefaultEncoding`

Use the encoding that is used internally by MLTE and the system. The preferred encoding is Unicode for a system that has ATSUI.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNMacOSEncoding`

Incoming and outgoing text should be in traditional Mac OS script system encodings even if MLTE uses another format internally. MLTE will use the Text Encoding Convertor (TEC) to convert text and offsets to match your application's preference.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

`kTXNUnicodeEncoding`

Incoming and outgoing text should be in Unicode even on systems that do not have ATSUI. MLTE will use the Text Encoding Convertor (TEC) to convert text and offsets to match the applications preference.

Available in Mac OS X v10.0 and later.

Declared in `MacTextEditor.h`.

Discussion
These convenience constants can be used in the functions `TXNNewObject` (page 2667) and `TXNSave` (page 2677).

Declared In
`MacTextEditor.h`

Result Codes

The most common result codes returned by MLTE are listed below.

Result Code	Value	Description
`kTXNEndIterationErr`	-22000	Function was not able to iterate through the data contained by a text object. Available in Mac OS X v10.0 and later.
`kTXNCannotAddFrameErr`	-22001	Frame was not added. The multiple-frame feature is currently not available in MLTE, so this error is returned any time you try to define an object with multiple frames. Available in Mac OS X v10.0 and later.
`kTXNInvalidFrameIDErr`	-22002	The frame ID is invalid. Available in Mac OS X v10.0 and later.
`kTXNIllegalToCrossDataBoundariesErr`	-22003	Offsets specify a range that crosses a data type boundary. Available in Mac OS X v10.0 and later.
`kTXNUserCanceledOperationErr`	-22004	A user canceled an operation before your application completed processing it. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
kTXNBadDefaultFileTypeWarning	-22005	Text file is not in the format you specified. Available in Mac OS X v10.0 and later.
kTXNCannotSetAutoIndentErr	-22006	Automatic indenting could not be enabled—the document has word wrapping enabled and you tried to enable auto indentation. Available in Mac OS X v10.0 and later.
kTXNRunIndexOutofBoundsErr	-22007	An index you supplied to a function is out of bounds. Available in Mac OS X v10.0 and later.
kTXNNoMatchErr	-22008	Returned by TXNFind (page 2641) when a match is not found. Available in Mac OS X v10.0 and later.
kTXNAttributeTagInvalidForRunErr	-22009	Tag for a specific run is not valid. Available in Mac OS X v10.0 and later.
kTXNSomeOrAllTagsInvalidForRunErr	-22010	Tags supplied to a function are not valid. Available in Mac OS X v10.0 and later.
kTXNInvalidRunIndex	-22011	Index is out of range for that run. Available in Mac OS X v10.0 and later.
kTXNAlreadyInitializedErr	-22012	You already called the TXNInitTextension (page 2662) function. Available in Mac OS X v10.0 and later.
kTXNCannotTurnTSMOffWhenUsingUnicodeErr	-22013	Your application tried to turn off the Text Services Manager while MLTE was set to process Unicode. Available in Mac OS X v10.0 and later.
kTXNCopyNotAllowedInEchoModeErr	-22014	Your application tried to copy text that was in echo mode. Available in Mac OS X v10.0 and later.
kTXNDataTypeNotAllowedErr	-22015	Your applications specifies a data type that MTLE does not allow. Available in Mac OS X v10.0 and later.
kTXNATSUIIsNotInstalledErr	-22016	Indicates ATSUI is not installed on the system. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
kTXNOutsideOfLineErr	-22017	Indicates a value that is beyond the length of the line. Available in Mac OS X v10.0 and later.
kTXNOutsideOfFrameErr	-22018	Indicates a value that is outside of the text object's frame. Available in Mac OS X v10.0 and later.
kTXNDisabledFunctionalityErr	-22019	Indicates the function has been disabled. Available in Mac OS X v10.3 and later.
kTXNOperationNotAllowedErr	-22020	Indicates that the function cannot be called in this context. Available in Mac OS X v10.4 and later.

Open Scripting Architecture Reference

Framework:	Carbon/Carbon.h
Declared in	ASDebugging.h AppleScript.h ASRegistry.h OSAComp.h OSA.h OSAGeneric.h

Overview

The Open Scripting Architecture (OSA) provides a standard and extensible mechanism for interapplication communication in Mac OS X. It provides support for creating scriptable applications and for writing scripting components to implement scripting languages. Every Mac OS X system includes the AppleScript component, which implements AppleScript, the standard scripting language defined by Apple. However, developers can write scripting components for additional scripting languages. For conceptual information on the OSA, see "Open Scripting Architecture" in AppleScript Overview.

You need to use this reference if you are writing a scripting component or if your application needs to interact with scripting components to manipulate and execute scripts. The API described in this document is implemented by the OpenScripting framework, a subframework of the Carbon framework. For information about working with components, see Scripting Components in Inside Macintosh: Interapplication Communication.

Important: Do not rely on the API descriptions in Interapplication Communication—*Open Scripting Architecture Reference* provides the current API documentation.

The Apple Event Manager, another part of the OSA, is implemented primarily by the AE framework, a subframework of the Application Services framework, and is documented in *Apple Event Manager Reference* and *Apple Events Programming Guide*. Applications use the Apple Event Manager to send and respond to Apple events and to make their operations and data available to AppleScript scripts.

Functions by Task

Saving and Loading Script Data

`OSALoad` (page 2832)
Loads script data.

`OSALoadFile` (page 2835)
Loads a script from the specified file into the specified scripting component, compiling the script if the file is a text file.

`OSAStore` (page 2850)
Gets a handle to script data in the form of a storage descriptor record.

`OSAStoreFile` (page 2851)
Stores a script into the specified file.

Executing and Disposing of Scripts

To execute a script, your application must first obtain a valid script ID for a compiled script or script context. You can use either the `OSALoad` function or the optional `OSACompile` function to obtain a script ID.

`OSAExecute` (page 2816)
Executes a compiled script or a script context.

`OSAScriptError` (page 2839)
Gets information about errors that occur during script execution.

`OSADispose` (page 2811)
Reclaims the memory occupied by script data.

Setting and Getting Script Information

`OSASetScriptInfo` (page 2846)
Sets information about script data according to the value you pass in the selector parameter.

`OSAGetScriptInfo` (page 2827)
Obtains information about script data according to the value you pass in the `selector` parameter.

Manipulating the Active Function

`OSASetActiveProc` (page 2840)
Sets the active function that a scripting component calls periodically while executing a script.

`OSAGetActiveProc` (page 2819)
Gets a pointer to the active function that a scripting component is currently using.

Compiling Scripts

Scripting components can provide three optional functions that get the name of a scripting component, compile a script, and update a script ID. A scripting component that supports the functions in this section has the `kOSASupportsCompiling` bit set in the `componentFlags` field of its component description record.

`OSAScriptingComponentName` (page 2840)
Gets the name of a scripting component.

`OSACompile` (page 2799)
Compiles the source data for a script and obtain a script ID for a compiled script or a script context.

`OSACopyID` (page 2802)
Updates script data after editing or recording and to perform undo or revert operations on script data.

Getting Source Data

`OSAGetSource` (page 2830)
Decompiles the script data identified by a script ID and obtains the equivalent source data.

`OSADisplay` (page 2810)
Converts a script value to text. Your application can then use its own functions to display this text to the user.

`OSACopyDisplayString` (page 2801)
Converts a script value to an attributed Unicode text string, which your application can display to the user.

`OSACopySourceString` (page 2804)
Decompiles the script data for the specified script and returns a copy of the equivalent source data as an attributed Unicode text string.

Coercing Script Values

Scripting components can provide support for two optional functions which coerce data in a descriptor record to a script value and coerce a script value to data in a descriptor record. A scripting component that supports the functions in this section has the `kOSASupportsAECoercion` bit set in the `componentFlags` field of its component description record.

`OSACoerceFromDesc` (page 2798)
Obtains the script ID for a script value that corresponds to the data in a descriptor record.

`OSACoerceToDesc` (page 2798)
Coerces a script value to a descriptor record of a desired descriptor type.

Manipulating the Create and Send Functions

Some scripting components provide functions that allow your application to set or get pointers to the create and send functions used by the scripting component when it sends and creates Apple events during script execution. If you do not set the pointers that specify these functions, the scripting component uses the standard `AECreateAppleEvent` and `AESend` functions with default parameters. A scripting component that supports the functions described in this section has the `kOSASupportsAESending` bit set in the `componentFlags` field of its component description record.

`OSASetCreateProc` (page 2841)
Specifies a create function that a scripting component should use instead of the Apple Event Manager's `AECreateAppleEvent` function when creating Apple events.

`OSAGetCreateProc` (page 2820)
Gets a pointer to the create function that a scripting component is currently using to create Apple events.

`OSASetSendProc` (page 2847)
Specifies a send function that a scripting component should use instead of the Apple Event Manger's `AESend` function when sending Apple events.

`OSAGetSendProc` (page 2829)
Gets a pointer to the send function that a scripting component is currently using.

`OSASetDefaultTarget` (page 2843)
Sets the default target application for Apple events.

Recording Scripts

Script editors use these functions to allow users to control recording. Any application can use these functions to provide its own script-recording interface. A scripting component that supports the functions described in this section has the `kOSASupportsRecording` bit set in the `componentFlags` field of its component description record.

`OSAStartRecording` (page 2848)
Turns on Apple event recording and records subsequent Apple events in a compiled script.

`OSAStopRecording` (page 2849)
Turns off Apple event recording.

Executing Scripts in One Step

You can use these functions if you know that the script data to be executed will be executed only once. A scripting component that supports the functions described in this section has the `kOSASupportsConvenience` bit set in the `componentFlags` field of its component description record.

`OSACompileExecute` (page 2800)
Compiles and executes a script in a single step rather than calling `OSACompile` and `OSAExecute`.

`OSADoScript` (page 2813)
Compiles and executes a script and converts the resulting script value to text in a single step rather than calling `OSACompile`, `OSAExecute`, and `OSADisplay`.

`OSADoScriptFile` (page 2815)
Loads a script from the specified file, compiles the script if the file is a text file, executes the script, converts the resulting script value to text, and stores the script back into the file if the script has persistent properties and the file is not a text file.

`OSALoadExecute` (page 2833)
Loads and executes a script in a single step rather than calling `OSALoad` and `OSAExecute`.

`OSALoadExecuteFile` (page 2834)
Loads a script from the specified file into the specified scripting component, compiles the script if the file is a text file, and executes the script.

Copying a Scripting Dictionary as a Scripting Definition File

`OSACopyScriptingDefinition` (page 2803)
Creates a copy of a scripting definition (sdef) from the specified file or bundle.

Manipulating Dialects

Scripting components that provide several dialects may provide five functions that allow you to switch between dialects dynamically and get information about currently available dialects. The codes for specific dialects are provided by the scripting component. A scripting component that supports the functions described in this section has the `kOSASupportsDialects` bit set in the `componentFlags` field of its component description record.

`OSASetCurrentDialect` (page 2842)
: Sets the current dialect for a scripting component.

`OSAGetCurrentDialect` (page 2821)
: Gets the dialect code for the dialect currently being used by a scripting component.

`OSAAvailableDialectCodeList` (page 2796)
: Obtains a descriptor list containing dialect codes for each of a scripting component's currently available dialects.

`OSAGetDialectInfo` (page 2822)
: Gets information about a specified dialect provided by a specified scripting component.

`OSAAvailableDialects` (page 2797)
: Obtains a descriptor list containing information about each of the currently available dialects for a scripting component.

Using Script Contexts to Handle Apple Events

The optional functions described in this section allow your application to use script contexts to handle Apple events. One way to do this is to install a general Apple event handler in your application's special handler dispatch table. The general Apple event handler provides initial handling for every Apple event received by your application. A scripting component that supports the functions described in this section has the `kOSASupportsEventHandling` bit set in the `componentFlags` field of its component description record.

`OSASetResumeDispatchProc` (page 2845)
: Sets the resume dispatch function called by a scripting component during execution of an AppleScript `continue` statement or its equivalent.

`OSAGetResumeDispatchProc` (page 2826)
: Gets the resume dispatch function currently being used by a scripting component instance during execution of an AppleScript `continue` statement or its equivalent

`OSAExecuteEvent` (page 2817)
: Handles an Apple event with the aid of a script context and obtains a script ID for the resulting script value.

`OSADoEvent` (page 2812)
: Handles an Apple event with the aid of a script context and obtains a reply event.

`OSAMakeContext` (page 2836)
: Gets a script ID for a new script context.

Initializing AppleScript

`ASInit` (page 2788)
: Initializes the AppleScript component.

Getting and Setting Styles for Source Data

`ASCopySourceAttributes` (page 2785)
Gets the current text style attributes AppleScript uses to display script text.

`ASSetSourceAttributes` (page 2791)
Sets the text style attributes used by the AppleScript component to display scripts.

`ASGetSourceStyleNames` (page 2787)
Obtains a list of style names that are each formatted according to the script format styles currently used by the AppleScript component.

Getting and Setting the Default Scripting Component

The default scripting component for any instance of the generic scripting component is initially AppleScript, but you can change it if necessary.

`OSAGetDefaultScriptingComponent` (page 2822)
Gets the subtype code for the default scripting component associated with an instance of the generic scripting component.

`OSASetDefaultScriptingComponent` (page 2842)
Sets the default scripting component associated with an instance of the generic scripting component.

Using Component-Specific Routines

You can't use the generic scripting component to call a component-specific routine. Instead, you must use an instance of the specific scripting component that supports the routine.

To facilitate the use of component-specific routines, the generic scripting component allows you to identify the scripting component that created stored script data, get an instance of a specified scripting component, and convert between generic script IDs and component-specific script IDs.

`OSAGetScriptingComponentFromStored` (page 2829)
Gets the subtype code for a scripting component that created a storage descriptor record.

`OSAGetScriptingComponent` (page 2828)
Gets the instance of a scripting component for a specified subtype.

`OSAGenericToRealID` (page 2818)
Converts a generic script ID to the corresponding component-specific script ID.

`OSARealToGenericID` (page 2837)
Converts a component-specific script ID to the corresponding generic script ID.

Manipulating Trailers for Generic Storage Descriptor Records

All scripting components must use the `OSAGetStorageType`, `OSAAddStorageType`, and `OSARemoveStorageType` functions described in this section to add, remove, and inspect the trailers appended to script data in generic storage descriptor records.

`OSAGetStorageType` (page 2831)
Retrieves the scripting component subtype from the script trailer appended to the script data in a generic storage descriptor record.

`OSAAddStorageType` (page 2796)
Adds a trailer to the script data in a generic storage descriptor record.

`OSARemoveStorageType` (page 2838)
Removes a trailer from the script data in a generic storage descriptor record

Miscellaneous

`ASGetAppTerminology` (page 2786)
Deprecated. Use `OSAGetAppTerminology` (page 2820) instead.

`ASGetHandler` (page 2786)
Deprecated. Use `OSAGetHandler` (page 2823) instead.

`ASGetProperty` (page 2787)
Deprecated. Use `OSAGetProperty` (page 2825) instead.

`ASSetHandler` (page 2790)
Deprecated. Use `OSASetHandler` (page 2843) instead.

`ASSetProperty` (page 2790)
Deprecated. Use `OSASetProperty` (page 2844) instead.

`OSAGetHandler` (page 2823)
Gets a script ID for the specified script handler from the specified script.

`OSAGetHandlerNames` (page 2824)
Gets a list of all handler names in the specified script as an `AEDescList` of descriptors of type `typeChar`.

`OSAGetProperty` (page 2825)
Gets the value of a specified script property from a specified script.

`OSAGetPropertyNames` (page 2825)
Gets a list of all property names from the specified script.

`OSAGetSysTerminology` (page 2832)
Gets one or more scripting terminology resources from the OSA system.

`OSASetHandler` (page 2843)
Sets a specified script handler in the specified script to the supplied handler.

`OSASetProperty` (page 2844)
Sets the value of a script property in a specified script, creating the property if it does not already exist.

`OSAGetAppTerminology` (page 2820) Deprecated in Mac OS X v10.5
Gets one or more scripting terminology resources from the specified file. (Deprecated. Use `OSACopyScriptingDefinition` (page 2803) instead.)

Creating, Invoking and Disposing Universal Procedure Pointers

`NewOSAActiveUPP` (page 2795)
Creates a new universal procedure pointer to an application-defined active function.

`NewOSACreateAppleEventUPP` (page 2795)
Creates a new universal procedure pointer to an application-defined Apple event creation function.

`NewOSASendUPP` (page 2795)
Creates a new universal procedure pointer to an application-defined send function.

`DisposeOSAActiveUPP` (page 2792)
Disposes of a universal procedure pointer to an application-defined active function.

`DisposeOSACreateAppleEventUPP` (page 2793)
Disposes of a universal procedure pointer to an application-defined Apple event create function.

`DisposeOSASendUPP` (page 2793)
Disposes of a universal procedure pointer to an application-defined send function.

`InvokeOSAActiveUPP` (page 2793)
Invokes an application-defined active function.

`InvokeOSACreateAppleEventUPP` (page 2794)
Invokes an application-defined Apple event creation function.

`InvokeOSASendUPP` (page 2794)
Invokes an application-defined send function.

Deprecated Functions

Warning: Do not use the OSA debugging functions listed here. They were were not intended for public use, they do not work, and they will return an error.

`OSADebuggerCreateSession` (page 2805)
Do not use.

`OSADebuggerDisposeCallFrame` (page 2805)
Do not use.

`OSADebuggerDisposeSession` (page 2805)
Do not use.

`OSADebuggerGetBreakpoint` (page 2806)
Do not use.

`OSADebuggerGetCallFrameState` (page 2806)
Do not use.

`OSADebuggerGetCurrentCallFrame` (page 2807)
Do not use.

`OSADebuggerGetDefaultBreakpoint` (page 2807)
Do not use.

`OSADebuggerGetPreviousCallFrame` (page 2807)
Do not use.

`OSADebuggerGetSessionState` (page 2808)
Do not use.

`OSADebuggerGetStatementRanges` (page 2808)
Do not use.

`OSADebuggerGetVariable` (page 2809)
Do not use.

`OSADebuggerSessionStep` (page 2809)
Do not use.

`OSADebuggerSetBreakpoint` (page 2809)
Do not use.

`OSADebuggerSetVariable` (page 2810)
Do not use.

`ASGetSourceStyles` (page 2788) Deprecated in Mac OS X v10.5
Gets the script format styles currently used by the AppleScript component to display scripts. (Deprecated. Use `ASGetSourceStyleNames` (page 2787) instead.)

`ASSetSourceStyles` (page 2792) Deprecated in Mac OS X v10.5
Sets the script format styles used by the AppleScript component to display scripts. (Deprecated. Use `ASSetSourceAttributes` (page 2791) instead.)

Functions

ASCopySourceAttributes

Gets the current text style attributes AppleScript uses to display script text.

```
OSAError ASCopySourceAttributes (
    ComponentInstance scriptingComponent,
    CFArrayRef *resultingSourceAttributes
);
```

Parameters

scriptingComponent
A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

resultingSourceAttributes
If successful, returns a reference to an array (of type `CFArray`) of dictionaries (of type `CFDictionary`) of text style attributes; otherwise, returns `nil`.

The order of elements in the array corresponds to the constants defined in "Source Style Constants" (page 2882), and therefore also to the names returned by `ASGetSourceStyleNames` (page 2787). For example, the first dictionary in the array (at position `kASSourceStyleUncompiledText`) describes the style for uncompiled text. However, you should not rely on there being any specific number of dictionaries in the returned array—instead, count the number of items in the array before accessing any of them.

This array is a copy and the caller is responsible for releasing it, according to the rules described in Ownership Policy in *Memory Management Programming Guide for Core Foundation*.

Return Value

A result code. See "Result Codes" (page 2885).

Discussion

A text style attribute is typically something that is meaningful to a `CFAttributedString`, such as the one returned by `OSACopyDisplayString` (page 2801) or `OSACopySourceString` (page 2804). However, clients may add other attributes using `ASSetSourceAttributes` (page 2791).

Availability
Available in Mac OS X v10.5 and later.

Declared In
`AppleScript.h`

ASGetAppTerminology

Deprecated. Use `OSAGetAppTerminology` (page 2820) instead.

```
OSAError ASGetAppTerminology (
   ComponentInstance scriptingComponent,
   FSSpec *fileSpec,
   short terminologID,
   Boolean *didLaunch,
   AEDesc *terminologyList
);
```

Return Value
A result code. See "Result Codes" (page 2885).

Version Notes
Provided for backward compatibility only. Use `OSAGetAppTerminology` (page 2820) instead.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ASDebugging.h`

ASGetHandler

Deprecated. Use `OSAGetHandler` (page 2823) instead.

```
OSAError ASGetHandler (
   ComponentInstance scriptingComponent,
   OSAID contextID,
   const AEDesc *handlerName,
   OSAID *resultingCompiledScriptID
);
```

Return Value
A result code. See "Result Codes" (page 2885).

Version Notes
Provided for backward compatibility only. Use `OSAGetHandler` (page 2823) instead.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ASDebugging.h`

ASGetProperty

Deprecated. Use `OSAGetProperty` (page 2825) instead.

```
OSAError ASGetProperty (
   ComponentInstance scriptingComponent,
   OSAID contextID,
   const AEDesc *variableName,
   OSAID *resultingScriptValueID
);
```

Return Value
A result code. See "Result Codes" (page 2885).

Version Notes
Provided for backward compatibility only. Use `OSAGetProperty` (page 2825) instead.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ASDebugging.h`

ASGetSourceStyleNames

Obtains a list of style names that are each formatted according to the script format styles currently used by the AppleScript component.

```
OSAError ASGetSourceStyleNames (
   ComponentInstance scriptingComponent,
   SInt32 modeFlags,
   AEDescList *resultingSourceStyleNamesList
);
```

Parameters

scriptingComponent
A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

modeFlags
Reserved for future use. Set to `kOSAModeNull`.

resultingSourceStyleNames
A pointer to a list of style names (for example, "Uncompiled Text," "Normal Text") that are each formatted according to the current script format styles. The order of the names corresponds to the order of the source style constants listed in "Source Style Constants" (page 2882). For example, the first name in the list (at position `kASSourceStyleUncompiledText`) is formatted according to the style for uncompiled text.

Return Value
A result code. See "Result Codes" (page 2885).

Availability
Available in Mac OS X v10.0 and later.

Declared In
AppleScript.h

ASGetSourceStyles

Gets the script format styles currently used by the AppleScript component to display scripts. (Deprecated in Mac OS X v10.5. Use ASGetSourceStyleNames (page 2787) instead.)

```
OSAError ASGetSourceStyles (
    ComponentInstance scriptingComponent,
    STHandle *resultingSourceStyles
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function OpenDefaultComponent or OpenComponent.

resultingSourceStyles

A pointer to a handle to a style element array defined by the TextEdit data type TEStyleTable that defines the styles used for different kinds of AppleScript terms.

Return Value
A result code. See "Result Codes" (page 2885).

Discussion
The ASGetSourceStyles function returns a style element array that defines the styles used for AppleScript terms. You can use the index constants described in "Source Style Constants" (page 2882) to identify individual styles returned in the *resultingSourceStyles* parameter. Other AppleScript dialects may define additional styles. When you have finished using the style element array, you must dispose of it.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
AppleScript.h

ASInit

Initializes the AppleScript component.

```
OSAError ASInit (
   ComponentInstance scriptingComponent,
   SInt32 modeFlags,
   UInt32 minStackSize,
   UInt32 preferredStackSize,
   UInt32 maxStackSize,
   UInt32 minHeapSize,
   UInt32 preferredHeapSize,
   UInt32 maxHeapSize
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

modeFlags

Reserved for future use. Set to `kOSAModeNull`.

minStackSize

The minimum size for the portion of the application's heap used by the AppleScript component's application-specific stack.

preferredStackSize

The preferred size for the portion of the application's heap used by the AppleScript component's application-specific stack.

maxStackSize

The maximum size for the portion of the application's heap used by the AppleScript component's application-specific stack.

minHeapSize

The minimum size for the portion of the application's heap used by the AppleScript component's application-specific heap. (See Version Notes section.)

preferredHeapSize

The preferred size for the portion of the application's heap used by the AppleScript component's application-specific heap. (See Version Notes section.)

maxHeapSize

The maximum size for the portion of the application's heap used by the AppleScript component's application-specific heap. (See Version Notes section.)

Return Value

A result code. See "Result Codes" (page 2885).

Discussion

Your application should set the *modeFlags* parameter to `kOSAModeNull`. You can use the other parameters to specify memory sizes for the portion of your application's heap used by the AppleScript component for its application-specific heap and stack. If your application sets any of these parameters to 0, the AppleScript component uses the corresponding value in your application's `'scsz'` resource. If that value is also set to 0, the AppleScript component uses the default values described in "Default Initialization Values" (page 2863).

If your application doesn't call `ASInit` explicitly, the AppleScript component initializes itself using the values specified in your application's `'scsz'` resource when your application first calls any scripting component routine. If any of these values are set to 0, the AppleScript component uses the corresponding default value.

If your application doesn't call `ASInit` explicitly and doesn't call any scripting component routines, the AppleScript component will not be initialized. For example, if your application opens and closes the AppleScript component or calls Component Manager functions such as `OpenDefaultComponent` or `FindNextComponent` but doesn't call any scripting component routines, the AppleScript component is not initialized.

When the AppleScript component is initialized, it uses your application's high memory to create the blocks that it locks for its own use. If you expect to lock any portion of high memory for a shorter time than you expect the AppleScript component to be available, you should call `ASInit` explicitly.

Version Notes
Starting in Mac OS X version 10.5, heap size parameter values are ignored—AppleScript's heap will grow as large as needed.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AppleScript.h`

ASSetHandler

Deprecated. Use `OSASetHandler` (page 2843) instead.

```
OSAError ASSetHandler (
   ComponentInstance scriptingComponent,
   OSAID contextID,
   const AEDesc *handlerName,
   OSAID compiledScriptID
);
```

Return Value
A result code. See "Result Codes" (page 2885).

Version Notes
Provided for backward compatibility only. Use `OSASetHandler` (page 2843) instead.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ASDebugging.h`

ASSetProperty

Deprecated. Use `OSASetProperty` (page 2844) instead.

```
OSAError ASSetProperty (
   ComponentInstance scriptingComponent,
   OSAID contextID,
   const AEDesc *variableName,
   OSAID scriptValueID
);
```

Return Value
A result code. See "Result Codes" (page 2885).

Version Notes
Provided for backward compatibility only. Use `OSASetProperty` (page 2844) instead.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`ASDebugging.h`

ASSetSourceAttributes

Sets the text style attributes used by the AppleScript component to display scripts.

```
OSAError ASSetSourceAttributes (
    ComponentInstance scriptingComponent,
    CFArrayRef sourceAttributes
);
```

Parameters

`scriptingComponent`

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

`sourceAttributes`

A reference to an array (of type `CFArray`) of dictionaries (of type `CFDictionary`) of text style attributes.

You can pass a `nil` reference for this parameter if you want the AppleScript component to display script text using its default styles.

Return Value
A result code. See "Result Codes" (page 2885).

Discussion
A text style attribute is typically something that is meaningful to a `CFAttributedString`, such as the one returned by `OSACopyDisplayString` (page 2801) or `OSACopySourceString` (page 2804). However, clients may add any attributes they like. Because of this, you should generally call `ASSetSourceAttributes` with a modified copy of the result from `ASCopySourceAttributes` (page 2785), not a built-from-scratch set of attributes.

The order of elements in the array should correspond to the constants defined in "Source Style Constants" (page 2882), and therefore also to the names returned by `ASGetSourceStyleNames` (page 2787). After calling `ASSetSourceAttributes`, you must dispose of the style element array you used to specify the text style attributes.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`AppleScript.h`

ASSetSourceStyles

Sets the script format styles used by the AppleScript component to display scripts. (Deprecated in Mac OS X v10.5. Use `ASSetSourceAttributes` (page 2791) instead.)

```
OSAError ASSetSourceStyles (
    ComponentInstance scriptingComponent,
    STHandle sourceStyles
);
```

Parameters

`scriptingComponent`

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

`sourceStyles`

A handle to a style element array defined by the TextEdit data type `TEStyleTable` that defines the styles used for different kinds of AppleScript terms. The style for each kind of term should be identified according to the index constants listed in "Source Style Constants" (page 2882).

Return Value
A result code. See "Result Codes" (page 2885).

Discussion
The `ASSetSourceStyles` function sets the script format styles used to display scripts. If you pass a `NULL` handle in the *`sourceStyles`* parameter, the AppleScript component uses its default styles.

After you have set the script format styles, you must dispose of the style element array you used to specify them.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.5.

Declared In
`AppleScript.h`

DisposeOSAActiveUPP

Disposes of a universal procedure pointer to an application-defined active function.

```
void DisposeOSAActiveUPP (
   OSAActiveUPP userUPP
);
```

Parameters

`userUPP`

The UPP to dispose of.

Availability
Available in Mac OS X v10.0 and later.

Declared In
OSA.h

DisposeOSACreateAppleEventUPP

Disposes of a universal procedure pointer to an application-defined Apple event create function.

```
void DisposeOSACreateAppleEventUPP (
   OSACreateAppleEventUPP userUPP
);
```

Parameters

userUPP
The UPP to dispose of.

Availability
Available in Mac OS X v10.0 and later.

Declared In
OSA.h

DisposeOSASendUPP

Disposes of a universal procedure pointer to an application-defined send function.

```
void DisposeOSASendUPP (
   OSASendUPP userUPP
);
```

Parameters

userUPP
The UPP to dispose of.

Availability
Available in Mac OS X v10.0 and later.

Declared In
OSA.h

InvokeOSAActiveUPP

Invokes an application-defined active function.

```
OSErr InvokeOSAActiveUPP (
   SRefCon refCon,
   OSAActiveUPP userUPP
);
```

Return Value
A result code. See "Result Codes" (page 2885).

Availability
Available in Mac OS X v10.0 and later.

Declared In
OSA.h

InvokeOSACreateAppleEventUPP

Invokes an application-defined Apple event creation function.

```
OSErr InvokeOSACreateAppleEventUPP (
   AEEventClass theAEEventClass,
   AEEventID theAEEventID,
   const AEAddressDesc *target,
   short returnID,
   SInt32 transactionID,
   AppleEvent *result,
   SRefCon refCon,
   OSACreateAppleEventUPP userUPP
);
```

Return Value
A result code. See "Result Codes" (page 2885).

Availability
Available in Mac OS X v10.0 and later.

Declared In
OSA.h

InvokeOSASendUPP

Invokes an application-defined send function.

```
OSErr InvokeOSASendUPP (
   const AppleEvent *theAppleEvent,
   AppleEvent *reply,
   AESendMode sendMode,
   AESendPriority sendPriority,
   SInt32 timeOutInTicks,
   AEIdleUPP idleProc,
   AEFilterUPP filterProc,
   SRefCon refCon,
   OSASendUPP userUPP
);
```

Return Value
A result code. See "Result Codes" (page 2885).

Availability
Available in Mac OS X v10.0 and later.

Declared In
OSA.h

NewOSAActiveUPP

Creates a new universal procedure pointer to an application-defined active function.

```
OSAActiveUPP NewOSAActiveUPP (
   OSAActiveProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to the active function.

Return Value

The new UPP.

Availability

Available in Mac OS X v10.0 and later.

Declared In

OSA.h

NewOSACreateAppleEventUPP

Creates a new universal procedure pointer to an application-defined Apple event creation function.

```
OSACreateAppleEventUPP NewOSACreateAppleEventUPP (
   OSACreateAppleEventProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to the creation function.

Return Value

The new UPP.

Availability

Available in Mac OS X v10.0 and later.

Declared In

OSA.h

NewOSASendUPP

Creates a new universal procedure pointer to an application-defined send function.

```
OSASendUPP NewOSASendUPP (
   OSASendProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to the send function.

Return Value

The new UPP.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OSA.h`

OSAAddStorageType

Adds a trailer to the script data in a generic storage descriptor record.

```
OSErr OSAAddStorageType (
   AEDataStorage scriptData,
   DescType dscType
);
```

Parameters

`scriptData`
A handle to the script data.

`dscType`
The descriptor type to be specified in the trailer added to the script data.

Return Value
A result code. See "Result Codes" (page 2885).

Discussion
The `OSAAddStorageType` function attaches a trailer to a handle (consequently expanding the data to which the handle refers) or updates an existing trailer.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OSAComp.h`

OSAAvailableDialectCodeList

Obtains a descriptor list containing dialect codes for each of a scripting component's currently available dialects.

```
OSAError OSAAvailableDialectCodeList (
   ComponentInstance scriptingComponent,
   AEDesc *resultingDialectCodeList
);
```

Parameters

`scriptingComponent`
A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

`resultingDialectCodeList`
A pointer to the returned descriptor list.

Return Value
A result code. See "Result Codes" (page 2885).

Discussion
Each item in the descriptor list returned by `OSAAvailableDialectCodeList` is a descriptor record of descriptor type `typeInteger` containing a dialect code for one of the specified scripting component's currently available dialects. Dialect codes are defined by individual scripting components.

You can pass any dialect code you obtain using `OSAAvailableDialectCodeList` to `OSAGetDialectInfo` to get information about the corresponding dialect.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OSA.h`

OSAAvailableDialects

Obtains a descriptor list containing information about each of the currently available dialects for a scripting component.

```
OSAError OSAAvailableDialects (
   ComponentInstance scriptingComponent,
   AEDesc *resultingDialectInfoList
);
```

Parameters

`scriptingComponent`
A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

`resultingDialectInfoList`
A pointer to the returned descriptor list.

Return Value
A result code. See "Result Codes" (page 2885).

Discussion
Each item in the list returned by `OSAAvailableDialects` is an AE record of descriptor type `typeOSADialectInfo`. Each descriptor record in the descriptor list contains, at a minimum, four keyword-specified descriptor records with the keywords described in "Dialect Descriptor Constants" (page 2865).

Rather than calling `OSAAvailableDialects` to obtain complete dialect information for a scripting component, it is usually more convenient to call `OSAAvailableDialectCodeList` to get a list of codes for a scripting component's dialects, then call `OSAGetDialectInfo` to get information about the specific dialect you're interested in.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OSA.h`

OSACoerceFromDesc

Obtains the script ID for a script value that corresponds to the data in a descriptor record.

```
OSAError OSACoerceFromDesc (
   ComponentInstance scriptingComponent,
   const AEDesc *scriptData,
   SInt32 modeFlags,
   OSAID *resultingScriptID
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

scriptData

A pointer to a descriptor record containing the script data to be coerced.

modeFlags

Information used by individual scripting components. To avoid setting mode flag values, specify `kOSAModeNull`. If the *scriptData* parameter contains an Apple event, you can use any of the mode flags listed in "Mode Flags" (page 2873).

resultingScriptValueID

A pointer to the resulting script ID for a script value. See the `OSAID` (page 2855) data type.

Return Value

A result code. See "Result Codes" (page 2885).

Discussion

The `OSACoerceFromDesc` function coerces the descriptor record in the *scriptData* parameter to the equivalent script value and returns a script ID for that value.

If you pass `OSACoerceFromDesc` an Apple event in the *scriptData* parameter, it returns a script ID for the equivalent compiled script in the *resultingScriptValueID* parameter. In this case you can specify any of the *modeFlags* values used by `OSACompile` to control the way the compiled script is executed.

If you call `OSACoerceFromDesc` using an instance of the generic scripting component, the generic scripting component uses the default scripting component to perform the coercion.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OSA.h`

OSACoerceToDesc

Coerces a script value to a descriptor record of a desired descriptor type.

```
OSAError OSACoerceToDesc (
   ComponentInstance scriptingComponent,
   OSAID scriptID,
   DescType desiredType,
   SInt32 modeFlags,
   AEDesc *result
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

scriptID

The script ID for the script value to coerce. See the `OSAID` (page 2855) data type.

desiredType

The desired descriptor type of the resulting descriptor record.

modeFlags

Information used by individual scripting components. To avoid setting mode flag values, specify `kOSAModeNull`.

result

A pointer to the resulting descriptor record.

Return Value

A result code. See "Result Codes" (page 2885).

Discussion

The `OSACoerceToDesc` function coerces the script value identified by *scriptValueID* to a descriptor record of the type specified by the *desiredType* parameter, if possible. Valid types include all the standard descriptor types, plus any special types supported by the scripting component.

If you want the descriptor type of the descriptor record returned in the result parameter to be the same as the descriptor type returned by a scripting component, use `OSACoerceToDesc` and specify `typeWildCard` as the desired type. If you want to get a script value in a form that you can display for humans to read, use `OSADisplay`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OSA.h`

OSACompile

Compiles the source data for a script and obtain a script ID for a compiled script or a script context.

```
OSAError OSACompile (
   ComponentInstance scriptingComponent,
   const AEDesc *sourceData,
   SInt32 modeFlags,
   OSAID *previousAndResultingScriptID
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

sourceData

A pointer to a descriptor record containing suitable source data for the specified scripting component.

modeFlags

Information used by individual scripting components. To avoid setting mode flag values, specify `kOSAModeNull`. Other possible mode flags are listed in "Mode Flags" (page 2873).

previousAndResultingScriptID

A pointer to the script ID for the resulting compiled script. If the value of this parameter on input is `kOSANullScript`, `OSACompile` returns a new script ID for the compiled script data. If the value of this parameter on input is an existing script ID, `OSACompile` updates the script ID so that it refers to the newly compiled script data. See the `OSAID` (page 2855) data type.

Return Value

A result code. See "Result Codes" (page 2885).

Discussion

You can pass a descriptor record containing source data suitable for a specific scripting component (usually text) to the `OSACompile` function to obtain a script ID for the equivalent compiled script or script context. To compile the source data as a script context for use with `OSAExecuteEvent` or `OSADoEvent`, you must set the `kOSAModeCompileIntoContext` flag, and the source data should include appropriate handlers.

After you have successfully compiled the script, you can use the returned script ID to refer to the compiled script when you call `OSAExecute` and other scripting component routines.

If you use `OSACompile` with an instance of the generic scripting component and pass `kOSANullScript` in the *previousAndResultingScriptID* parameter, the generic scripting component uses the default scripting component to compile the script.

If you're recompiling a script, specify the original script ID in the *previousAndResultingScriptID* parameter. The generic scripting component uses the script ID to determine which scripting component it should use to compile the script.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OSA.h`

OSACompileExecute

Compiles and executes a script in a single step rather than calling `OSACompile` and `OSAExecute`.

```
OSAError OSACompileExecute (
   ComponentInstance scriptingComponent,
   const AEDesc *sourceData,
   OSAID contextID,
   SInt32 modeFlags,
   OSAID *resultingScriptValueID
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

sourceData

A pointer to a descriptor record identifying suitable source data for the specified scripting component.

contextID

The script ID for the context to be used during script execution. The constant `kOSANullScript` in this parameter indicates that the scripting component should use its default context. See the `OSAID` (page 2855) data type.

modeFlags

Information used by individual scripting components. To avoid setting mode flag values, specify `kOSAModeNull`. Other possible mode flags are listed in "Mode Flags" (page 2873).

resultingScriptValueID

A pointer to the script ID for the script value returned.

Return Value

A result code. See "Result Codes" (page 2885).

Discussion

The `OSACompileExecute` function compiles source data and executes the resulting compiled script, using the script context identified by the *contextID* parameter to maintain state information such as the binding of variables. After successfully executing the script, `OSACompileExecute` disposes of the compiled script and returns either the script ID for the resulting script value or, if execution does not result in a value, the constant `kOSANullScript`.

If the result c ode returned by `OSACompileExecute` is a general result code, there was some problem in arranging for the script to be run. If the result code is `errOSAScriptError`, an error occurred during script execution. In this case, you can obtain more detailed error information by calling `OSAScriptError`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OSA.h`

OSACopyDisplayString

Converts a script value to an attributed Unicode text string, which your application can display to the user.

```
OSAError OSACopyDisplayString (
    ComponentInstance scriptingComponent,
    OSAID scriptID,
    SInt32 modeFlags,
    CFAttributedStringRef *result
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

scriptID

The script ID for the script value to display. See the `OSAID` (page 2855) data type.

modeFlags

Information used by individual scripting components. To avoid setting any mode flags, specify `kOSAModeNull`. To make the resulting text readable by humans only, so that it can't be recompiled, specify `kOSAModeDisplayForHumans`.

result

If successful, a reference to the script data as an attributed Unicode text string; otherwise not defined.

Because the `result` parameter returns a copy, the caller is responsible for releasing this string object, according to the rules described in Ownership Policy in *Memory Management Programming Guide for Core Foundation*.

Return Value
A result code. See "Result Codes" (page 2885).

Discussion
The `OSACopyDisplayString` function is analogous to `OSADisplay` (page 2810), except that it returns the script text as an attributed Unicode text string. An instance of `CFAttributedString` manages a character string and an associated set of attributes that apply to characters or ranges of characters in the string. You can call `ASCopySourceAttributes` (page 2785) to get the current AppleScript source style attributes.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`OSA.h`

OSACopyID

Updates script data after editing or recording and to perform undo or revert operations on script data.

```
OSAError OSACopyID (
   ComponentInstance scriptingComponent,
   OSAID fromID,
   OSAID *toID
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

fromID

The script ID for script data that you want to be associated with the script ID in the *toID* parameter. See the OSAID (page 2855) data type.

toID

A pointer to the script ID for the script data to be replaced. If the value of this parameter is kOSANullScript, the OSACopyID function returns a new script ID. See the OSAID (page 2855) data type.

Return Value
A result code. See "Result Codes" (page 2885).

Discussion
The OSACopyID function replaces the script data identified by the script ID in the *toID* parameter with the script data identified by the script ID in the *fromID* parameter.

Availability
Available in Mac OS X v10.0 and later.

Declared In
OSA.h

OSACopyScriptingDefinition

Creates a copy of a scripting definition (sdef) from the specified file or bundle.

```
OSAError OSACopyScriptingDefinition (
   const FSRef *ref,
   SInt32 modeFlags,
   CFDataRef *sdef
);
```

Parameters

ref

A file reference to the application file or bundle from which to copy the scripting definition.

modeFlags

Reserved for future use. Set to kOSAModeNull.

sdef

On return, the resulting scripting definition, as XML data.

Return Value
A result code. See "Result Codes" (page 2885).

Discussion
If the target application does not have a true scripting definition (sdef) but does have an 'aete' resource or a Cocoa script suite, this function translates the existing information to an sdef. As a result, OSACopyScriptingDefinition works for any scriptable application.

To provide a scripting definition in your application:

1. Put the sdef file in the Resources folder of the application bundle.
2. Add an entry to your information property list (Info.plist) file:
   - key: "OSAScriptingDefinition"

- value: “MyApplication.sdef” (the name of the sdef)

For an introduction to scripting definitions, see “Specifying Scripting Terminology” in AppleScript Overview. See the man page for sdef(5) for details of the sdef format.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`ASDebugging.h`

OSACopySourceString

Decompiles the script data for the specified script and returns a copy of the equivalent source data as an attributed Unicode text string.

```
OSAError OSACopySourceString (
   ComponentInstance scriptingComponent,
   OSAID scriptID,
   SInt32 modeFlags,
   CFAttributedStringRef *result
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

scriptID

The script ID for the script data to decompile. If you pass `kOSANullScript` in this parameter, `OSACopySourceString` returns a null source description (such as an empty text string). See the `OSAID` (page 2855) data type.

modeFlags

No mode information is currently supported, so you should specify `kOSAModeNull` for this parameter.

result

If successful, a reference to the script data as an attributed Unicode text string; otherwise not defined.

Because the `result` parameter returns a copy, the caller is responsible for releasing this string object, according to the rules described in Ownership Policy in *Memory Management Programming Guide for Core Foundation*.

Return Value
A result code. See “Result Codes” (page 2885).

Discussion
The `OSACopySourceString` function is analogous to `OSAGetSource` (page 2830), except that it returns the decompiled script data as an attributed Unicode text string (a Core Foundation attributed string object). This data can be displayed to the user or compiled and executed. You can call `ASCopySourceAttributes` (page 2785) to get the current AppleScript source style attributes.

Availability
Available in Mac OS X v10.5 and later.

Declared In
OSA.h

OSADebuggerCreateSession

Do not use.

Unsupported

```
OSAError OSADebuggerCreateSession (
   ComponentInstance scriptingComponent,
   OSAID inScript,
   OSAID inContext,
   OSADebugSessionRef *outSession
);
```

Return Value
A result code. See "Result Codes" (page 2885).

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In
OSA.h

OSADebuggerDisposeCallFrame

Do not use.

Unsupported

```
OSAError OSADebuggerDisposeCallFrame (
   ComponentInstance scriptingComponent,
   OSADebugCallFrameRef inCallFrame
);
```

Return Value
A result code. See "Result Codes" (page 2885).

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In
OSA.h

OSADebuggerDisposeSession

Do not use.

Unsupported

```
OSAError OSADebuggerDisposeSession (
   ComponentInstance scriptingComponent,
   OSADebugSessionRef inSession
);
```

Return Value
A result code. See "Result Codes" (page 2885).

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In
`OSA.h`

OSADebuggerGetBreakpoint

Do not use.

Unsupported

```
OSAError OSADebuggerGetBreakpoint (
   ComponentInstance scriptingComponent,
   OSADebugSessionRef inSession,
   UInt32 inSrcOffset,
   OSAID *outBreakpoint
);
```

Return Value
A result code. See "Result Codes" (page 2885).

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In
`OSA.h`

OSADebuggerGetCallFrameState

Do not use.

Unsupported

```
OSAError OSADebuggerGetCallFrameState (
   ComponentInstance scriptingComponent,
   OSADebugCallFrameRef inCallFrame,
   AERecord *outState
);
```

Return Value
A result code. See "Result Codes" (page 2885).

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In
OSA.h

OSADebuggerGetCurrentCallFrame

Do not use.

Unsupported

```
OSAError OSADebuggerGetCurrentCallFrame (
   ComponentInstance scriptingComponent,
   OSADebugSessionRef inSession,
   OSADebugCallFrameRef *outCallFrame
);
```

Return Value
A result code. See "Result Codes" (page 2885).

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In
OSA.h

OSADebuggerGetDefaultBreakpoint

Do not use.

Unsupported

```
OSAError OSADebuggerGetDefaultBreakpoint (
   ComponentInstance scriptingComponent,
   OSADebugSessionRef inSession,
   OSAID *outBreakpoint
);
```

Return Value
A result code. See "Result Codes" (page 2885).

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In
OSA.h

OSADebuggerGetPreviousCallFrame

Do not use.

Unsupported

```
OSAError OSADebuggerGetPreviousCallFrame (
   ComponentInstance scriptingComponent,
   OSADebugCallFrameRef inCurrentFrame,
   OSADebugCallFrameRef *outPrevFrame
);
```

Return Value
A result code. See "Result Codes" (page 2885).

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In
`OSA.h`

OSADebuggerGetSessionState

Do not use.

Unsupported

```
OSAError OSADebuggerGetSessionState (
   ComponentInstance scriptingComponent,
   OSADebugSessionRef inSession,
   AERecord *outState
);
```

Return Value
A result code. See "Result Codes" (page 2885).

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In
`OSA.h`

OSADebuggerGetStatementRanges

Do not use.

Unsupported

```
OSAError OSADebuggerGetStatementRanges (
   ComponentInstance scriptingComponent,
   OSADebugSessionRef inSession,
   AEDescList *outStatementRangeArray
);
```

Return Value
A result code. See "Result Codes" (page 2885).

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In
OSA.h

OSADebuggerGetVariable

Do not use.

Unsupported

```
OSAError OSADebuggerGetVariable (
   ComponentInstance scriptingComponent,
   OSADebugCallFrameRef inCallFrame,
   const AEDesc *inVariableName,
   OSAID *outVariable
);
```

Return Value
A result code. See "Result Codes" (page 2885).

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In
OSA.h

OSADebuggerSessionStep

Do not use.

Unsupported

```
OSAError OSADebuggerSessionStep (
   ComponentInstance scriptingComponent,
   OSADebugSessionRef inSession,
   OSADebugStepKind inKind
);
```

Return Value
A result code. See "Result Codes" (page 2885).

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In
OSA.h

OSADebuggerSetBreakpoint

Do not use.

Unsupported

```
OSAError OSADebuggerSetBreakpoint (
   ComponentInstance scriptingComponent,
   OSADebugSessionRef inSession,
   UInt32 inSrcOffset,
   OSAID inBreakpoint
);
```

Return Value
A result code. See "Result Codes" (page 2885).

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In
OSA.h

OSADebuggerSetVariable

Do not use.

Unsupported

```
OSAError OSADebuggerSetVariable (
   ComponentInstance scriptingComponent,
   OSADebugCallFrameRef inCallFrame,
   const AEDesc *inVariableName,
   OSAID inVariable
);
```

Return Value
A result code. See "Result Codes" (page 2885).

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In
OSA.h

OSADisplay

Converts a script value to text. Your application can then use its own functions to display this text to the user.

```
OSAError OSADisplay (
   ComponentInstance scriptingComponent,
   OSAID scriptValueID,
   DescType desiredType,
   SInt32 modeFlags,
   AEDesc *resultingText
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

scriptValueID

The script ID for the script value to coerce. See the `OSAID` (page 2855) data type.

desiredType

The desired text descriptor type, such as `typeChar`, for the resulting descriptor record.

modeFlags

Information used by individual scripting components. To avoid setting any mode flags, specify `kOSAModeNull`. To make the resulting text readable by humans only, so that it can't be recompiled, specify `kOSAModeDisplayForHumans`.

resultingText

A pointer to the resulting descriptor record.

Return Value

A result code. See "Result Codes" (page 2885).

Discussion

The `OSADisplay` function coerces the script value identified by *scriptValueID* to a descriptor record of the text type specified by the *desiredType* parameter, if possible. Valid types include the standard text descriptor types, plus any special types supported by the scripting component.

Unlike `OSAGetSource` (page 2830), `OSADisplay` can coerce only script values and always produces a descriptor record of a text descriptor type. In addition, if you specify the mode flag `kOSAModeDisplayForHumans`, the resulting text cannot be recompiled.

If you want to get a script value in a form that you can display for humans to read, use `OSADisplay`. If you want the descriptor type of the descriptor record returned in the *resultingText* parameter to be the same as the descriptor type returned by a scripting component, use `OSACoerceToDesc` and specify `typeWildCard` as the desired type.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OSA.h`

OSADispose

Reclaims the memory occupied by script data.

```
OSAError OSADispose (
   ComponentInstance scriptingComponent,
   OSAID scriptID
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

scriptID

The script ID for the script data to be disposed of. See the `OSAID` (page 2855) data type.

Return Value
A result code. See "Result Codes" (page 2885).

Discussion
The `OSADispose` function releases the memory assigned to the script data identified by the *scriptID* parameter. The script ID passed to the `OSADispose` function is no longer valid if the function returns successfully. A scripting component can then reuse that script ID for other script data.

A call to `OSADispose` returns `noErr` if the script ID is `kOSANullScript`, although it does not dispose of anything.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OSA.h`

OSADoEvent

Handles an Apple event with the aid of a script context and obtains a reply event.

```
OSAError OSADoEvent (
   ComponentInstance scriptingComponent,
   const AppleEvent *theAppleEvent,
   OSAID contextID,
   SInt32 modeFlags,
   AppleEvent *reply
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

theAppleEvent

A pointer to the Apple event to be handled.

contextID

The script ID for the script context to be used to handle the Apple event. See the `OSAID` (page 2855) data type.

modeFlags

Information used by individual scripting components. To avoid setting mode flag values, specify `kOSAModeNull`. Other possible mode flags are listed in "Mode Flags" (page 2873).

reply
A pointer to the reply Apple event.

Return Value
A result code. See "Result Codes" (page 2885).

Discussion
The `OSADoEvent` function resembles both `OSADoScript` and `OSAExecuteEvent`. However, unlike `OSADoScript`, the script `OSADoEvent` executes must be in the form of a script context, and execution is initiated by an Apple event. Unlike `OSAExecuteEvent`, `OSADoEvent` returns a reply Apple event rather than the script ID of the resulting script value.

The `OSADoEvent` function, like `OSAExecuteEvent`, attempts to use the script context specified by the *`contextID`* parameter to handle the Apple event specified by the *`theAppleEvent`* parameter. If the scripting component determines that the script context can't handle the event (for example, if a script written in an AppleScript dialect doesn't include statements that handle the event), `OSADoEvent` immediately returns `errAEEventNotHandled` rather than `errOSAScriptError`. This causes the Apple Event Manager to look for an appropriate handler in the application's Apple event dispatch table or elsewhere, using standard Apple event dispatching.

If the scripting component determines that the script context can handle the event, `OSADoEvent` executes the script context's handler for the event and returns the resulting script ID.

The `OSADoEvent` function returns a reply event that contains either the resulting script value or, if an error occurred during script execution, information about the error. If the error `errOSAScriptError` occurs during script execution, `OSADoEvent` calls `OSAScriptError` and returns the appropriate error information in the reply. The `OSADoEvent` function never returns `errOSAScriptError`.

If the script context specifies that the Apple event should be passed to the application's standard handler for that event (for example, with an AppleScript continue statement), `OSADoEvent` passes the event to the resume dispatch function currently being used by the scripting component. The resume dispatch function dispatches the event directly to the application's standard handler for that event (that is, without calling `OSADoEvent` again). If the *`contextID`* parameter is `kOSANullScript`, the `OSADoEvent` function passes the event directly to the resume dispatch function. If the call to the resume dispatch function is successful, execution of the script context proceeds from the point at which the resume dispatch function was called.

Special Considerations

Like `OSAExecuteEvent`, `OSADoEvent` can generate the result code `errAEEventNotHandled` in at least two ways. If the scripting component determines that a script context doesn't declare a handler for a particular event, `OSADoEvent` immediately returns `errAEEventNotHandled`. If a scripting component calls its resume dispatch function during script execution and the application's standard handler for the event fails to handle it, `OSADoEvent` returns `errAEEventNotHandled` in the reply Apple event.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OSA.h`

OSADoScript

Compiles and executes a script and converts the resulting script value to text in a single step rather than calling `OSACompile`, `OSAExecute`, and `OSADisplay`.

```
OSAError OSADoScript (
   ComponentInstance scriptingComponent,
   const AEDesc *sourceData,
   OSAID contextID,
   DescType desiredType,
   SInt32 modeFlags,
   AEDesc *resultingText
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

sourceData

A pointer to a descriptor record identifying suitable source data for the specified scripting component.

contextID

The script ID for the context to be used during script execution. The constant `kOSANullScript` in this parameter indicates that the scripting component should use its default context. See the `OSAID` (page 2855) data type.

desiredType

The desired text descriptor type, such as `typeChar`, for the resulting descriptor record.

modeFlags

Information used by individual scripting components. To avoid setting mode flag values, specify `kOSAModeNull`. Other possible mode flags are listed in "Mode Flags" (page 2873).

resultingText

A pointer to the resulting descriptor record.

Return Value

A result code.

If the result code returned by `OSADoScript` is a general result code, there was some problem in arranging for the script to be run. If an error occurs during script execution, the error message of the error is stored in `resultingText`, and the function returns `errOSAScriptError`. You can use `OSAScriptError` (page 2839) to obtain more information about the particular error.

For additional information on result codes, see "Result Codes" (page 2885).

Discussion

Calling the `OSADoScript` function is equivalent to calling `OSACompile` followed by `OSAExecute` and `OSADisplay`. After compiling the source data, executing the compiled script using the script context identified by the *contextID* parameter, and returning the text equivalent of the resulting script value in the *resultingText* parameter, `OSADoScript` disposes of both the compiled script and the resulting script value.

Special Considerations

Prior to Mac OS X version 10.5, if an error occurred during script execution, the error message of the error was not returned in `resultingText`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OSA.h`

OSADoScriptFile

Loads a script from the specified file, compiles the script if the file is a text file, executes the script, converts the resulting script value to text, and stores the script back into the file if the script has persistent properties and the file is not a text file.

```
OSAError OSADoScriptFile (
   ComponentInstance scriptingComponent,
   const FSRef *scriptFile,
   OSAID contextID,
   DescType desiredType,
   SInt32 modeFlags,
   AEDesc *resultingText
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`. See the Component Manager documentation for a description of the `ComponentInstance` data type.

scriptFile

Identifies the file to load the script from and to save the script back to (if the script has persistent properties and the file is not a text file). See the File Manager documentation for a description of the `FSRef` data type.

File format is determined by inspection. If the file is a text file, `OSADoScriptFile` uses the following steps to determine the text encoding:

- If a Unicode BOM is present, that determines the encoding—one of UTF-16BE, UTF-16LE, or UTF-8
- Otherwise, if the file is valid UTF-8, it is assumed to be UTF-8.
- Otherwise, it is assumed to be in the primary encoding.

contextID

The script ID for the context to be used during script execution. The constant `kOSANullScript` in this parameter indicates that the scripting component should use its default context. See the `OSAID` (page 2855) data type.

desiredType

The desired text descriptor type, such as `typeChar`, for the resulting descriptor record.

modeFlags

Information for use by the scripting component. Can include any of the mode flags that would normally be sent to the `OSACompile` (page 2799) (if the file is a text file), `OSADisplay` (page 2810), `OSAExecute` (page 2816), and `OSALoad` (page 2832) functions. For descriptions of the mode flag usage of those functions, see the chapter "Scripting Components" in "Interapplication Communication" at http://developer.apple.com/documentation/mac/IAC/IAC-2.html.

resultingText

The descriptor record for the resulting script value. The `AEDesc` data type is described in Apple Event Manager Reference.

Return Value

A result code. See "Result Codes" (page 2885).

Discussion
This routine is effectively equivalent to calling `OSALoadFile` (page 2835), followed by `OSAExecute` (page 2816), `OSADisplay` (page 2810), and then `OSAStoreFile` (page 2851) if the script has persistent properties. After execution, the compiled source and the resulting value are disposed. Only the `resultingText` descriptor is retained. If an error occurs during script execution, the error message of the error is stored in `resultingText`, and the function returns `errOSAScriptError`. You can use `OSAScriptError` (page 2839) to obtain more information about the particular error.

Special Considerations

Prior to Mac OS X version 10.5, if an error occurred during script execution, the error message of the error was not returned in `resultingText`.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`OSA.h`

OSAExecute

Executes a compiled script or a script context.

```
OSAError OSAExecute (
   ComponentInstance scriptingComponent,
   OSAID compiledScriptID,
   OSAID contextID,
   SInt32 modeFlags,
   OSAID *resultingScriptValueID
);
```

Parameters

scriptingComponent
A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

compiledScriptID
The script ID for the compiled script to be executed. See the `OSAID` (page 2855) data type.

contextID
The script ID for the context to be used during script execution. The constant `kOSANullScript` in this parameter indicates that the scripting component should use its default context. See the `OSAID` (page 2855) data type.

modeFlags
Information used by individual scripting components. To avoid setting mode flag values, specify `kOSAModeNull`. Other possible mode flags are listed in the description that follows.

resultingScriptValueID
A pointer to the script ID for the script value returned. See the `OSAID` (page 2855) data type.

Return Value
A result code. See "Result Codes" (page 2885). If the result code returned by `OSAExecute` is a general result code, there was some problem in arranging for the script to be run. If the result code is `errOSAScriptError`, an error occurred during script execution. In this case, you can obtain more detailed error information by calling `OSAScriptError`.

Discussion
The `OSAExecute` function executes the compiled script identified by the *`compiledScriptID`* parameter, using the script context identified by the *`contextID`* parameter to maintain state information, such as the binding of variables, for the compiled script. After successfully executing a script, `OSAExecute` returns the script ID for a resulting script value, or, if execution does not result in a value, the constant `kOSANullScript`. You can use the `OSACoerceToDesc` function to coerce the resulting script value to a descriptor record of a desired descriptor type, or the `OSADisplay` (page 2810) function to obtain the equivalent source data for the script value. You can control the way in which the scripting component executes a script by adding any of the flags described in "Mode Flags" (page 2873).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OSA.h`

OSAExecuteEvent

Handles an Apple event with the aid of a script context and obtains a script ID for the resulting script value.

```
OSAError OSAExecuteEvent (
   ComponentInstance scriptingComponent,
   const AppleEvent *theAppleEvent,
   OSAID contextID,
   SInt32 modeFlags,
   OSAID *resultingScriptValueID
);
```

Parameters

`scriptingComponent`
A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

`theAppleEvent`
A pointer to the Apple event to be handled.

`contextID`
The script ID for the script context to be used to handle the Apple event. See the `OSAID` (page 2855) data type.

`modeFlags`
Information used by individual scripting components. To avoid setting mode flag values, specify `kOSAModeNull`. Other possible mode flags are listed in "Mode Flags" (page 2873).

`resultingScriptValueID`
A pointer to the script ID for the resulting script value.

Return Value
A result code. See "Result Codes" (page 2885).

Discussion
The `OSAExecuteEvent` function attempts to use the script context specified by the *`contextID`* parameter to handle the Apple event specified by the *`theAppleEvent`* parameter. If the scripting component determines that the script context can't handle the event (for example, if a script written in AppleScript doesn't include

statements that handle the event), `OSAExecuteEvent` immediately returns `errAEEventNotHandled` rather than `errOSAScriptError`. This causes the Apple Event Manager to look for an appropriate handler in the application's Apple event dispatch table or elsewhere, using standard Apple event dispatching.

If the scripting component determines that the script context can handle the event, `OSAExecuteEvent` executes the script context's handler and returns the resulting script ID. If execution of the script context's handler for the event generates an error, `OSAExecuteEvent` returns `errOSAScriptError`, and you can get more detailed error information by calling the `OSAScriptError` function.

If the script context identified by the *`contextID`* parameter specifies that the Apple event should be passed to the application's default handler for that event (for example, with an AppleScript `continue` statement), `OSAExecuteEvent` passes the event to the resume dispatch function currently being used by the scripting component. The resume dispatch function dispatches the event directly to the application's standard handler for that event (that is, without calling `OSAExecuteEvent` again). If the *`contextID`* parameter is `kOSANullScript`, the `OSAExecuteEvent` function passes the event directly to the resume dispatch function. If a call to the resume dispatch function is successful, execution of the script context proceeds from the point at which the resume dispatch function was called.

Special Considerations

The `OSAExecuteEvent` function can generate the result code `errAEEventNotHandled` in at least two ways. If the scripting component determines that a script context doesn't declare a handler for a particular event, `OSAExecuteEvent` immediately returns `errAEEventNotHandled`. If a scripting component calls its resume dispatch function during script execution and the application's standard handler for the event fails to handle it, `OSAExecuteEvent` returns `errOSAScriptError` and a call to `OSAScriptError` with `kOSAErrorNumber` in the *`selector`* parameter returns `errAEEventNotHandled` as the resulting error description.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OSA.h`

OSAGenericToRealID

Converts a generic script ID to the corresponding component-specific script ID.

```
OSAError OSAGenericToRealID (
   ComponentInstance genericScriptingComponent,
   OSAID *theScriptID,
   ComponentInstance *theExactComponent
);
```

Parameters

`genericScriptingComponent`

A component instance for the generic scripting component, created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

`theScriptID`

A pointer to the generic script ID that you want to convert. The `OSAGenericToRealID` function returns, in this parameter, the component-specific script ID that corresponds to the generic script ID that you pass in this parameter. See the `OSAID` (page 2855) data type.

theExactComponent
On return, a pointer to the component instance that created the script ID returned in the *theScriptID* parameter.

Return Value
A result code. See "Result Codes" (page 2885).

Discussion
You can't use the generic scripting component and a generic script ID with component-specific routines. Instead, you can use the component instance and script ID returned by `OSAGenericToRealID`.

Given a generic script ID (that is, a script ID returned by a call to a standard component routine via the generic scripting component), the `OSAGenericToRealID` function returns the equivalent component-specific script ID and the component instance that created that script ID. The `OSAGenericToRealID` function modifies the script ID in place, changing the generic script ID you pass in the *theScriptID* parameter to the corresponding component-specific script ID.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OSAGeneric.h`

OSAGetActiveProc

Gets a pointer to the active function that a scripting component is currently using.

```
OSAError OSAGetActiveProc (
   ComponentInstance scriptingComponent,
   OSAActiveUPP *activeProc,
   SRefCon *refCon
);
```

Parameters
scriptingComponent
A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

activeProc
On return, a pointer a UPP to the active function currently set for the specified scripting component.

refCon
On return, a pointer to the reference constant associated with the active function for the specified scripting component.

Return Value
A result code. See "Result Codes" (page 2885).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OSA.h`

OSAGetAppTerminology

Gets one or more scripting terminology resources from the specified file. (Deprecated in Mac OS X v10.5. Use `OSACopyScriptingDefinition` (page 2803) instead.)

```
OSAError OSAGetAppTerminology (
   ComponentInstance scriptingComponent,
   SInt32 modeFlags,
   FSSpec *fileSpec,
   short terminologyID,
   Boolean *didLaunch,
   AEDesc *terminologyList
);
```

Parameters

scriptingComponent

Identifies the current scripting component. See the Component Manager documentation for a description of the `ComponentInstance` data type.

modeFlags

Information for use by the scripting component. No mode flags are applicable for this function, so pass the value `kOSAModeNull`.

fileSpec

Specifies the file to search. See the File Manager documentation for a description of the `FSSpec` data type.

terminologyID

A dialect code obtained from a previous call to the `OSAGetDialectInfo` function or the `OSAGetCurrentDialect` function.

didLaunch

On return, has the value `true` if the application's scripting size resource or plist flags indicate that it has a dynamic terminology (in which case, the application will have been launched).

terminologyList

On return, a descriptor list containing zero or more terminology resources. See Apple Event Manager Reference for a description of the `AEDesc` data type.

Return Value

A result code. See "Result Codes" (page 2885).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`ASDebugging.h`

OSAGetCreateProc

Gets a pointer to the create function that a scripting component is currently using to create Apple events.

```
OSAError OSAGetCreateProc (
   ComponentInstance scriptingComponent,
   OSACreateAppleEventUPP *createProc,
   SRefCon *refCon
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

createProc

On return, a pointer to the UPP to the create function currently set for the specified scripting component.

refCon

On return, a pointer to the reference constant associated with the create function for the specified scripting component.

Return Value
A result code. See "Result Codes" (page 2885).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OSA.h`

OSAGetCurrentDialect

Gets the dialect code for the dialect currently being used by a scripting component.

```
OSAError OSAGetCurrentDialect (
   ComponentInstance scriptingComponent,
   short *resultingDialectCode
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

resultingDialectCode

On return, a pointer to the code for the current dialect of the specified scripting component.

Return Value
A result code. See "Result Codes" (page 2885).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OSA.h`

OSAGetDefaultScriptingComponent

Gets the subtype code for the default scripting component associated with an instance of the generic scripting component.

```
OSAError OSAGetDefaultScriptingComponent (
   ComponentInstance genericScriptingComponent,
   ScriptingComponentSelector *scriptingSubType
);
```

Parameters

genericScriptingComponent

A component instance for the generic scripting component, created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

scriptingSubType

On return, a pointer to the subtype code for the default scripting component associated with the instance of the generic scripting component specified in the *genericScriptingComponent* parameter.

Return Value

A result code. See "Result Codes" (page 2885).

Discussion

The `OSAGetDefaultScriptingComponent` function returns the subtype code for the default scripting component. This is the scripting component that will be used by `OSAStartRecording`, `OSACompile`, or `OSACompileExecute` if no existing script ID is specified. From the user's point of view, the default scripting component corresponds to the scripting language selected in the Script Editor application when the user first creates a new script.

Each instance of the generic scripting component has its own default scripting component, which is initially AppleScript. You can use `OSASetDefaultScriptingComponent` to change the default scripting component.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OSAGeneric.h`

OSAGetDialectInfo

Gets information about a specified dialect provided by a specified scripting component.

```
OSAError OSAGetDialectInfo (
   ComponentInstance scriptingComponent,
   short dialectCode,
   OSType selector,
   AEDesc *resultingDialectInfo
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

dialectCode

A code for the dialect about which you want information. You can obtain a list of a scripting component's dialect codes by calling `OSAAvailableDialectCodeList`.

selector

A constant that indicates what kind of information you want `OSAGetDialectInfo` to return in the result parameter. This constant determines the descriptor type for the descriptor record returned. See the description in "Dialect Descriptor Constants" (page 2865) for a list of the standard constants you can specify in this parameter.

resultingDialectInfo

A pointer to a descriptor record containing the requested information. The descriptor record's descriptor type corresponds to the constant specified in the `selector` parameter.

Return Value
A result code. See "Result Codes" (page 2885).

Discussion
After you obtain a list of dialect codes by calling `OSAAvailableDialectCodeList`, you can pass any of those codes to `OSAGetDialectInfo` to get information about the corresponding dialect. The descriptor type of the descriptor record returned by `OSAGetDialectInfo` depends on the constant specified in the *selector* parameter. All scripting components support the "Dialect Descriptor Constants" (page 2865) constants for this parameter. Individual scripting components may allow you to specify additional constants.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OSA.h`

OSAGetHandler

Gets a script ID for the specified script handler from the specified script.

```
OSAError OSAGetHandler (
   ComponentInstance scriptingComponent,
   SInt32 modeFlags,
   OSAID contextID,
   const AEDesc *handlerName,
   OSAID *resultingCompiledScriptID
);
```

Parameters

scriptingComponent

Identifies the current scripting component. See the Component Manager documentation for a description of the `ComponentInstance` data type.

modeFlags

Information for use by the scripting component. No mode flags are applicable for this function, so pass the value `kOSAModeNull`.

contextID

Specifies the script to get the script handler for. See the `OSAID` (page 2855) data type.

handlerName

A descriptor record that specifies the name of the handler to get. The descriptor must be of type `typeChar`, or of a type that can be coerced to `typeChar`. The handler name is case-sensitive and must exactly match the case of the handler name as supplied by the `OSAGetHandlerNames` function or the `OSAGetSource` (page 2830) function. See Apple Event Manager Reference for a description of the `AEDesc` data type.

resultingCompiledScriptID

On return, the `OSAID` for the specified handler, or `kOSANullScript` if the handler does not exist. If the handler has no input parameters, it may be executed by calling `OSAExecute`; if it requires input parameters, you can create an Apple event that supplies the handler parameters and execute it with `OSAExecuteEvent`. You may also copy it to another script with the `OSASetHandler` function or get its source code with the `OSAGetSource` function. See the `OSAID` (page 2855) data type.

Return Value
A result code. See "Result Codes" (page 2885).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ASDebugging.h`

OSAGetHandlerNames

Gets a list of all handler names in the specified script as an `AEDescList` of descriptors of type `typeChar`.

```
OSAError OSAGetHandlerNames (
   ComponentInstance scriptingComponent,
   SInt32 modeFlags,
   OSAID contextID,
   AEDescList *resultingHandlerNames
);
```

Parameters

scriptingComponent

Identifies the current scripting component. See the Component Manager documentation for a description of the `ComponentInstance` data type.

modeFlags

Information for use by the scripting component. No mode flags are applicable for this function, so pass the value `kOSAModeNull`.

contextID

See the `OSAID` (page 2855) data type.

resultingHandlerNames

On return, a list of all handler names, as an `AEDescList` of descriptors of type `typeChar`. See Apple Event Manager Reference for a description of the `AEDescList` data type.

Return Value
A result code. See "Result Codes" (page 2885).

Availability
Available in Mac OS X v10.0 and later.

Declared In
ASDebugging.h

OSAGetProperty

Gets the value of a specified script property from a specified script.

```
OSAError OSAGetProperty (
   ComponentInstance scriptingComponent,
   SInt32 modeFlags,
   OSAID contextID,
   const AEDesc *variableName,
   OSAID *resultingScriptValueID
);
```

Parameters

scriptingComponent
Identifies the current scripting component. See the Component Manager documentation for a description of the ComponentInstance data type.

modeFlags
Information for use by the scripting component. No mode flags are applicable for this function, so pass the value kOSAModeNull.

contextID
Specifies the script to get the script property from. See the OSAID (page 2855) data type.

variableName
A descriptor record that specifies the name of the property to get. The descriptor must be of type typeChar, or of a type that can be coerced to typeChar. The variable name is case-sensitive and must exactly match the case of the variable name as supplied by the OSAGetPropertyNames function or the OSAGetSource (page 2830) function. See Apple Event Manager Reference for a description of the AEDesc data type.

resultingScriptValueID
On return, a script ID whose associated data supplies the value for the property specified by the variableName parameter. Note that the value is returned as an OSAID, not an AEDesc. To get it as an AEDesc, use the OSACoerceToDesc function; to get it as user-readable text, use OSADisplay (page 2810). See the OSAID (page 2855) data type.

Return Value
A result code. See "Result Codes" (page 2885).

Availability
Available in Mac OS X v10.0 and later.

Declared In
ASDebugging.h

OSAGetPropertyNames

Gets a list of all property names from the specified script.

```
OSAError OSAGetPropertyNames (
   ComponentInstance scriptingComponent,
   SInt32 modeFlags,
   OSAID contextID,
   AEDescList *resultingPropertyNames
);
```

Parameters

scriptingComponent

Identifies the current scripting component. See the Component Manager documentation for a description of the `ComponentInstance` data type.

modeFlags

Information for use by the scripting component. No mode flags are applicable for this function, so pass the value `kOSAModeNull`.

contextID

Specifies the script to get the property names from. See the `OSAID` (page 2855) data type.

resultingPropertyNames

On return, a list of all property names, as an `AEDescList` of descriptors of type `typeChar`. You can extract these descriptors from the list and use them as input values to the `OSAGetProperty` function or the `OSASetProperty` (page 2844) function. See Apple Event Manager Reference for a description of the `AEDescList` data type.

Return Value

A result code. See "Result Codes" (page 2885).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ASDebugging.h`

OSAGetResumeDispatchProc

Gets the resume dispatch function currently being used by a scripting component instance during execution of an AppleScript `continue` statement or its equivalent

```
OSAError OSAGetResumeDispatchProc (
   ComponentInstance scriptingComponent,
   AEEventHandlerUPP *resumeDispatchProc,
   SRefCon *refCon
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

resumeDispatchProc

On return, a pointer to a UPP to the resume dispatch function for the specified scripting component. If no resume dispatch function has been registered, `OSAGetResumeDispatchProc` returns `kOSAUseStandardDispatch` (the default).

refCon

On return, a pointer to the reference constant associated with the resume dispatch function.

Return Value
A result code. See "Result Codes" (page 2885).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OSA.h`

OSAGetScriptInfo

Obtains information about script data according to the value you pass in the `selector` parameter.

```
OSAError OSAGetScriptInfo (
   ComponentInstance scriptingComponent,
   OSAID scriptID,
   OSType selector,
   long *result
);
```

Parameters

`scriptingComponent`
A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

`scriptID`
The script ID for the script data about which to obtain information. See the `OSAID` (page 2855) data type.

`selector`
A value that determines what kind of information `OSAGetScriptInfo` returns. The value can be one of the constants described in "Script Information Selectors" (page 2880). In addition to the standard constants, the AppleScript component also supports the `kASHasOpenHandler` constant. For additional information, see the Version Notes section below.

`result`
On return, a pointer to the requested information, which you can coerce to the appropriate descriptor type for the value specified in the *`selector`* parameter.

Return Value
A result code. See "Result Codes" (page 2885).

Version Notes
In Mac OS X, if you specify `kOSAScriptIsModified` for the value of the *`selector`* parameter, `OSAGetScriptInfo` returns `true` if the script has been modified and `false` if it has not.

The following information describes the behavior of `OSAGetScriptInfo` in versions of the Mac OS prior to Mac OS X: Although you can specify `kOSAScriptIsModified` when you are using the AppleScript component without generating an error, the current version of AppleScript interprets this request conservatively. The AppleScript component stores script data in a network of interlocking structures, and running a script can cause any of these structures to be modified. If you pass a script ID is to `OSAGetScriptInfo` with `kOSAScriptIsModified` as the value of the *`selector`* parameter, the AppleScript component returns 1 if there is any possibility that the script data or related structures may have been modified, and 0 if there is no possibility that they have been modified.

Availability
Available in Mac OS X v10.0 and later.

Declared In
OSA.h

OSAGetScriptingComponent

Gets the instance of a scripting component for a specified subtype.

```
OSAError OSAGetScriptingComponent (
   ComponentInstance genericScriptingComponent,
   ScriptingComponentSelector scriptingSubType,
   ComponentInstance *scriptingInstance
);
```

Parameters

genericScriptingComponent
A component instance for the generic scripting component, created by a prior call to the Component Manager function OpenDefaultComponent or OpenComponent.

scriptingSubType
A subtype code for a scripting component.

scriptingInstance
On return, a pointer to a component instance for the scripting component identified by the *scriptingSubType* parameter.

Return Value
A result code. See "Result Codes" (page 2885).

Discussion
You can't use the generic scripting component with component-specific routines. Instead, use an instance of the specific scripting component, which you can obtain with OSAGetScriptingComponent.

The OSAGetScriptingComponent function returns, in the *scriptingInstance* parameter, an instance of the scripting component identified by the *scriptingSubType* parameter. Each instance of the generic scripting component keeps track of a single instance of each component subtype, so OSAGetScriptingComponent always returns the same instance of a specified scripting component that the generic scripting component uses for standard scripting component routines.

For example, you can use OSAGetScriptingComponent to get the subtype code for the default scripting component (that is, the scripting component used by the generic scripting component for new scripts). You can then get an instance of the default scripting component by passing its subtype code to OSAGetScriptingComponent. Finally, you can pass that instance to OSAScriptingComponentName to obtain the default scripting component's name so you can display it to the user.

Similarly, you can pass kAppleScriptSubtype in the *scriptingSubType* parameter to obtain an instance of the AppleScript component. This is necessary, for example, to call AppleScript-specific routines such as ASGetSourceStyles (which is deprecated in Mac OS X version 10.5 in favor of ASCopySourceAttributes (page 2785)).

Availability
Available in Mac OS X v10.0 and later.

Declared In
OSAGeneric.h

OSAGetScriptingComponentFromStored

Gets the subtype code for a scripting component that created a storage descriptor record.

```
OSAError OSAGetScriptingComponentFromStored (
   ComponentInstance genericScriptingComponent,
   const AEDesc *scriptData,
   ScriptingComponentSelector *scriptingSubType
);
```

Parameters

genericScriptingComponent
A component instance for the generic scripting component, created by a prior call to the Component Manager function OpenDefaultComponent or OpenComponent.

scriptData
A pointer to either a generic storage descriptor record or a component-specific storage descriptor record.

scriptingSubType
On return, a pointer to a subtype code identifying the scripting component that created the descriptor record specified by the *scriptData* parameter.

Return Value
A result code. See "Result Codes" (page 2885).

Discussion
The OSAGetScriptingComponentFromStored function returns, in the *scriptingSubType* parameter, the subtype code for the scripting component that created the script data specified by the *scriptData* parameter.

The generic scripting component automatically identifies the appropriate scripting component for you when you use it to call OSALoad. By calling OSAGetScriptingComponentFromStored, you can determine, without loading a script, which scripting component created the script data.

Availability
Available in Mac OS X v10.0 and later.

Declared In
OSAGeneric.h

OSAGetSendProc

Gets a pointer to the send function that a scripting component is currently using.

```
OSAError OSAGetSendProc (
   ComponentInstance scriptingComponent,
   OSASendUPP *sendProc,
   SRefCon *refCon
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

sendProc

On return, a pointer to the UPP to the send function currently set for the specified scripting component.

refCon

On return, a pointer to the reference constant associated with the send function for the specified scripting component.

Return Value
A result code. See "Result Codes" (page 2885).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OSA.h`

OSAGetSource

Decompiles the script data identified by a script ID and obtains the equivalent source data.

```
OSAError OSAGetSource (
   ComponentInstance scriptingComponent,
   OSAID scriptID,
   DescType desiredType,
   AEDesc *resultingSourceData
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`

scriptID

The script ID for the script data to decompile. If you pass `kOSANullScript` in this parameter, `OSAGetSource` returns a null source description (such as an empty text string). See the `OSAID` (page 2855) data type.

desiredType

The desired descriptor type of the resulting descriptor record, or `typeBest` if any type will do.

resultingSourceData

A pointer to the resulting descriptor record.

Return Value
A result code. See "Result Codes" (page 2885).

Discussion
The `OSAGetSource` function decompiles the script data identified by the specified script ID and returns a descriptor record containing the equivalent source data. The source data returned need not be exactly the same as the source data originally passed to `OSACompile`—for example, white space and formatting might be different—but it should be a reasonable equivalent suitable for user viewing and editing.

The difference between `OSACoerceToDesc` and `OSAGetSource` is that `OSAGetSource` creates source data that can be displayed to a user or compiled and executed to generate an appropriate value, whereas `OSACoerceToDesc` actually returns the value. For example, if you call `OSAGetSource` and specify a string value, it returns the text surrounded by quotation marks (so that it can be properly compiled). If you call `OSACoerceToDesc` and specify a string value, it simply returns the text.

The main difference between `OSADisplay` and `OSAGetSource` is that `OSAGetSource` can coerce any form of script data using a variety of descriptor types, whereas `OSADisplay` can coerce only script values and always produces a descriptor record of a text descriptor type.

A scripting component that supports the `OSAGetSource` function has the `kOSASupportsGetSource` bit set in the `componentFlags` field of its component description record.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OSA.h`

OSAGetStorageType

Retrieves the scripting component subtype from the script trailer appended to the script data in a generic storage descriptor record.

```
OSErr OSAGetStorageType (
   AEDataStorage scriptData,
   DescType *dscType
);
```

Parameters

`scriptData`
A handle to the script data.

`dscType`
A pointer to the descriptor type specified in the script data trailer.

Return Value
A result code. See "Result Codes" (page 2885).

Discussion
The `OSAGetStorageType` function retrieves the scripting component subtype from the trailer. If no trailer can be found, `OSAGetStorageType` returns the error `errOSABadStorageType`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OSAComp.h`

OSAGetSysTerminology

Gets one or more scripting terminology resources from the OSA system.

```
OSAError OSAGetSysTerminology (
   ComponentInstance scriptingComponent,
   SInt32 modeFlags,
   short terminologyID,
   AEDesc *terminologyList
);
```

Parameters

scriptingComponent

Identifies the current scripting component. See the Component Manager documentation for a description of the `ComponentInstance` data type.

modeFlags

Information for use by the scripting component. No mode flags are applicable for this function, so pass the value `kOSAModeNull`.

terminologyID

terminologyList

On return, one or more terminology resources from the OSA system. These include the built-in terminology for AppleScript as well as the standard suites, but not the terminology for installed scripting additions. The terminology may be returned as a single `AEDesc` of type `typeAEUT` or as a list of such descriptors. The internal format of the `typeAEUT` descriptor is the resource format described in AEUserTermTypes.r. See Apple Event Manager Reference Apple Event Manager Reference for a description of the `AEDesc` data type.

Return Value

A result code. See "Result Codes" (page 2885).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ASDebugging.h`

OSALoad

Loads script data.

```
OSAError OSALoad (
   ComponentInstance scriptingComponent,
   const AEDesc *scriptData,
   SInt32 modeFlags,
   OSAID *resultingScriptID
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

scriptData

A pointer to the descriptor record containing the script data to be loaded.

modeFlags
: Information used by individual scripting components. To avoid setting any mode flags, specify `kOSAModeNull`. To indicate that only the minimum script data required to run the script should be loaded, pass `kOSAModePreventGetSource` in this parameter.

resultingScriptID
: On return, a pointer to the script ID for the compiled script. See the `OSAID` (page 2855) data type.

Return Value
A result code. See "Result Codes" (page 2885).

Discussion
The `OSALoad` function loads script data and returns a script ID. The generic scripting component uses the descriptor record in the *scriptData* parameter to determine which scripting component should load the script. If the descriptor record is of type `typeOSAGenericStorage`, the generic scripting component uses the trailer at the end of the script data to identify the scripting component. If the descriptor record's type is the subtype value for another scripting component, the generic scripting component uses the descriptor type to identify the scripting component.

If you want the script ID returned by `OSALoad` to identify only the minimum script data required to run the script and you are sure that you won't need to display the source data to the user, specify the `kOSAModePreventGetSource` flag in the *modeFlags* parameter.

Scripting components other than the generic scripting component can load script data only if it has been saved in a descriptor record whose descriptor type matches the scripting component's subtype.

Script data may change after it has been loaded—for example, if your application allows the user to edit a script's source data. To test whether script data has been modified, pass its script ID to `OSAGetScriptInfo`. If it has changed, you can call `OSAStore` again to obtain a handle to the modified script data and save it.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OSA.h`

OSALoadExecute

Loads and executes a script in a single step rather than calling `OSALoad` and `OSAExecute`.

```
OSAError OSALoadExecute (
   ComponentInstance scriptingComponent,
   const AEDesc *scriptData,
   OSAID contextID,
   SInt32 modeFlags,
   OSAID *resultingScriptValueID
);
```

Parameters

scriptingComponent
: A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

scriptData
: A pointer to the descriptor record identifying the script data to be loaded and executed.

contextID
The script ID for the context to be used during script execution. The constant `kOSANullScript` in this parameter indicates that the scripting component should use its default context. See the `OSAID` (page 2855) data type.

modeFlags
Information used by individual scripting components. To avoid setting mode flag values, specify `kOSAModeNull`. Other possible mode flags are listed in "Mode Flags" (page 2873).

resultingScriptValueID
A pointer to the script ID for the script value returned. See the `OSAID` (page 2855) data type.

Return Value
A result code. See "Result Codes" (page 2885).

Discussion
The `OSALoadExecute` function loads script data and executes the resulting compiled script, using the script context identified by the *contextID* parameter to maintain state information such as the binding of variables. After successfully executing the script, `OSALoadExecute` disposes of the compiled script and returns either the script ID for the resulting script value or, if execution does not result in a value, the constant `kOSANullScript`.

You can control the way in which the scripting component executes a script by adding any of the "Mode Flags" (page 2873) flags to the *modeFlags* parameter.

If the result code returned by `OSALoadExecute` is a general result code, there was some problem in arranging for the script to be run. If the result code is `errOSAScriptError`, an error occurred during script execution. In this case, you can obtain more detailed error information by calling `OSAScriptError`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OSA.h`

OSALoadExecuteFile

Loads a script from the specified file into the specified scripting component, compiles the script if the file is a text file, and executes the script.

```
OSAError OSALoadExecuteFile (
   ComponentInstance scriptingComponent,
   const FSRef *scriptFile,
   OSAID contextID,
   SInt32 modeFlags,
   OSAID *resultingScriptValueID
);
```

Parameters

scriptingComponent
A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`. See the Component Manager documentation for a description of the `ComponentInstance` data type.

scriptFile

Identifies the file to load the script from. See the File Manager documentation for a description of the `FSRef` data type.

File format is determined by inspection. If the file is a text file, `OSALoadExecuteFile` uses the following steps to determine the text encoding:

- If a Unicode BOM is present, that determines the encoding—one of UTF-16BE, UTF-16LE, or UTF-8
- Otherwise, if the file is valid UTF-8, it is presumed to be UTF-8.
- Otherwise, it is assumed to be in the primary encoding.

contextID

The script ID for the context to be used during script execution. The constant `kOSANullScript` in this parameter indicates that the scripting component should use its default context. See the `OSAID` (page 2855) data type.

modeFlags

Information for use by the scripting component. Can include any of the mode flags that would normally be sent to the `OSACompileExecute` (page 2800) (if the file is a text file) and `OSALoadExecute` (page 2833) functions. For descriptions of the mode flag usage of those functions, see the chapter "Scripting Components" in "Interapplication Communication" at http://developer.apple.com/documentation/mac/IAC/IAC-2.html.

resultingScriptValueID

The script ID for the resulting script value. See the `OSAID` (page 2855) data type.

Return Value
A result code. See "Result Codes" (page 2885).

Discussion
This routine is effectively equivalent to calling `OSALoadFile` (page 2835) followed by `OSAExecute` (page 2816). After execution, the compiled source is disposed. Only the resulting value ID is retained.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`OSA.h`

OSALoadFile

Loads a script from the specified file into the specified scripting component, compiling the script if the file is a text file.

```
OSAError OSALoadFile (
   ComponentInstance scriptingComponent,
   const FSRef *scriptFile,
   Boolean *storable,
   SInt32 modeFlags,
   OSAID *resultingScriptID
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`. See the Component Manager documentation for a description of the `ComponentInstance` data type.

scriptFile

Identifies the file to load the script from. See the File Manager documentation for a description of the `FSRef` data type.

File format is determined by inspection. If the file is a text file, `OSALoadFile` uses the following steps to determine the text encoding:

- If a Unicode BOM is present, that determines the encoding—one of UTF-16BE, UTF-16LE, or UTF-8
- Otherwise, if the file is valid UTF-8, it is presumed to be UTF-8.
- Otherwise, it is assumed to be in the primary encoding.

storable

If `storable` is not `NULL`, on return it is set to indicate whether a compiled script can be stored into the script file using `OSAStoreFile` (page 2851).

modeFlags

Information for use by the scripting component. Can include any of the mode flags that would normally be sent to the `OSACompile` (page 2799) (if the file is a text file) and `OSALoad` (page 2832) functions. For descriptions of the mode flag usage of those functions, see the chapter "Scripting Components" in "Interapplication Communication" at http://developer.apple.com/documentation/mac/IAC/IAC-2.html.

resultingScriptID

The returned script ID for the compiled script. See the `OSAID` (page 2855) data type.

Return Value

A result code. See "Result Codes" (page 2885).

Availability

Available in Mac OS X v10.3 and later.

Declared In

`OSA.h`

OSAMakeContext

Gets a script ID for a new script context.

```
OSAError OSAMakeContext (
   ComponentInstance scriptingComponent,
   const AEDesc *contextName,
   OSAID parentContext,
   OSAID *resultingContextID
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

contextName

A pointer to the name of the new context. Some scripting components may use context names for semantic purposes. If the value of this parameter is `typeNull`, `OSAMakeContext` creates an unnamed context.

parentContext

The existing context from which the new context inherits bindings. If the value of this parameter is `kOSANullScript`, the new context does not inherit bindings from any other context.

resultingContextID

A pointer to the script ID for the resulting script context. See the `OSAID` (page 2855) data type.

Return Value
A result code. See "Result Codes" (page 2885).

Discussion
The `OSAMakeContext` function creates a new script context that you may pass to `OSAExecute` or `OSAExecuteEvent`. The new script context inherits the bindings of the script context specified in the *parentContext* parameter.

If you call `OSAMakeContext` using an instance of the generic scripting component, the generic scripting component uses the default scripting component to create the new script context.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OSA.h`

OSARealToGenericID

Converts a component-specific script ID to the corresponding generic script ID.

```
OSAError OSARealToGenericID (
   ComponentInstance genericScriptingComponent,
   OSAID *theScriptID,
   ComponentInstance theExactComponent
);
```

Parameters

genericScriptingComponent

A component instance for the generic scripting component, created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

theScriptID

A pointer to the component-specific script ID that you want to convert. You must have obtained this script ID from the scripting component instance passed in the *theExactComponent* parameter. The OSARealToGenericID function returns, in this parameter, the generic script ID that corresponds to the component-specific script ID that you pass in this parameter. See the OSAID (page 2855) data type.

theExactComponent

A scripting component instance returned by a generic scripting component routine.

Return Value
A result code. See "Result Codes" (page 2885).

Discussion
The OSARealToGenericID function performs the reverse of the task performed by OSAGenericToRealID. Given a component-specific script ID and an exact scripting component instance (that is, the component instance that created the component-specific script ID), the OSARealToGenericID function returns the corresponding generic script ID. The OSARealToGenericID function modifies the script ID in place, changing the component-specific script ID passed in the *theScriptID* parameter to the corresponding generic script ID.

You'll need to do this if you have obtained a component-specific script ID using an exact scripting component instance and you want to refer to the same script in calls that use an instance of the generic scripting component. You can't use a component-specific script ID with the generic scripting component.

The script ID you pass in the *theScriptID* parameter must be a component-specific script ID obtained from a scripting component instance known to the generic scripting component. You can obtain such an instance by calling either OSAGetScriptingComponent or OSAGenericToRealID.

Availability
Available in Mac OS X v10.0 and later.

Declared In
OSAGeneric.h

OSARemoveStorageType

Removes a trailer from the script data in a generic storage descriptor record

```
OSErr OSARemoveStorageType (
   AEDataStorage scriptData
);
```

Parameters

scriptData

A handle to the script data.

Return Value
A result code. See "Result Codes" (page 2885).

Discussion
The OSARemoveStorageType function removes an existing trailer (reducing the handle's size). If no trailer can be found, then the handle is not modified, and noErr is returned.

Availability
Available in Mac OS X v10.0 and later.

Declared In
OSAComp.h

OSAScriptError

Gets information about errors that occur during script execution.

```
OSAError OSAScriptError (
   ComponentInstance scriptingComponent,
   OSType selector,
   DescType desiredType,
   AEDesc *resultingErrorDescription
);
```

Parameters

scriptingComponent
A component instance created by a prior call to the Component Manager function OpenDefaultComponent or OpenComponent.

selector
A value that determines what OSAScriptError returns. The value can be one of the constants described in "OSAScriptError Selectors" (page 2877).

desiredType
The desired descriptor type of the resulting descriptor record. The description that follows explains how this is determined by the value passed in the selector parameter.

resultingErrorDescription
On return, a pointer to the resulting descriptor record.

Return Value
A result code. See "Result Codes" (page 2885).

Discussion
Whenever the OSAExecute function returns the error errOSAScriptError, you can use the OSAScriptError function to get more specific information about the error from the scripting component that encountered it. (This information remains available only until the next call to the same scripting component.) The information returned by OSAScriptError depends on the value passed in the *selector* parameter, which also determines the descriptor type you should specify in the desiredType parameter.

Every scripting component should support calls to OSAScriptError that pass kOSAErrorNumber, kOSAErrorMessage, or kOSAErrorPartialResult in the *selector* parameter.

Some scripting components may also support calls that pass other values in the selector parameter, including kOSAErrorRange, which provides start and end positions delimiting the errant expression in the source data. If the value of the *selector* parameter is kOSAErrorRange, the value of *desiredType* must be typeOSAErrorRange.

If the value of the selector parameter is kOSAErrorNumber, scripting components may return, in the *resultingErrorDescription* parameter, one of the general error codes described in "Result Codes" (page 2885).

If you call OSAScriptError using an instance of the generic scripting component, the generic scripting component uses the same instance of a scripting component that it used for the previous call.

Availability
Available in Mac OS X v10.0 and later.

Declared In
OSA.h

OSAScriptingComponentName

Gets the name of a scripting component.

```
OSAError OSAScriptingComponentName (
   ComponentInstance scriptingComponent,
   AEDesc *resultingScriptingComponentName
);
```

Parameters

scriptingComponent
A component instance created by a prior call to the Component Manager function OpenDefaultComponent or OpenComponent.

resultingScriptingComponentName
On return, a pointer to the name of the scripting component; or, if the component is the generic scripting component, the name of the default scripting component.

Return Value
A result code. See "Result Codes" (page 2885).

Discussion
The OSAScriptingComponentName function returns a descriptor record that you can coerce to a text descriptor type such as typeChar. This can be useful if you want to display the name of the scripting language in which the user should write a new script.

Availability
Available in Mac OS X v10.0 and later.

Declared In
OSA.h

OSASetActiveProc

Sets the active function that a scripting component calls periodically while executing a script.

```
OSAError OSASetActiveProc (
   ComponentInstance scriptingComponent,
   OSAActiveUPP activeProc,
   SRefCon refCon
);
```

Parameters

scriptingComponent
A component instance created by a prior call to the Component Manager function OpenDefaultComponent or OpenComponent.

activeProc

A pointer to the active function to set. If the value of this parameter is `NULL`, `OSASetActiveProc` sets the scripting component's default active function.

refCon

A reference constant to be associated with the active function. This parameter can be used for many purposes; for example, it could contain a handle to data used by the active function.

Return Value
A result code. See "Result Codes" (page 2885).

Discussion
The `OSASetActiveProc` function allows your application to set a pointer to the active function called periodically by the scripting component during script execution. To get time periodically during script execution for its own purposes, your application can substitute its own active function for use by the scripting component. If you do not specify an active function, the scripting component uses its default active function, which allows a user to cancel script execution.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OSA.h`

OSASetCreateProc

Specifies a create function that a scripting component should use instead of the Apple Event Manager's `AECreateAppleEvent` function when creating Apple events.

```
OSAError OSASetCreateProc (
   ComponentInstance scriptingComponent,
   OSACreateAppleEventUPP createProc,
   SRefCon refCon
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

createProc

A universal procedure pointer to the create function to set.

refCon

A reference constant.

Return Value
A result code. See "Result Codes" (page 2885).

Discussion
To gain control over the creation and addressing of Apple events, your application can provide its own create function for use by scripting components. To set a new create function, call the `OSASetCreateProc` function; to get the current create function, call `OSAGetCreateProc`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
OSA.h

OSASetCurrentDialect

Sets the current dialect for a scripting component.

```
OSAError OSASetCurrentDialect (
   ComponentInstance scriptingComponent,
   short dialectCode
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function OpenDefaultComponent or OpenComponent.

dialectCode

The code for the dialect to be set.

Return Value
A result code. See "Result Codes" (page 2885).

Availability
Available in Mac OS X v10.0 and later.

Declared In
OSA.h

OSASetDefaultScriptingComponent

Sets the default scripting component associated with an instance of the generic scripting component.

```
OSAError OSASetDefaultScriptingComponent (
   ComponentInstance genericScriptingComponent,
   ScriptingComponentSelector scriptingSubType
);
```

Parameters

genericScriptingComponent

A component instance for the generic scripting component, created by a prior call to the Component Manager function OpenDefaultComponent or OpenComponent.

scriptingSubType

The subtype code for the scripting component you want to set as the default.

Return Value
A result code. See "Result Codes" (page 2885).

Discussion
The OSASetDefaultScriptingComponent function sets the default scripting component for the specified instance of the generic scripting component to the scripting component identified by the *scriptingSubType* parameter.

Each instance of the generic scripting component has its own default scripting component, which is initially AppleScript. You can use `OSAGetDefaultScriptingComponent` to get the current default scripting component for an instance of the generic scripting component.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OSAGeneric.h`

OSASetDefaultTarget

Sets the default target application for Apple events.

```
OSAError OSASetDefaultTarget (
   ComponentInstance scriptingComponent,
   const AEAddressDesc *target
);
```

Parameters

`scriptingComponent`
A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

`target`
The address of the application that is being made the default application. If you pass a null descriptor record in this parameter, the scripting component treats the current process as the default target.

Return Value
A result code. See "Result Codes" (page 2885).

Discussion
Scripting components that support manipulation of the create and send functions also support the `OSASetDefaultTarget` function. The `OSASetDefaultTarget` function establishes the default target application for Apple event sending and the default application from which the scripting component should obtain terminology information. For example, AppleScript statements that refer to the default application do not need to be enclosed in `tell/end tell` statements.

If your application doesn't call this function, or if you pass a null descriptor record in the target parameter, the scripting component treats the current process (that is, the application that calls `OSAExecute` or related functions) as the default target application.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OSA.h`

OSASetHandler

Sets a specified script handler in the specified script to the supplied handler.

```
OSAError OSASetHandler (
   ComponentInstance scriptingComponent,
   SInt32 modeFlags,
   OSAID contextID,
   const AEDesc *handlerName,
   OSAID compiledScriptID
);
```

Parameters

scriptingComponent

Identifies the current scripting component. See the Component Manager documentation for a description of the `ComponentInstance` data type.

modeFlags

Information for use by the scripting component. Pass the value `kOSAModeDontDefine` to prevent a handler from being created if it doesn't already exist. Otherwise, pass `kOSAModeNull` to avoid setting mode flag values (no other flags are applicable for this function).

contextID

Specifies the script to set the script handler for. See `OSAID` (page 2855) for a description of the `OSAID` data type.

handlerName

A descriptor record that specifies the handler to set. The descriptor must be of type `typeChar`, or of a type that can be coerced to `typeChar`. If the handler does not already exist, it is created, unless you pass the value `kOSAModeDontDefine` for the `modeFlags` parameter. The handler name is case-sensitive and must exactly match the case of the handler name as supplied by the `OSAGetHandlerNames` function or the `OSAGetSource` (page 2830) function. See Apple Event Manager Reference for a description of the `AEDesc` data type.

compiledScriptID

The `OSAID` value to set the handler to, normally obtained by a previous call to `OSAGetHandler`. Any other value will return an error value of `errOSAInvalidID`. Note that a script compiled by `OSACompile` is not itself a handler. See the `OSAID` (page 2855) data type.

Return Value

A result code. See "Result Codes" (page 2885).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ASDebugging.h`

OSASetProperty

Sets the value of a script property in a specified script, creating the property if it does not already exist.

```
OSAError OSASetProperty (
   ComponentInstance scriptingComponent,
   SInt32 modeFlags,
   OSAID contextID,
   const AEDesc *variableName,
   OSAID scriptValueID
);
```

Parameters

scriptingComponent

See the Component Manager documentation for a description of the `ComponentInstance` data type.

modeFlags

Information for use by the scripting component. Pass the value `kOSAModeDontDefine` to prevent a property from being created if it doesn't already exist in the specified script. Otherwise, pass `kOSAModeNull` to avoid setting mode flag values (no other flags are applicable for this function).

contextID

Specifies the script to set the script property for. See the `OSAID` (page 2855) data type.

variableName

A descriptor record that specifies the name of the property to set. The descriptor must be of type `typeChar`, or of a type that can be coerced to `typeChar`. The variable name is case-sensitive and must exactly match the case of the variable name as supplied by the `OSAGetPropertyNames` function or the `OSAGetSource` (page 2830) function. See Apple Event Manager Reference for a description of the `AEDesc` data type.

scriptValueID

A script ID whose associated data should be used to set the value for the property specified by `variableName`. Note that the value is specified by an `OSAID`, not an `AEDesc`. You can set a property to a value returned from script execution (from the `OSAExecute` function), extracted from another property (with the `OSAGetProperty` function), or converted from an `AEDesc` (by the `OSACoerceFromDesc` function). See the `OSAID` (page 2855) data type.

Return Value
A result code. See "Result Codes" (page 2885).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`ASDebugging.h`

OSASetResumeDispatchProc

Sets the resume dispatch function called by a scripting component during execution of an AppleScript `continue` statement or its equivalent.

```
OSAError OSASetResumeDispatchProc (
   ComponentInstance scriptingComponent,
   AEEventHandlerUPP resumeDispatchProc,
   SRefCon refCon
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

resumeDispatchProc

A UPP to the resume dispatch function. You can specify one of the following in this parameter:

- a pointer to a resume dispatch function
- the `kOSAUseStandardDispatch` constant, which causes the Apple Event Manager to dispatch the event using standard Apple event dispatching (the handler registered in the application with AEInstallEventHandler is used)
- the `kOSANoDispatch` constant, which tells the Apple Event Manager that the processing of the Apple event is complete and that no dispatching should occur

refCon

A reference constant. You can pass the constant `kOSADontUsePhac` in this parameter, as described in the Discussion section below.

Return Value
A result code. See "Result Codes" (page 2885).

Discussion
The `OSASetResumeDispatchProc` function sets the resume dispatch function that the specified instance of a scripting component calls during execution of an AppleScript continue statement or its equivalent. The resume dispatch function should dispatch the event to the application's standard handler for that event.

If you are using a general handler for preliminary processing of Apple events, and if you can rely on standard Apple event dispatching to dispatch the event correctly, you don't need to provide a resume dispatch function. Instead, you can specify `kOSAUseStandardDispatch` as the value of the *resumeDispatchProc* parameter and the constant `kOSADontUsePhac` as the value of the *refCon* parameter. This causes the Apple Event Manager to use standard Apple event dispatching except that it bypasses your application's special handler dispatch table and thus won't call your predispatch Apple event handler recursively. (A predispatch handler is called immediately before the Apple Event Manager dispatches an event.)

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OSA.h`

OSASetScriptInfo

Sets information about script data according to the value you pass in the selector parameter.

```
OSAError OSASetScriptInfo (
   ComponentInstance scriptingComponent,
   OSAID scriptID,
   OSType selector,
   long value
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

scriptID

The script ID for the script data whose information is to be set. See the `OSAID` (page 2855) data type.

selector

A value that determines which information `OSASetScriptInfo` sets.

The value can be one of the constants described in "Script Information Selectors" (page 2880). For more information, see the Version Notes section below.

In Mac OS X, the AppleScript component does not set a value.

value

The value to set.

In Mac OS X, the AppleScript component does not set a value.

Return Value
A result code. See "Result Codes" (page 2885).

Version Notes
In Mac OS X, if you specify `kOSAScriptIsModified` for the value of the *selector* parameter, it is ignored, and no value is set.

The following information describes the behavior of `OSASetScriptInfo` in versions of the Mac OS prior to Mac OS X: The `OSASetScriptInfo` function sets script information according to the value you pass in the *selector* parameter. If you use the `kOSAScriptIsModified` constant, `OSASetScriptInfo` sets a value that indicates whether the script data has been modified since it was created or passed to `OSALoad`. Some scripting components may provide additional constants.

For related information, see the `OSAGetScriptInfo` (page 2827) function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OSA.h`

OSASetSendProc

Specifies a send function that a scripting component should use instead of the Apple Event Manger's `AESend` function when sending Apple events.

```
OSAError OSASetSendProc (
   ComponentInstance scriptingComponent,
   OSASendUPP sendProc,
   SRefCon refCon
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

sendProc

A universal procedure pointer (UPP) to the send function to set.

refCon

A reference constant.

Return Value
A result code. See "Result Codes" (page 2885).

Discussion
The send function provided by your application can perform almost any action instead of or in addition to sending Apple events; for example, it can be used to facilitate concurrent script execution. To set a new send function, call the `OSASetSendProc` function; to get the current send function, call `OSAGetSendProc`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OSA.h`

OSAStartRecording

Turns on Apple event recording and records subsequent Apple events in a compiled script.

```
OSAError OSAStartRecording (
   ComponentInstance scriptingComponent,
   OSAID *compiledScriptToModifyID
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

compiledScriptToModifyID

A pointer to the script ID for the compiled script in which to record. See the `OSAID` (page 2855) data type.

Return Value
A result code. See "Result Codes" (page 2885).

Discussion
The `OSAStartRecording` function turns on Apple event recording. Subsequent Apple events are recorded (that is, appended to any existing statements) in the compiled script specified by the *compiledScriptToModifyID* parameter. If the source data for the compiled script is currently displayed in a script editor's window, the script editor's handler for the Recorded Text event should display each new

statement in the window as it is recorded. Users should not be able to change a script that is open in a script editor window while it is being recorded into. Recording continues until a call to `OSAStopRecording` turns recording off.

To record into a new compiled script, pass the constant `kOSANullScript` in the *`compiledScriptToModifyID`* parameter. The scripting component should respond by creating a new compiled script and recording into that.

The generic scripting component uses its default scripting component to create and record into a new compiled script.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OSA.h`

OSAStopRecording

Turns off Apple event recording.

```
OSAError OSAStopRecording (
   ComponentInstance scriptingComponent,
   OSAID compiledScriptID
);
```

Parameters

`scriptingComponent`

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

`compiledScriptID`

A script ID for the compiled script into which Apple events are being recorded. See the `OSAID` (page 2855) data type.

Return Value
A result code. See "Result Codes" (page 2885).

Discussion
The `OSAStopRecording` function turns off recording. If the script is not currently open in a script editor window, the *`compiledScriptToModifyID`* parameter supplied to `OSAStartRecording` is then augmented to contain the newly recorded statements. If the script is currently open in a script editor window, the script data that corresponds to the *`compiledScriptToModifyID`* parameter supplied to `OSAStartRecording` is updated continuously until the client application calls `OSAStopRecording`.

If the compiled script identified by the script ID in the *`compiledScriptID`* parameter is not being recorded into or recording is not currently on, `OSAStopRecording` returns `noErr`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OSA.h`

OSAStore

Gets a handle to script data in the form of a storage descriptor record.

```
OSAError OSAStore (
   ComponentInstance scriptingComponent,
   OSAID scriptID,
   DescType desiredType,
   SInt32 modeFlags,
   AEDesc *resultingScriptData
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

scriptID

The script ID for the script data for which to obtain a data handle.

desiredType

The desired type of the descriptor record to be returned. If you want to store the script data in the form used by a generic storage descriptor record, specify `typeOSAGenericStorage`.

modeFlags

Information used by individual scripting components. To avoid setting any mode flags, specify `kOSAModeNull`. To indicate that only the minimum script data required to run the script should be returned, pass `kOSAModePreventGetSource` in this parameter. (In this case the script data returned is not identical to the compiled script data and can't be used to generate source data.) If the `scriptID` parameter identifies a script context, you can pass `kOSAModeDontStoreParent` in this parameter to store the script context without storing its parent context.

resultingScriptData

On return, a pointer to the resulting descriptor record.

Return Value

A result code. See "Result Codes" (page 2885).

Discussion

The `OSAStore` function writes script data to a descriptor record so that the data can later be saved in a resource or written to the data fork of a document. You can then reload the data for the descriptor record as a compiled script (although possibly with a different script ID) by passing the descriptor record to `OSALoad`.

If you want the returned script data to be as small as possible and you are sure that you won't need to display the source data to the user, specify the `kOSAModePreventGetSource` flag in the `modeFlags` parameter. If the `scriptID` parameter identifies a script context and you don't want the returned script data to include the associated parent context, specify the `kOSAModeDontStoreParent` flag in the `modeFlags` parameter.

The desired type is either `typeOSAGenericStorage` (for a generic storage descriptor record) or a specific scripting component subtype value (for a component-specific storage descriptor record).

To store either a generic storage descriptor record or a component-specific storage descriptor record with your application's resources, use `'scpt'` as the resource type. The generic scripting component subtype, the generic storage descriptor type, and the resource type for stored script data all have the same value, though they serve different purposes.

Availability

Available in Mac OS X v10.0 and later.

Declared In
OSA.h

OSAStoreFile

Stores a script into the specified file.

```
OSAError OSAStoreFile (
   ComponentInstance scriptingComponent,
   OSAID scriptID,
   DescType desiredType,
   SInt32 modeFlags,
   const FSRef *scriptFile
);
```

Parameters

scriptingComponent
A component instance created by a prior call to the Component Manager function OpenDefaultComponent or OpenComponent. See the Component Manager documentation for a description of the ComponentInstance data type.

scriptID
Specifies the script to store. See the OSAID (page 2855) data type.

desiredType
Specifies how the script should be stored. The desired type is either typeOSAGenericStorage (for a generic storage descriptor record) or a specific scripting component subtype value (for a component-specific storage descriptor record).

modeFlags
Information used by individual scripting components. To avoid setting any mode flags, specify kOSAModeNull. To indicate that only the minimum script data required to run the script should be stored, pass kOSAModePreventGetSource in this parameter. (In this case the stored script data is not identical to the compiled script data and can't be used to generate source data.) If the scriptID parameter identifies a script context, you can pass kOSAModeDontStoreParent in this parameter to store the script context without storing its parent context.

scriptFile
Identifies the file to store the script into. See the File Manager documentation for a description of the FSRef data type.

Return Value
A result code. See "Result Codes" (page 2885).

Availability
Available in Mac OS X v10.3 and later.

Declared In
OSA.h

Callbacks

Your application can provide alternative active, send, and create functions for use by scripting components during script execution. All scripting components support routines that allow you to set and get the current active function called periodically by the scripting component during script execution. Some scripting components also support routines that allow you to set and get the current send and create functions used by the scripting component when it creates and sends Apple events during script execution.

OSAActiveProcPtr

Defines a pointer to an application-defined active function that performs periodic tasks during script compilation such as checking for Command-period, spinning the cursor, and checking for system-level errors.

```
typedef OSErr (*OSAActiveProcPtr) (
    long refCon
);
```

If you name your function `MyOSAActiveProc`, you would declare it like this:

```
OSErr MyOSAActiveProc (
    long refCon
);
```

Parameters

refCon

A reference constant.

Return Value

A result code. See "Result Codes" (page 2885).

Discussion

Every scripting component calls an active function periodically during script compilation and execution and provides routines that allow your application to set or get the pointer to the active function.

If you don't set an alternative active function for a scripting component, it uses its own default active function. A scripting component's default active function allows a user to cancel script execution by pressing Command-period and calls `WaitNextEvent` to give other processes time.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OSA.h`

OSACreateAppleEventProcPtr

Defines a pointer to an application-defined create function that allows you to gain control over the creation and addressing of Apple events.

```
typedef OSErr (*OSACreateAppleEventProcPtr)
(
    AEEventClass theAEEventClass,
    AEEventID theAEEventID,
    const AEAddressDesc * target,
    short returnID,
    long transactionID,
    AppleEvent * result,
    long refCon
);
```

If you name your function `MyOSACreateAppleEventProc`, you would declare it like this:

```
OSErr MyOSACreateAppleEventProc (
    AEEventClass theAEEventClass,
    AEEventID theAEEventID,
    const AEAddressDesc * target,
    short returnID,
    long transactionID,
    AppleEvent * result,
    long refCon
);
```

Parameters

`theAEEventClass`

The event class of the Apple event to create.

`theAEEventID`

The event ID of the Apple event to create.

`target`

A pointer to an address descriptor. This descriptor identifies the target (or server) application for the Apple event.

`returnID`

The return ID for the created Apple event.

`transactionID`

The transaction ID for this Apple event. A transaction is a series of Apple events that are sent back and forth between the client and server applications, beginning with the client's initial request for a service. All Apple events that are part of a transaction must have the same transaction ID. The constant `kAnyTransactionID` specifies that the Apple event is not one of a series of interdependent Apple events.

`result`

A pointer to an Apple event. On successful return, this parameter should point to the new Apple event. On error, this should be a `NULL` descriptor.

`refCon`

A reference constant.

Return Value

A result code. See "Result Codes" (page 2885).

Discussion

Every scripting component calls a create function whenever it creates an Apple event during script execution and provides routines that allow you to set or get the pointer to the create function.

Providing your own create function can be useful, for example, if your application needs to add its own transaction code to the event. An alternative create function takes the same parameters as the `AECreateAppleEvent` function plus a reference constant.

If you don't set an alternative create function for a scripting component, it uses the standard Apple Event Manager function `AECreateAppleEvent`, which it calls with its own default parameters.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OSA.h`

OSASendProcPtr

Defines a pointer to an application-defined send function that performs almost any action instead of or in addition to sending Apple events.

```
typedef OSErr (*OSASendProcPtr) (
    const AppleEvent * theAppleEvent,
    AppleEvent * reply,
    AESendMode sendMode,
    AESendPriority sendPriority,
    long timeOutInTicks,
    AEIdleUPP idleProc,
    AEFilterUPP filterProc,
    long refCon
);
```

If you name your function `MyOSASendProc`, you would declare it like this:

```
OSErr MyOSASendProc (
    const AppleEvent * theAppleEvent,
    AppleEvent * reply,
    AESendMode sendMode,
    AESendPriority sendPriority,
    long timeOutInTicks,
    AEIdleUPP idleProc,
    AEFilterUPP filterProc,
    long refCon
);
```

Parameters

theAppleEvent
A pointer to the Apple event.

reply
A pointer to a reply Apple event.

sendMode
Specifies various options for how the Apple event should be handled.

sendPriority
A value that specifies the priority for processing the Apple event.

timeOutInTicks

If the reply mode specified in the *sendMode* parameter is `kAEWaitReply`, or if a return receipt is requested, this parameter specifies the length of time (in ticks) that the client application is willing to wait for the reply or return receipt before timing out. If this parameter is `kNoTimeOut`, the Apple event never times out.

idleProc

A universal procedure pointer to a function that handles events (such as update, operating-system, activate, and null events) received while waiting for a reply.

filterProc

A universal procedure pointer to a function that determines which incoming Apple events should be received while the handler waits for a reply or a return receipt. This parameter may be `NULL`.

refCon

A reference constant.

Return Value
A result code. See "Result Codes" (page 2885).

Discussion
Every scripting component calls a send function whenever it sends an Apple event during script execution and provides routines that allow you to set or get the pointer to the send function.

For example, before sending an Apple event, an alternative send function can modify the event or save a copy of the event. An alternative send function takes the same parameters as the `AESend` function plus a reference constant.

If you don't set an alternative send function for a scripting component, it uses the standard Apple Event Manager function `AESend`, which it calls with its own default parameters.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OSA.h`

Data Types

OSAID

Used by a scripting component to keep track of script data in memory.

```
typedef unsigned long OSAID;
```

Discussion
A scripting component assigns a script ID when it creates the associated script data (that is, a compiled script, a script value, a script context, or other kinds of script data supported by a scripting component) or loads it into memory. The scripting routines that create, load, compile, and execute scripts all return script IDs, and your application must pass valid script IDs to the other routines that manipulate scripts. A script ID remains valid until a client application calls `OSADispose` to reclaim the memory used for the corresponding script data.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OSA.h`

GenericID

Represents the ID for generic scripting components.

```
typedef OSAID GenericID;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OSAGeneric.h`

OSAError

Represents an OSA result code.

```
typedef ComponentResult OSAError;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OSA.h`

ScriptingComponentSelector

```
typedef OSType ScriptingComponentSelector;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`OSAGeneric.h`

StatementRange

```
struct StatementRange {
    unsigned long startPos;
    unsigned long endPos;
};
typedef struct StatementRange StatementRange;
```

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In
OSA.h

OSAActiveUPP

Defines a universal procedure pointer (UPP) to an application-defined active function.

```
typedef OSAActiveProcPtr OSAActiveUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
OSA.h

OSACreateAppleEventUPP

Defines a universal procedure pointer (UPP) to an application-defined Apple event creation function.

```
typedef OSACreateAppleEventProcPtr OSACreateAppleEventUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
OSA.h

OSASendUPP

Defines a universal procedure pointer (UPP) to an application-defined send function.

```
typedef OSASendProcPtr OSASendUPP;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
OSA.h

OSADebugCallFrameRef

```
typedef OSAID OSADebugCallFrameRef;
```

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In
OSA.h

OSADebugSessionRef

```
typedef OSAID OSADebugSessionRef;
```

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In
`OSA.h`

Constants

cClosure

```
enum {
    cClosure = 'clsr',
    cRawData = 'rdat',
    cStringClass = typeChar,
    cNumber = 'nmbr',
    cListElement = 'celm',
    cListOrRecord = 'lr  ',
    cListOrString = 'ls  ',
    cListRecordOrString = 'lrs ',
    cNumberOrString = 'ns  ',
    cNumberOrDateTime = 'nd  ',
    cNumberDateTimeOrString = 'nds ',
    cAliasOrString = 'sf  ',
    cSeconds = 'scnd',
    typeSound = 'snd ',
    enumBooleanValues = 'boov',
    kAETrue = typeTrue,
    kAEFalse = typeFalse,
    enumMiscValues = 'misc',
    kASCurrentApplication = 'cura',
    formUserPropertyID = 'usrp'
};
```

cCoercion

```
enum {
    cCoercion = 'coec',
    cCoerceUpperCase = 'txup',
    cCoerceLowerCase = 'txlo',
    cCoerceRemoveDiacriticals = 'txdc',
    cCoerceRemovePunctuation = 'txpc',
    cCoerceRemoveHyphens = 'txhy',
    cCoerceOneByteToTwoByte = 'txex',
    cCoerceRemoveWhiteSpace = 'txws',
    cCoerceSmallKana = 'txsk',
    cCoerceZenkakuhankaku = 'txze',
    cCoerceKataHiragana = 'txkh',
    cZone = 'zone',
    cMachine = 'mach',
    cAddress = 'addr',
    cRunningAddress = 'radd',
    cStorage = 'stor'
};
```

cHandleBreakpoint

```
enum {
    cHandleBreakpoint = 'brak'
```

```
};
```

Component Flags

Indicate which features a scripting component supports.

```
enum {
    kOSASupportsCompiling = 0x0002,
    kOSASupportsGetSource = 0x0004,
    kOSASupportsAECoercion = 0x0008,
    kOSASupportsAESending = 0x0010,
    kOSASupportsRecording = 0x0020,
    kOSASupportsConvenience = 0x0040,
    kOSASupportsDialects = 0x0080,
    kOSASupportsEventHandling = 0x0100
};
```

Constants

`kOSASupportsCompiling`
Set if the scripting component supports the functions described in "Compiling Scripts" (page 2778).
Available in Mac OS X v10.0 and later.
Declared in `OSA.h`.

`kOSASupportsGetSource`
Set if the scripting component supports the `OSAGetSource` function.
Available in Mac OS X v10.0 and later.
Declared in `OSA.h`.

`kOSASupportsAECoercion`
Set if the scripting component supports the `OSACoerceFromDesc` and `OSACoerceToDesc` functions.
Available in Mac OS X v10.0 and later.
Declared in `OSA.h`.

`kOSASupportsAESending`
Set if the scripting component supports the functions described in "Manipulating the Create and Send Functions" (page 2779).
Available in Mac OS X v10.0 and later.
Declared in `OSA.h`.

`kOSASupportsRecording`
Set if the scripting component supports the `OSAStartRecording` (page 2848)and `OSAStopRecording` (page 2849) functions.
Available in Mac OS X v10.0 and later.
Declared in `OSA.h`.

`kOSASupportsConvenience`
Set if the script component supports the `OSALoadExecute` (page 2833), `OSACompileExecute` (page 2800), and `OSADoScript` (page 2813) functions.
Available in Mac OS X v10.0 and later.
Declared in `OSA.h`.

`kOSASupportsDialects`
: Set if the scripting component supports the `OSASetCurrentDialect` (page 2842), `OSAGetCurrentDialect` (page 2821), `OSAAvailableDialectCodeList` (page 2796), `OSAGetDialectInfo` (page 2822), and `OSAAvailableDialects` (page 2797) functions.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSASupportsEventHandling`
: Set if the scripting component supports the event handling functions described in "Using Script Contexts to Handle Apple Events" (page 2781).

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

Discussion
Your application can use the Component Manager to find a scripting component that supports a specific group of functions or to determine whether a particular scripting component supports a specific group of functions. Each of these flags identifies one of these groups of functions. To specify one or more groups of functions for the Component Manager, use these constants to set the equivalent bits in the `componentFlags` field of a component description record.

Declared In
`OSA.h`

Considerations Flags

```
enum {
  kAECase                       = 'case',
  kAEDiacritic                  = 'diac',
  kAEWhiteSpace                 = 'whit',
  kAEHyphens                    = 'hyph',
  kAEExpansion                  = 'expa',
  kAEPunctuation                = 'punc',
  kAEZenkakuHankaku             = 'zkhk',
  kAESmallKana                  = 'skna',
  kAEKataHiragana               = 'hika',
  kASConsiderReplies            = 'rmte',
  kASNumericStrings             = 'nume',
  enumConsiderations            = 'cons'
};
```

Constants
`kASNumericStrings`
: Should strings be considered as numbers?

Available in Mac OS X v10.4 and later.

Declared in `ASRegistry.h`.

Version Notes
The constant `kASNumericStrings` is available starting with Mac OS X version 10.4.

Declared In
`ASRegistry.h`

Considerations Bit Masks

Specify settings for string comparisons.

```
enum {
  kAECaseConsiderMask           = 0x00000001,
  kAEDiacriticConsiderMask      = 0x00000002,
  kAEWhiteSpaceConsiderMask     = 0x00000004,
  kAEHyphensConsiderMask        = 0x00000008,
  kAEExpansionConsiderMask      = 0x00000010,
  kAEPunctuationConsiderMask    = 0x00000020,
  kASConsiderRepliesConsiderMask = 0x00000040,
  kASNumericStringsConsiderMask = 0x00000080,
  kAECaseIgnoreMask             = 0x00010000,
  kAEDiacriticIgnoreMask        = 0x00020000,
  kAEWhiteSpaceIgnoreMask       = 0x00040000,
  kAEHyphensIgnoreMask          = 0x00080000,
  kAEExpansionIgnoreMask        = 0x00100000,
  kAEPunctuationIgnoreMask      = 0x00200000,
  kASConsiderRepliesIgnoreMask  = 0x00400000,
  kASNumericStringsIgnoreMask   = 0x00800000,
  enumConsidsAndIgnores         = 'csig'
};
```

Constants

`kASNumericStringsConsiderMask`

If bit at this position is set, consider strings to represent numerical values for comparison. For example, compare the string "1.01" as if it were the number 1.01.

Available in Mac OS X v10.4 and later.

Declared in `ASRegistry.h`.

`kASNumericStringsIgnoreMask`

If bit at this position is set, do not compare strings as numeric values.

Available in Mac OS X v10.4 and later.

Declared in `ASRegistry.h`.

Discussion

AppleScript has various settings for string comparisons, such as whether to consider or ignore capitalization. When your application receives an Apple event from AppleScript, it contains an attribute with the keyword `enumConsidsAndIgnores`. You can extract the consideration bit information from that attribute as `typeSInt32`, then use the bit masks in this enum to determine which considering and ignoring flags are currently set. You can use that information to conduct comparisons with the same criteria currently in use by AppleScript.

Version Notes

The constants `kASNumericStringsConsiderMask` and `kASNumericStringsIgnoreMask` are available starting with Mac OS X version 10.4.

Declared In

`ASRegistry.h`

cString

```
enum {
    cString = cStringClass
};
```

Current Dialect Constants

```
enum {
    kOSASelectSetCurrentDialect = 0x0701,
    kOSASelectGetCurrentDialect = 0x0702,
    kOSASelectAvailableDialects = 0x0703,
    kOSASelectGetDialectInfo = 0x0704,
    kOSASelectAvailableDialectCodeList = 0x0705
};
```

Discussion
AppleScript is designed so that scripts can be displayed in different dialects, which are representations of AppleScript that resemble human languages or programming languages. While dialects are supported, they are not particularly useful because no currently available OSA language supports dialects other than English.

Date and Time Constants

```
enum {
    pASWeekday = 'wkdy',
    pASMonth = 'mnth',
    pASDay = 'day ',
    pASYear = 'year',
    pASTime = 'time',
    pASDateString = 'dstr',
    pASTimeString = 'tstr',
    cMonth = pASMonth,
    cJanuary = 'jan ',
    cFebruary = 'feb ',
    cMarch = 'mar ',
    cApril = 'apr ',
    cMay = 'may ',
    cJune = 'jun ',
    cJuly = 'jul ',
    cAugust = 'aug ',
    cSeptember = 'sep ',
    cOctober = 'oct ',
    cNovember = 'nov ',
    cDecember = 'dec '
};
```

Default Initialization Values

Initialization constants passed to `ASInit` function.

```
enum {
    kASDefaultMinStackSize = 4 * 1024,
    kASDefaultPreferredStackSize = 16 * 1024,
    kASDefaultMaxStackSize = 16 * 1024,
    kASDefaultMinHeapSize = 4 * 1024,
    kASDefaultPreferredHeapSize = 16 * 1024,
    kASDefaultMaxHeapSize = 32L * 1024 * 1024
};
```

Constants

`kASDefaultMinStackSize`

Represents the default value for the minimum size for the portion of the application's heap used by the AppleScript component's application-specific stack.

Available in Mac OS X v10.0 and later.

Declared in `AppleScript.h`.

`kASDefaultPreferredStackSize`

Represents the default value for the preferred size for the portion of the application's heap used by the AppleScript component's application-specific stack.

Available in Mac OS X v10.0 and later.

Declared in `AppleScript.h`.

`kASDefaultMaxStackSize`

Represents the default value for the maximum size for the portion of the application's heap used by the AppleScript component's application-specific stack.

Available in Mac OS X v10.0 and later.

Declared in `AppleScript.h`.

`kASDefaultMinHeapSize`

Represents the default value for the minimum size for the portion of the application's heap used by the AppleScript component's application-specific heap. (See Version Notes section.)

Available in Mac OS X v10.0 and later.

Declared in `AppleScript.h`.

`kASDefaultPreferredHeapSize`

Represents the default value for the preferred size for the portion of the application's heap used by the AppleScript component's application-specific heap. (See Version Notes section.)

Available in Mac OS X v10.0 and later.

Declared in `AppleScript.h`.

`kASDefaultMaxHeapSize`

Represents the default value for the maximum size for the portion of the application's heap used by the AppleScript component's application-specific heap. (See Version Notes section.)

Available in Mac OS X v10.0 and later.

Declared in `AppleScript.h`.

Discussion

You can pass these constants to the `ASInit` (page 2788) function to use the default values when initializing the AppleScript component. These values are also used if `ASInit` is not called explicitly, or if any of `ASInit`'s parameters are zero.

Version Notes

Starting in Mac OS X version 10.5, heap size parameter values are ignored—AppleScript's heap will grow as large as needed.

Declared In
AppleScript.h

Dialect Descriptor Constants

Define the descriptor type and keywords for descriptor records describing the dialects supported by a scripting component.

```
enum {
    typeOSADialectInfo = 'difo',
    keyOSADialectName = 'dnam',
    keyOSADialectCode = 'dcod',
    keyOSADialectLangCode = 'dlcd',
    keyOSADialectScriptCode = 'dscd'
};
```

Constants

`typeOSADialectInfo`
The descriptor type for each item in list returned by `OSAAvailableDialects`.
Available in Mac OS X v10.0 and later.
Declared in `OSA.h`.

`keyOSADialectName`
Used with a descriptor record of any text type, such as type `typeChar`.
Available in Mac OS X v10.0 and later.
Declared in `OSA.h`.

`keyOSADialectCode`
Used with a descriptor record of type `typeShortInteger`.
Available in Mac OS X v10.0 and later.
Declared in `OSA.h`.

`keyOSADialectLangCode`
Used with a descriptor record of type `typeShortInteger`.
Available in Mac OS X v10.0 and later.
Declared in `OSA.h`.

`keyOSADialectScriptCode`
Used with a descriptor record of type `typeShortInteger`.
Available in Mac OS X v10.0 and later.
Declared in `OSA.h`.

Discussion
These constants define the descriptor type for each item in the list returned by `OSAAvailableDialects` and the keywords for descriptor records of that type. The keyword constants can also be used in the *`selector`* parameter of `OSAGetDialectInfo` to obtain information about the dialects supported by a scripting component.

Generic Scripting Component Selectors

```
enum {
    kGSSSelectGetDefaultScriptingComponent = 0x1001,
    kGSSSelectSetDefaultScriptingComponent = 0x1002,
    kGSSSelectGetScriptingComponent = 0x1003,
    kGSSSelectGetScriptingComponentFromStored = 0x1004,
    kGSSSelectGenericToRealID = 0x1005,
    kGSSSelectRealToGenericID = 0x1006,
    kGSSSelectOutOfRange = 0x1007
};
```

Global Properties

```
enum {
    pASIt = 'it  ',
    pASMe = 'me  ',
    pASResult = 'rslt',
    pASSpace = 'spac',
    pASReturn = 'ret ',
    pASTab = 'tab ',
    pASPi = 'pi  ',
    pASParent = 'pare',
    kASInitializeEventCode = 'init',
    pASPrintLength = 'prln',
    pASPrintDepth = 'prdp',
    pASTopLevelScript = 'ascr'
};
```

kASAdd

```
enum {
    kASAdd = '+   ',
    kASSubtract = '-   ',
    kASMultiply = '*   ',
    kASDivide = '/   ',
    kASQuotient = 'div ',
    kASRemainder = 'mod ',
    kASPower = '^   ',
    kASEqual = kAEEquals,
    kASNotEqual = '   ',
    kASGreaterThan = kAEGreaterThan,
    kASGreaterThanOrEqual = kAEGreaterThanEquals,
    kASLessThan = kAELessThan,
    kASLessThanOrEqual = kAELessThanEquals,
    kASComesBefore = 'cbfr',
    kASComesAfter = 'cafr',
    kASConcatenate = 'ccat',
    kASStartsWith = kAEBeginsWith,
    kASEndsWith = kAEEndsWith,
    kASContains = kAEContains
};
```

kASAnd

```
enum {
    kASAnd = kAEAND,
    kASOr = kAEOR,
    kASNot = kAENOT,
    kASNegate = 'neg ',
    keyASArg = 'arg '
};
```

kASErrorEventCode

```
enum {
    kASErrorEventCode = 'err ',
    kOSAErrorArgs = 'erra',
    keyAEErrorObject = 'erob',
    pLength = 'leng',
    pReverse = 'rvse',
    pRest = 'rest',
    pInherits = 'c@#^',
    pProperties = 'pALL',
    keyASUserRecordFields = 'usrf',
    typeUserRecordFields = typeAEList
};
```

kASStartLogEvent

```
enum {
    kASStartLogEvent = 'log1',
    kASStopLogEvent = 'log0',
    kASCommentEvent = 'cmnt'
};
```

kDialectBundleResType

```
enum {
    kDialectBundleResType = 'Dbdl',
    cConstant = typeEnumerated,
    cClassIdentifier = pClass,
    cObjectBeingExamined = typeObjectBeingExamined,
    cList = typeAEList,
    cSmallReal = typeSMFloat,
    cReal = typeFloat,
    cRecord = typeAERecord,
    cReference = cObjectSpecifier,
    cUndefined = 'undf',
    cMissingValue = 'msng',
    cSymbol = 'symb',
    cLinkedList = 'llst',
    cVector = 'vect',
    cEventIdentifier = 'evnt',
    cKeyIdentifier = 'kyid',
    cUserIdentifier = 'uid ',
```

```
    cPreposition = 'prep',
    cKeyForm = enumKeyForm,
    cScript = 'scpt',
    cHandler = 'hand',
    cProcedure = 'proc'
};
```

keyAETarget

```
enum {
    keyAETarget = 'targ',
    keySubjectAttr = 'subj',
    keyASReturning = 'Krtn',
    kASAppleScriptSuite = 'ascr',
    kASScriptEditorSuite = 'ToyS',
    kASTypeNamesSuite = 'tpnm',
    typeAETE = 'aete',
    typeAEUT = 'aeut',
    kGetAETE = 'gdte',
    kGetAEUT = 'gdut',
    kUpdateAEUT = 'udut',
    kUpdateAETE = 'udte',
    kCleanUpAEUT = 'cdut',
    kASComment = 'cmnt',
    kASLaunchEvent = 'noop',
    keyScszResource = 'scsz',
    typeScszResource = 'scsz',
    kASSubroutineEvent = 'psbr',
    keyASSubroutineName = 'snam',
    kASPrepositionalSubroutine = 'psbr',
    keyASPositionalArgs = 'parg'
};
```

keyAppHandledCoercion

```
enum {
    keyAppHandledCoercion = 'idas'
};
```

keyASPrepositionAt

```
enum {
    keyASPrepositionAt = 'at  ',
    keyASPrepositionIn = 'in  ',
    keyASPrepositionFrom = 'from',
    keyASPrepositionFor = 'for ',
    keyASPrepositionTo = 'to  ',
    keyASPrepositionThru = 'thru',
    keyASPrepositionThrough = 'thgh',
    keyASPrepositionBy = 'by  ',
    keyASPrepositionOn = 'on  ',
    keyASPrepositionInto = 'into',
    keyASPrepositionOnto = 'onto',
```

```
    keyASPrepositionBetween = 'btwn',
    keyASPrepositionAgainst = 'agst',
    keyASPrepositionOutOf = 'outo',
    keyASPrepositionInsteadOf = 'isto',
    keyASPrepositionAsideFrom = 'asdf',
    keyASPrepositionAround = 'arnd',
    keyASPrepositionBeside = 'bsid',
    keyASPrepositionBeneath = 'bnth',
    keyASPrepositionUnder = 'undr'
};
```

keyASPrepositionOver

```
enum {
    keyASPrepositionOver = 'over',
    keyASPrepositionAbove = 'abve',
    keyASPrepositionBelow = 'belw',
    keyASPrepositionApartFrom = 'aprt',
    keyASPrepositionGiven = 'givn',
    keyASPrepositionWith = 'with',
    keyASPrepositionWithout = 'wout',
    keyASPrepositionAbout = 'abou',
    keyASPrepositionSince = 'snce',
    keyASPrepositionUntil = 'till'
};
```

keyOSASourceEnd

Specifies the end of an error range.

```
enum {
    keyOSASourceEnd = 'srce'
};
```

Constants

`keyOSASourceEnd`

Field of a `typeOSAErrorRange` record of `typeShortInteger`. This field specifies the end of the error range.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

Declared In

`OSA.h`

keyOSASourceStart

Specifies the start of an error range.

```
enum {
    keyOSASourceStart = 'srcs'
};
```

Constants

`keyOSASourceStart`

Field of a `typeOSAErrorRange` record of `typeShortInteger`. This field specifies the start of the error range.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

Declared In

`OSA.h`

keyProcedureName

```
enum {
    keyProcedureName = 'dfnm',
    keyStatementRange = 'dfsr',
    keyLocalsNames = 'dfln',
    keyGlobalsNames = 'dfgn',
    keyParamsNames = 'dfpn'
};
```

keyProgramState

```
enum {
    keyProgramState = 'dsps'
};
```

kGenericComponentVersion

Specifies the generic component version.

```
enum {
    kGenericComponentVersion = 0x0100
};
```

Constants

`kGenericComponentVersion`

Indicates the component version this header file describes.

Available in Mac OS X v10.0 and later.

Declared in `OSAGeneric.h`.

Declared In

`OSAGeneric.h`

kOSAComponentType

Defines the Component Manager type code for components that support the standard scripting component routines.

```
enum {
    kOSAComponentType = 'osa '
};
```

Constants

`kOSAComponentType`

Specifies the standard OSA component type.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

Declared In

`OSA.h`

kOSAGenericScriptingComponentSubtype

Defines the subtype code for the generic scripting component.

```
enum {
    kOSAGenericScriptingComponentSubtype = 'scpt'
};
```

kOSAModeDontDefine

```
enum {
    kOSAModeDontDefine = 0x0001
};
```

Constants

`kOSAModeDontDefine`

This mode flag can be passed to `OSASetProperty` (page 2844) or `OSASetHandler` (page 2843) and will prevent properties or handlers from being defined in a context that doesn't already have bindings for them. An error is returned if a current binding doesn't already exist.

Available in Mac OS X v10.0 and later.

Declared in `ASDebugging.h`.

kOSANullScript

Defines a null script ID.

```
enum {
    kOSANullScript = 0
};
```

Discussion
If the execution of a script does not result in a value, `OSAExecute` returns the constant `kOSANullScript` as the script ID. If a client application passes `kOSANullScript` to the `OSAGetSource` function instead of a valid script ID, the scripting component should display a null source description (possibly an empty text string). If a client application passes `kOSANullScript` to `OSAStartRecording`, the scripting component creates a new compiled script for editing or recording.

kOSARecordedText

Defines the event code for the Recorded Text event.

```
enum {
    kOSARecordedText = 'recd'
};
```

kOSAScriptResourceType

Defines the resource type for stored script data.

```
enum {
    kOSAScriptResourceType = kOSAGenericScriptingComponentSubtype
};
```

Constants
`kOSAScriptResourceType`
Resource type for scripts.
Available in Mac OS X v10.0 and later.
Declared in `OSA.h`.

kOSASelectComponentSpecificStart

```
enum {
    kOSASelectComponentSpecificStart = 0x1001
};
```

Constants
`kOSASelectComponentSpecificStart`
Scripting component specific selectors are added beginning with this value.
Available in Mac OS X v10.0 and later.
Declared in `OSA.h`.

kOSASelectCopyScript

```
enum {
    kOSASelectCopyScript = 0x0105
};
```

kOSASuite

Defines the suite for the Recorded Text event.

```
enum {
    kOSASuite = 'ascr'
};
```

Mode Flags

Specify information used by the scripting component.

```
enum {
    kOSAModePreventGetSource = 0x00000001
};
enum {
    kOSAModeNeverInteract = kAENeverInteract,
    kOSAModeCanInteract = kAECanInteract,
    kOSAModeAlwaysInteract = kAEAlwaysInteract,
    kOSAModeDontReconnect = kAEDontReconnect
};
enum {
    kOSAModeCantSwitchLayer = 0x00000040
};
enum {
    kOSAModeDoRecord = 0x00001000
};
enum {
    kOSAModeCompileIntoContext = 0x00000002
};
enum {
    kOSAModeAugmentContext = 0x00000004
};
enum {
    kOSAModeDisplayForHumans = 0x00000008
};
enum {
    kOSAModeDontStoreParent = 0x00010000
};
enum {
    kOSAModeDispatchToDirectObject = 0x00020000
};
enum {
    kOSAModeDontGetDataForArguments = 0x00040000
};
enum {
kOSAModeFullyQualifyDescriptors = 0x00080000
};
```

Constants

`kOSAModePreventGetSource`

This mode flag may be passed to `OSALoad` (page 2832), `OSAStore` (page 2850), or `OSACompile` (page 2799) to instruct the scripting component to not retain the "source" of an expression. This will cause a call to `OSAGetSource` (page 2830) to return the error `errOSASourceNotAvailable` if used. However, some scripting components may not retain the source anyway. This is mainly used when either space efficiency is desired, or a script is to be "locked" so that its implementation may not be viewed.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSAModeNeverInteract`

This mode flag may be passed to the functions `OSACompile` (page 2799), `OSAExecute` (page 2816), `OSALoadExecute` (page 2833), `OSACompileExecute` (page 2800), `OSADoScriptFile` (page 2815), `OSAExecuteEvent` (page 2817), and `OSADoEvent` (page 2812) to indicate whether or not the script may interact with the user if necessary. Adds `kAENeverInteract` to the `sendMode` parameter of `AESend` for events sent when the script is executed.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSAModeCanInteract`

This mode flag may be passed to the functions `OSACompile` (page 2799), `OSAExecute` (page 2816), `OSALoadExecute` (page 2833), `OSACompileExecute` (page 2800), `OSADoScriptFile` (page 2815), `OSAExecuteEvent` (page 2817), and `OSADoEvent` (page 2812) to indicate whether or not the script may interact with the user. Adds `kAECanInteract` to the `sendMode` parameter of `AESend` for events sent when the script is executed.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSAModeAlwaysInteract`

This mode flag may be passed to the functions `OSACompile` (page 2799), `OSAExecute` (page 2816), `OSALoadExecute` (page 2833), `OSACompileExecute` (page 2800), `OSADoScriptFile` (page 2815), `OSAExecuteEvent` (page 2817), and `OSADoEvent` (page 2812) to indicate whether or not the script may interact with the user. Adds `kAEAlwaysInteract` to the `sendMode` parameter of `AESend` for events sent when the script is executed.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSAModeDontReconnect`

This mode flag may be passed to the functions `OSACompile` (page 2799), `OSAExecute` (page 2816), `OSALoadExecute` (page 2833), `OSACompileExecute` (page 2800), `OSADoScriptFile` (page 2815), `OSAExecuteEvent` (page 2817), and `OSADoEvent` (page 2812) to indicate whether or not the script may reconnect if necessary. Adds `kAEDontReconnect` to the `sendMode` parameter of `AESend` for events sent when the script is executed.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSAModeCantSwitchLayer`

This mode flag may be passed to the functions `OSACompile` (page 2799), `OSAExecute` (page 2816), `OSALoadExecute` (page 2833), `OSACompileExecute` (page 2800), `OSADoScriptFile` (page 2815), `OSAExecuteEvent` (page 2817), and `OSADoEvent` (page 2812) to indicate whether Apple events should be sent with the `kAECanSwitchLayer` mode flag sent. This flag is exactly the opposite of the Apple event flag `kAECanSwitchLayer`. This is to provide a more convenient default, such as not supplying any mode (see `kOSANullMode` in the "Null Mode Flags" (page 2877)) means to send events with `kAECanSwitchLayer`. Supplying the `kOSAModeCantSwitchLayer` mode flag will cause `AESend` to be called without `kAECanSwitchLayer`.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSAModeDoRecord`

This mode flag may be passed to the functions `OSACompile` (page 2799), `OSAExecute` (page 2816), `OSALoadExecute` (page 2833), `OSACompileExecute` (page 2800), `OSADoScriptFile` (page 2815), `OSAExecuteEvent` (page 2817), and `OSADoEvent` (page 2812) to indicate whether Apple events should be sent with the `kAEDontRecord` mode flag. This flag is exactly the opposite the Apple event flag `kAEDontRecord`. This is to provide a more convenient default, such as not supplying any mode (see `kOSANullMode` in the "Null Mode Flags" (page 2877)) means to send events with `kAEDontRecord`. Supplying the kOSAModeDoRecord mode flag will cause `AESend` to be called without `kAEDontRecord`.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSAModeCompileIntoContext`

This is a mode flag for `OSACompile` (page 2799) that indicates that a context should be created as the result of compilation. All handler definitions are inserted into the new context, and variables are initialized by evaluating their initial values in a null context (for example, they must be constant expressions).

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSAModeAugmentContext`

This is a mode flag for `OSACompile` (page 2799) that indicates that the previous script ID (input to `OSACompile`) should be augmented with any new definitions in the `sourceData` parameter rather than replaced with a new script. This means that the previous script ID must designate a context. The presence of this flag causes the kOSAModeCompileIntoContext flag to be implicitly used, causing any new definitions to be initialized in a null context.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSAModeDisplayForHumans`

This mode flag may be passed to `OSADisplay` (page 2810) or `OSADoScriptFile` (page 2815) to indicate that output only need be human-readable, not re-compilable by `OSACompile` (page 2799). If used, output may be arbitrarily "beautified", for example, quotes may be left off of string values, and long lists may have ellipses.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSAModeDontStoreParent`

This mode flag may be passed to `OSAStore` (page 2850) in the case where the `scriptID` parameter is a context. This causes the context to be saved, but not the context's parent context. When the stored context is loaded back in, the parent will be `kOSANullMode` (see the "Null Mode Flags" (page 2877)).

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSAModeDispatchToDirectObject`

This mode flag may be passed to `OSAExecuteEvent` (page 2817) to cause the event to be dispatched to the direct object of the event. The direct object (or subject attribute if the direct object is a non-object specifier) will be resolved, and the resulting script object will be the recipient of the message. The context argument to `OSAExecuteEvent` will serve as the root of the lookup/resolution process.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSAModeDontGetDataForArguments`

This mode flag may be passed to `OSAExecuteEvent` (page 2817) to indicate that components do not have to get the data of object specifier arguments.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSAModeFullyQualifyDescriptors`

This mode flag may be passed to `OSACoerceToDesc` (page 2798) to indicate that the resulting descriptor should be fully qualified (i.e. should include the root application reference).

Available in Mac OS X v10.3 and later.

Declared in `OSA.h`.

Null Mode Flags

Indicate a function's default mode settings are to be used.

```
enum {
    kOSANullMode = 0,
    kOSAModeNull = 0
};
```

OSADebugStepKind

```
typedef UInt32 OSADebugStepKind;
enum {
    eStepOver = 0,
    eStepIn = 1,
    eStepOut = 2,
    eRun = 3
};
```

OSAProgramState

```
typedef UInt32 OSAProgramState;
enum {
    eNotStarted = 0,
    eRunnable = 1,
    eRunning = 2,
    eStopped = 3,
    eTerminated = 4
};
```

OSAScriptError Selectors

Define selectors used to retrieve information about script errors from the `OSAScriptError` function.

```
enum {
    kOSAErrorNumber = keyErrorNumber
};
enum {
    kOSAErrorMessage = keyErrorString
};
enum {
    kOSAErrorBriefMessage = 'errb'
};
enum {
    kOSAErrorApp = 'erap'
};
enum {
    kOSAErrorPartialResult = 'ptlr'
};
enum {
    kOSAErrorOffendingObject = 'erob'
};
enum {
    kOSAErrorExpectedType = 'errt'
};
enum {
    kOSAErrorRange = 'erng'
};
```

Constants

`kOSAErrorNumber`

This selector is used to determine the error number of a script error. These error numbers may be either system error numbers, or error numbers that are scripting component specific. The value of `desiredType` must be `typeShortInteger`.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSAErrorMessage`

This selector is used to determine the full error message associated with the error number. It should include the name of the application which caused the error, as well as the specific error that occurred. This selector is sufficient for simple error reporting (but see kOSAErrorBriefMessage). The value of `desiredType` must be `typeChar` or another text descriptor type.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSAErrorBriefMessage`

This selector is used to determine a brief error message associated with the error number. This message should not mention the name of the application which caused the error, any partial results or offending object (see kOSAErrorApp, kOSAErrorPartialResult , and kOSAErrorOffendingObject). The value of `desiredType` must be `typeChar` or another text descriptor type.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSAErrorApp`

This selector is used to determine which application actually got the error (if it was the result of an `AESend`). The value of `desiredType` must be `typeProcessSerialNumber` (for the PSN) or a text descriptor type such as `typeChar` (for the name).

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSAErrorPartialResult`
: This selector is used to determine any partial result returned by an operation. If an `AESend` call failed, but a partial result was returned, then the partial result may be returned as an `AEDesc`. The value of `desiredType` must be `typeBest` (for the best type) or `typeWildCard` (for the default type).

  Available in Mac OS X v10.0 and later.

  Declared in `OSA.h`.

`kOSAErrorOffendingObject`
: This selector is used to determine any object which caused the error that may have been indicated by an application. The result is an `AEDesc`. The value of `desiredType` must be `typeObjectSpecifier`, `typeBest`, or `typeWildCard`. For some scripting components, including AppleScript, these three values are equivalent.

  Available in Mac OS X v10.0 and later.

  Declared in `OSA.h`.

`kOSAErrorRange`
: This selector is used to determine the source text range (start and end positions) of where the error occurred. The value of `desiredType` must be `typeOSAErrorRange`.

  Available in Mac OS X v10.0 and later.

  Declared in `OSA.h`.

`kOSAErrorExpectedType`
: This selector is used to determine the type expected by a coercion operation if a type error occurred.

  Available in Mac OS X v10.0 and later.

  Declared in `OSA.h`.

Recording Constants

```
enum {
    kOSASelectStartRecording = 0x0501,
    kOSASelectStopRecording = 0x0502
};
```

Resume Dispatch Function Constants

Define constants used with the `OSASetResumeDispatchProc` function.

```
enum {
    kOSAUseStandardDispatch = kAEUseStandardDispatch
};
enum {
    kOSANoDispatch = kAENoDispatch
};
enum {
    kOSADontUsePhac = 0x0001
};
```

Constants

`kOSAUseStandardDispatch`
Used in the `resumeDispatchProc` parameter of `OSASetResumeDispatchProc` (page 2845) and `OSAGetResumeDispatchProc` (page 2826) to indicate that the event is dispatched using standard Apple event dispatching (the handler registered in the application with `AEInstallEventHandler` should be used).

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSANoDispatch`
Used in the `resumeDispatchProc` parameter of `OSASetResumeDispatchProc` (page 2845) to tell the Apple Event Manager that the processing of the Apple event is complete and that no dispatching should occur.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSADontUsePhac`
Used in the `refCon` parameter of `OSASetResumeDispatchProc` (page 2845) to dispatch the event using standard Apple event dispatching, except that the predispatch handler should not be called. Used only in conjunction with `kOSAUseStandardDispatch`. This is useful when the predispatch handler is used to lookup a context associated with an event's direct parameter and call `OSAExecuteEvent` (page 2817) or `OSADoEvent` (page 2812). Failure to bypass the predispatch handler when resuming an event in this case would result in an infinite loop. (A predispatch handler is called immediately before the Apple Event Manager dispatches an event.)

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

Declared In
`OSA.h`

Script Document File Type

Defines the file type of script document files.

```
enum {
    kOSAFileType = 'osas'
};
```

Script Information Selectors

Specify which script information is set or returned.

```
enum {
    kOSAScriptIsModified = 'modi'
};
enum {
    kOSAScriptIsTypeCompiledScript = 'cscr'
};
enum {
    kOSAScriptIsTypeScriptValue = 'valu'
};
enum {
    kOSAScriptIsTypeScriptContext = 'cntx'
};
enum {
    kOSAScriptBestType = 'best'
};
enum {
    kOSACanGetSource = 'gsrc'
};
enum {
    kASHasOpenHandler = 'hsod'
};
```

Constants

`kOSAScriptIsModified`
This selector is used to determine whether there have been any changes since the script data was loaded or created. In Mac OS X, the AppleScript component returns a value of `false` if no changes have been made, and a value of `true` if changes may have been made. For more information, see the Version Notes section for the `OSAGetScriptInfo` (page 2827) function.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSAScriptIsTypeCompiledScript`
This selector is used to determine whether or not the script data is a compiled script. The selector returns a boolean.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSAScriptIsTypeScriptValue`
This selector is used to determine whether or not the script data is a script value. The selector returns a boolean.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSAScriptIsTypeScriptContext`
This selector is used to determine whether or not the script data is a script context. The selector returns a boolean.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSAScriptBestType`
A descriptor type that you can pass to `OSACoerceToDesc`.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSACanGetSource`
: This selector is used to determine whether a script has source associated with it that when given to `OSAGetSource`, the call will not fail. The selector returns a boolean.

  Available in Mac OS X v10.0 and later.

  Declared in `OSA.h`.

`kASHasOpenHandler`
: This selector is used to query a context as to whether it contains a handler for the `kAEOpenDocuments` event. This allows "applets" to be distinguished from "droplets." `OSAGetScriptInfo` (page 2827) returns false if there is no `kAEOpenDocuments` handler, and returns the error value `errOSAInvalidAccess` if the input is not a context.

  Available in Mac OS X v10.0 and later.

  Declared in `AppleScript.h`.

Source Constants

```
enum {
    kOSASelectGetSource = 0x0201
};
```

Source Style Constants

Identify script format styles used by the AppleScript component to display scripts.

```
enum {
    kASSourceStyleUncompiledText = 0,
    kASSourceStyleNormalText = 1,
    kASSourceStyleLanguageKeyword = 2,
    kASSourceStyleApplicationKeyword = 3,
    kASSourceStyleComment = 4,
    kASSourceStyleLiteral = 5,
    kASSourceStyleUserSymbol = 6,
    kASSourceStyleObjectSpecifier = 7,
    kASNumberOfSourceStyles = 8
};
```

Constants

`kASSourceStyleUncompiledText`
: Script format style for uncompiled text.

  Available in Mac OS X v10.0 and later.

  Declared in `AppleScript.h`.

`kASSourceStyleNormalText`
: Script format style for normal text.

  Available in Mac OS X v10.0 and later.

  Declared in `AppleScript.h`.

`kASSourceStyleLanguageKeyword`
: Script format style for keywords of the AppleScript Language.

  Available in Mac OS X v10.0 and later.

  Declared in `AppleScript.h`.

`kASSourceStyleApplicationKeyword`
Script format style for keywords of a scriptable application.

Available in Mac OS X v10.0 and later.

Declared in `AppleScript.h`.

`kASSourceStyleComment`
Script format style for comment text.

Available in Mac OS X v10.0 and later.

Declared in `AppleScript.h`.

`kASSourceStyleLiteral`
Script format style for literal text.

Available in Mac OS X v10.0 and later.

Declared in `AppleScript.h`.

`kASSourceStyleUserSymbol`
A user-defined symbol, such as a variable or custom handler name.

Available in Mac OS X v10.0 and later.

Declared in `AppleScript.h`.

`kASSourceStyleObjectSpecifier`
Deprecated.

Available in Mac OS X v10.0 and later.

Declared in `AppleScript.h`.

`kASNumberOfSourceStyles`
Deprecated. (The number of different format styles available.)

See the Discussion section for why you should not use this constant.

Available in Mac OS X v10.0 and later.

Declared in `AppleScript.h`.

Discussion
These constants are used to access specific styles in the style information used by the `ASCopySourceAttributes` (page 2785), `ASSetSourceAttributes` (page 2791), and `ASGetSourceStyleNames` (page 2787) functions (and the deprecated functions `ASGetSourceStyles` (page 2788) and `ASSetSourceStyles` (page 2792)).

The order of the style information corresponds to the order of the constants. For example, the first dictionary in the array returned by `ASCopySourceAttributes` (page 2785) (at position `kASSourceStyleUncompiledText`) describes the style for uncompiled text. However, you should not rely on there being any specific number of dictionaries (such as `kASNumberOfSourceStyles`) in the returned array—instead, count the number of items in the array before accessing any of them.

Declared In
`AppleScript.h`

typeAppleScript

Define descriptor types for the AppleScript instance of the Open Scripting Architecture type.

```
enum {
    typeAppleScript = 'ascr',
    kAppleScriptSubtype = typeAppleScript,
    typeASStorage = typeAppleScript
};
```

Constants

`kAppleScriptSubtype`

Defines the Component Manager subtype for the AppleScript component.

Available in Mac OS X v10.0 and later.

Declared in `AppleScript.h`.

`typeASStorage`

Defines the AppleScript constant for storage descriptor records.

Available in Mac OS X v10.0 and later.

Declared in `AppleScript.h`.

typeOSAErrorRange

Defines the descriptor type for an error range.

```
enum {
    typeOSAErrorRange = 'erng'
};
```

typeOSAGenericStorage

Defines the descriptor type for generic storage descriptor records.

```
enum {
    typeOSAGenericStorage = kOSAScriptResourceType
};
```

Constants

`typeOSAGenericStorage`

Default type given to `OSAStore` (page 2850), which creates "generic" loadable script data descriptors.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

Declared In

`OSA.h`

typeStatementRange

```
enum {
    typeStatementRange = 'srng'
};
```

Weekdays

```
enum {
    cWeekday = pASWeekday,
    cSunday = 'sun ',
    cMonday = 'mon ',
    cTuesday = 'tue ',
    cWednesday = 'wed ',
    cThursday = 'thu ',
    cFriday = 'fri ',
    cSaturday = 'sat ',
    pASQuote = 'quot',
    pASSeconds = 'secs',
    pASMinutes = 'min ',
    pASHours = 'hour',
    pASDays = 'days',
    pASWeeks = 'week',
    cWritingCodeInfo = 'citl',
    pScriptCode = 'pscd',
    pLangCode = 'plcd',
    kASMagicTellEvent = 'tell',
    kASMagicEndTellEvent = 'tend'
};
```

Result Codes

The most common result codes returned by Open Scripting Architecture are listed in Table 1-1. Open Scripting Architecture may also return the result codes `noErr` (0), and `badComponentInstance` (-2147450879).

Result Code	Value	Description
`errOSACantCoerce`	-1700	A value can't be coerced to the desired type. Available in Mac OS X v10.0 and later.
`OSAMissingParameter`	-1701	A parameter is missing for a function invocation. Available in Mac OS X v10.0 and later.
`errOSACorruptData`	-1702	Some data could not be read. Available in Mac OS X v10.0 and later.
`errOSATypeError`	-1703	Same as `errAEWrongDataType`; wrong descriptor type. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
`OSAMessageNotUnderstood`	-1708	A message was sent to an object that didn't handle it. Available in Mac OS X v10.0 and later.
`OSAUndefinedHandler`	-1717	A function to be returned doesn't exist. Available in Mac OS X v10.0 and later.
`OSAIllegalIndex`	-1719	An index was out of range. Specialization of `errOSACantAccess`. Available in Mac OS X v10.0 and later.
`OSAIllegalRange`	-1720	The specified range is illegal. Specialization of `errOSACantAccess`. Available in Mac OS X v10.0 and later.
`OSAParameterMismatch`	-1721	The wrong number of parameters were passed to the function, or a parameter pattern cannot be matched. Available in Mac OS X v10.0 and later.
`OSAIllegalAccess`	-1723	A container can not have the requested object. Available in Mac OS X v10.0 and later.
`errOSACantAccess`	-1728	An object is not found in a container. Available in Mac OS X v10.0 and later.
`errOSARecordingIsAlreadyOn`	-1732	Recording is already on. Available in Mac OS X v10.0 and later.
`errOSASystemError`	-1750	Scripting component error. Available in Mac OS X v10.0 and later.
`errOSAInvalidID`	-1751	Invalid script id. Available in Mac OS X v10.0 and later.
`errOSABadStorageType`	-1752	Script doesn't seem to belong to AppleScript. Available in Mac OS X v10.0 and later.
`errOSAScriptError`	-1753	Script error. Available in Mac OS X v10.0 and later.
`errOSABadSelector`	-1754	Invalid selector given. Available in Mac OS X v10.0 and later.
`errOSASourceNotAvailable`	-1756	Invalid access. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
errOSANoSuchDialect	-1757	Source not available. Available in Mac OS X v10.0 and later.
errOSADataFormatObsolete	-1758	No such dialect. Available in Mac OS X v10.0 and later.
errOSADataFormatTooNew	-1759	Data couldn't be read because its format is obsolete. Available in Mac OS X v10.0 and later.
errOSAComponentMismatch	-1761	Parameters are from two different components. Available in Mac OS X v10.0 and later.
errOSACantOpenComponent	-1762	Can't connect to system with that ID. Available in Mac OS X v10.0 and later.
errOSAGeneralError	-2700	No actual error code is to be returned. Available in Mac OS X v10.0 and later.
errOSADivideByZero	-2701	An attempt to divide by zero was made. Available in Mac OS X v10.0 and later.
errOSANumericOverflow	-2702	An integer or real value is too large to be represented. Available in Mac OS X v10.0 and later.
errOSACantLaunch	-2703	An application can't be launched, or when it is, remote and program linking is not enabled. Available in Mac OS X v10.0 and later.
errOSAAppNotHighLevelEventAware	-2704	An application can't respond to Apple events. Available in Mac OS X v10.0 and later.
errOSACorruptTerminology	-2705	An application's terminology resource is not readable. Available in Mac OS X v10.0 and later.
errOSAStackOverflow	-2706	The runtime stack overflowed. Available in Mac OS X v10.0 and later.
errOSAInternalTableOverflow	-2707	A runtime internal data structure overflowed. Available in Mac OS X v10.0 and later.
errOSADataBlockTooLarge	-2708	An intrinsic limitation is exceeded for the size of a value or data structure. Available in Mac OS X v10.0 and later.
errOSACantGetTerminology	-2709	Can't get the event dictionary. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
errOSACantCreate	-2710	Can't make class <class identifier>. Available in Mac OS X v10.0 and later.
OSASyntaxError	-2740	A syntax error occurred. Available in Mac OS X v10.0 and later.
OSASyntaxTypeError	-2741	Another form of syntax was expected. Available in Mac OS X v10.0 and later.
OSATokenTooLong	-2742	A name or number is too long to be parsed. Available in Mac OS X v10.0 and later.
OSADuplicateParameter	-2750	A formal parameter, local variable, or instance variable is specified more than once. Available in Mac OS X v10.0 and later.
OSADuplicateProperty	-2751	A formal parameter, local variable, or instance variable is specified more than once. Available in Mac OS X v10.0 and later.
OSADuplicateHandler	-2752	More than one handler is defined with the same name in a scope where the language doesn't allow it. Available in Mac OS X v10.0 and later.
OSAUndefinedVariable	-2753	A variable is accessed that has no value. Available in Mac OS X v10.0 and later.
OSAInconsistentDeclarations	-2754	A variable is declared inconsistently in the same scope, such as both local and global. Available in Mac OS X v10.0 and later.
OSAControlFlowError	-2755	An illegal control flow occurs in an application. For example, there is no catcher for the throw, or there was a non-lexical loop exit. Available in Mac OS X v10.0 and later.
OSAIllegalAssign	-10003	An object can never be set in a container Available in Mac OS X v10.0 and later.
errOSACantAssign	-10006	An object cannot be set in a container. Available in Mac OS X v10.0 and later.

Printing Plug-in Interfaces Reference

Framework:	ApplicationServices/ApplicationServices.h, Carbon/Carbon.h
Declared in	PMIOModule.h PMPluginHeader.h PMPrinterBrowsers.h PMPrinterModule.h PMPrintingDialogExtensions.h

Overview

As printer vendors and application developers extend the printing capabilities of their hardware and software products, they need a way to extend the Mac OS X printing system to make new printing features available to their customers. To address this need, Mac OS X has introduced the **printing plug-in** —a component architecture based on Core Foundation Plug-in Services. There are four types of printing plug-ins in Mac OS X:

- **I/O modules** are used by the printing system to communicate with a printer using a standard transport-layer interface, such as AppleTalk or TCP/IP.
- **Printer browsers** provide a way for people to discover available local and network printers.
- **Printer modules** are used by the printing system to convert the graphics content in a print job for output to a specific printer or family of printers.
- **Printing dialog extensions** provide a way for people to view and change the settings for a set of related printing features. The user interface of a printing dialog extension is a pane in one of the printing dialogs.

This reference document is relevant for anyone writing a plug-in that provides support for printing.

If you're writing a printing dialog extension, you should refer to this document as you implement the required callback functions. For conceptual information about printing dialog extensions, see *Extending Printing Dialogs*.

Functions

PMCreateLocalizedPaperSizeCFString

Returns a text description of the physical dimensions of the paper specified in a paper information ticket. The unit of measure is based on the current language and script system.

```
CFStringRef PMCreateLocalizedPaperSizeCFString (
   PMTicketRef listofPaperTickets,
   UInt16 paperToUse
);
```

Parameters

listofPaperTickets

A ticket that contains a list of paper information tickets. Typically you would obtain this list ticket from a printer module template ticket.

paperToUse

A one-based index that specifies an entry in a list ticket.Upon entering, this parameter should specify a valid entry in `listofPaperTickets`.

Return Value

A text description of the physical dimensions of the paper specified in a paper information ticket. Numeric values are localized to English or metric measure, based on the current language and script system. The caller assumes ownership of the string and is responsible for releasing it. The return value `NULL` indicates that the function failed to create the string.

Discussion

The name `PMCreateLocalizedPaperSizeCFString` suggests that this function does some additional localization, but that is misleading. This function takes an array of paper information tickets and an index, finds the desired paper information ticket, and simply calls `PMCreatePaperSizeCFString` to get the physical paper size.

Availability

Available in Mac OS X version 10.0 and later.

Not available to 64-bit applications.

Declared In

`PMPrintingDialogExtensionsDeprecated.h`

PMCreatePaperSizeCFString

Returns a text description of the physical dimensions of the paper specified in a paper information ticket. The unit of measure is based on the current language and script system.

```
CFStringRef PMCreatePaperSizeCFString (
   PMTicketRef selectedPaper
);
```

Parameters

selectedPaper

A paper information ticket. Typically you would obtain this ticket from a page format ticket, or possibly from a printer module template ticket.

Return Value

A text description of the physical dimensions of the paper specified in a paper information ticket. Numeric values are localized to English or metric measure, based on the current language and script system. The caller assumes ownership of the string, and is responsible for releasing it. The return value `NULL` indicates that the function failed to create the string.

Availability

Available in Mac OS X version 10.0 and later.

Not available to 64-bit applications.

Declared In
`PMPrintingDialogExtensionsDeprecated.h`

Callbacks by Task

Printing Plug-in Callbacks

The three callback functions described in this section are used by the printing system to manage a printing plug-in after it is loaded. Printing plug-ins (except for printer browsers) are required to implement these functions.

`PMPluginRetainProcPtr` (page 2916)

`PMPluginReleaseProcPtr` (page 2915)
Defines a pointer to the retain function in a printing plug-in interface. Your custom retain function increments the reference count of an instance of one of these interfaces.

`PMPluginGetAPIVersionProcPtr` (page 2915)

Printing Dialog Extension Callbacks

The callback functions described in this section must be implemented by all printing dialog extensions.

`PMPDEPrologueProcPtr` (page 2911)

`PMPDEInitializeProcPtr` (page 2909)

`PMPDESyncProcPtr` (page 2913)

`PMPDEGetSummaryTextProcPtr` (page 2908)

`PMPDEOpenProcPtr` (page 2911)

`PMPDECloseProcPtr` (page 2908)

`PMPDETerminateProcPtr` (page 2914)

Printer Module Callbacks

The callback functions described in this section must be implemented by all printer modules.

PMPrintJobProcPtr (page 2921)

PMPrintPageProcPtr (page 2921)

PMTerminateProcPtr (page 2922)

Printer Browser Module Callbacks

The callback functions described in this section must be implemented by all printer browser modules.

PMCOMAddRefProcPtr (page 2895)

PMCOMQueryInterfaceProcPtr (page 2896)

PMCOMReleaseProcPtr (page 2896)

PMPrBrowserAPIVersionProcPtr (page 2916)

PMPrBrowserGetLookupSpecProcPtr (page 2916)

PMPrBrowserGetSelectedPrintersProcPtr (page 2917)

PMPrBrowserInitializeProcPtr (page 2917)

PMPrBrowserPrologueProcPtr (page 2918)

PMPrBrowserResizeProcPtr (page 2918)

PMPrBrowserSelectionStatusProcPtr (page 2919)

PMPrBrowserSyncProcPtr (page 2919)

PMPrBrowserSyncRequestProcPtr (page 2920)

PMPrBrowserTerminateProcPtr (page 2920)

PMPrBrowserWorksetPrintersProcPtr (page 2920)

I/O Module Callbacks

The callback functions described in this section must be implemented by all I/O modules.

Callbacks

GetConnInfoProcPtr

```
typedef OSStatus(* GetConnInfoProcPtr) (
    const void *jobContext,
    CFStringRef *connectionType,
    CFStringRef *pbmPath
);
```

If you name your function `MyGetConnInfoCallback`, you would declare it like this:

```
OSStatus MyGetConnInfoCallback (
   const void *jobContext,
   CFStringRef *connectionType,
   CFStringRef *pbmPath
);
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`PMPrinterModuleDeprecated.h`

PMBeginJobProcPtr

```
typedef OSStatus(* PMBeginJobProcPtr) (
    PMContext printerModuleContext,
    const void *jobContext,
    PMTicketRef jobTicket,
    PMTicketRef *converterSetup
);
```

If you name your function `MyPMBeginJobCallback`, you would declare it like this:

```
OSStatus MyPMBeginJobCallback (
   PMContext printerModuleContext,
   const void *jobContext,
   PMTicketRef jobTicket,
   PMTicketRef *converterSetup
);
```

PMCancelJobProcPtr

```
typedef OSStatus(* PMCancelJobProcPtr) (
    PMContext printerModuleContext,
    const void *jobContext
);
```

If you name your function `MyPMCancelJobCallback`, you would declare it like this:

```
OSStatus MyPMCancelJobCallback (
   PMContext printerModuleContext,
   const void *jobContext
);
```

PMCOMAddRefProcPtr

```
typedef UInt32(* PMCOMAddRefProcPtr) (
    void *thisPointer
);
```

If you name your function `MyPMCOMAddRefCallback`, you would declare it like this:

```
UInt32 MyPMCOMAddRefCallback (
void *thisPointer
);
```

PMCOMQueryInterfaceProcPtr

```
typedef SInt32(* PMCOMQueryInterfaceProcPtr) (
    void *thisPointer,
    CFUUIDBytes iid,
    void **ppv
);
```

If you name your function `MyPMCOMQueryInterfaceCallback`, you would declare it like this:

```
SInt32 MyPMCOMQueryInterfaceCallback (
void *thisPointer,
CFUUIDBytes iid,
void **ppv
);
```

PMCOMReleaseProcPtr

```
typedef UInt32(* PMCOMReleaseProcPtr) (
    void *thisPointer
);
```

If you name your function `MyPMCOMReleaseCallback`, you would declare it like this:

```
UInt32 MyPMCOMReleaseCallback (
void *thisPointer
);
```

PMCreatePrinterBrowserModuleInfoProcPtr

```
typedef OSStatus(* PMCreatePrinterBrowserModuleInfoProcPtr) (
    CFStringRef connectionType,
    CFArrayRef *printerBrowserInfo
);
```

If you name your function `MyPMCreatePrinterBrowserModuleInfoCallback`, you would declare it like this:

```
OSStatus MyPMCreatePrinterBrowserModuleInfoCallback (
   CFStringRef connectionType,
   CFArrayRef *printerBrowserInfo
);
```

PMCreatePrinterTicketsProcPtr

```
typedef OSStatus(* PMCreatePrinterTicketsProcPtr) (
    PMContext printerModuleContext,
    PMTicketRef *printerInfo,
    PMTemplateRef *jobTemplate
);
```

If you name your function `MyPMCreatePrinterTicketsCallback`, you would declare it like this:

```
OSStatus MyPMCreatePrinterTicketsCallback (
   PMContext printerModuleContext,
   PMTicketRef *printerInfo,
   PMTemplateRef *jobTemplate
);
```

PMCreatePrintingDialogExtensionsPathsProcPtr

```
typedef OSStatus(* PMCreatePrintingDialogExtensionsPathsProcPtr)  (
    PMContext printerModuleContext,
    CFArrayRef *pdePaths
);
```

If you name your function `MyPMCreatePrintingDialogExtensionsPathsCallback`, you would declare it like this:

```
OSStatus MyPMCreatePrintingDialogExtensionsPathsCallback (
   PMContext printerModuleContext,
   CFArrayRef *pdePaths
);
```

PMEndJobProcPtr

```
typedef OSStatus(* PMEndJobProcPtr) (
    PMContext printerModuleContext,
    const void *jobContext
);
```

If you name your function `MyPMEndJobCallback`, you would declare it like this:

```
OSStatus MyPMEndJobCallback (
   PMContext printerModuleContext,
   const void *jobContext
);
```

PMImageAccessProcPtr

```
typedef OSStatus(* PMImageAccessProcPtr) (
    PMContext printerModuleContext,
    const void *jobContext,
    CFStringRef grafBase,
    PMDrawingCtx drawingCtx,
    PMImageRef imageRef,
    PMImageRef *outImageRefPtr
);
```

If you name your function `MyPMImageAccessCallback`, you would declare it like this:

```
OSStatus MyPMImageAccessCallback (
   PMContext printerModuleContext,
   const void *jobContext,
   CFStringRef grafBase,
   PMDrawingCtx drawingCtx,
   PMImageRef imageRef,
   PMImageRef *outImageRefPtr
);
```

PMInitializeProcPtr

```
typedef OSStatus(* PMInitializeProcPtr) (
    CFDataRef printerAddress,
    const void *jobContext,
    const PMIOProcs *pmIOProcs,
   const PMNotificationProcPtr pmNotificationProc,
    PMContext *printerModuleContext
);
```

If you name your function `MyPMInitializeCallback`, you would declare it like this:

```
OSStatus MyPMInitializeCallback (
   CFDataRef printerAddress,
   const void *jobContext,
   const PMIOProcs *pmIOProcs,
   const PMNotificationProcPtr pmNotificationProc,
   PMContext *printerModuleContext
);
```

PMIOCloseProcPtr

```
typedef OSStatus(* PMIOCloseProcPtr) (
    const void *jobContext
);
```

If you name your function `MyPMIOCloseCallback`, you would declare it like this:

```
OSStatus MyPMIOCloseCallback (
   const void *jobContext
```

```
);
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`PMPrinterModuleDeprecated.h`

PMIOGetAttributeProcPtr

```
typedef OSStatus(* PMIOGetAttributeProcPtr) (
    const void *jobContext,
    CFStringRef attribute,
    CFTypeRef *result
);
```

If you name your function `MyPMIOGetAttributeCallback`, you would declare it like this:

```
OSStatus MyPMIOGetAttributeCallback (
   const void *jobContext,
   CFStringRef attribute,
   CFTypeRef *result
);
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`PMPrinterModuleDeprecated.h`

PMIOModuleCloseProcPtr

```
typedef OSStatus(* PMIOModuleCloseProcPtr) (
    IOMContext ioModuleContext,
    Boolean abort
);
```

If you name your function `MyCallback`, you would declare it like this:

```
OSStatus MyCallback (
IOMContext ioModuleContext,
Boolean abort
);
```

PMIOModuleGetAttributeProcPtr

```
typedef OSStatus(* PMIOModuleGetAttributeProcPtr) (
    IOMContext ioModuleContext,
    CFStringRef attribute,
    CFTypeRef *result
);
```

If you name your function `MyCallback`, you would declare it like this:

```
OSStatus MyCallback (
IOMContext ioModuleContext,
CFStringRef attribute,
CFTypeRef *result
);
```

PMIOModuleGetConnectionInfoProcPtr

```
typedef OSStatus(* PMIOModuleGetConnectionInfoProcPtr) (
    CFStringRef *connectionType,
    CFStringRef *pbmPath
);
```

If you name your function `MyCallback`, you would declare it like this:

```
OSStatus MyCallback (
CFStringRef *connectionType,
CFStringRef *pbmPath
);
```

PMIOModuleInitializeProcPtr

```
typedef OSStatus(* PMIOModuleInitializeProcPtr) (
    CFDataRef printerAddress,
    IOMContext *ioModuleContextPtr
);
```

If you name your function `MyCallback`, you would declare it like this:

```
OSStatus MyCallback (
CFDataRef printerAddress,
IOMContext *ioModuleContextPtr
);
```

PMIOModuleOpenProcPtr

```
typedef OSStatus(* PMIOModuleOpenProcPtr) (
    IOMContext ioModuleContext,
    PMTicketRef jobTicket,
    UInt32 *bufferSize
);
```

If you name your function `MyCallback`, you would declare it like this:

```
OSStatus MyCallback (
IOMContext ioModuleContext,
PMTicketRef jobTicket,
UInt32 *bufferSize
);
```

PMIOModuleReadProcPtr

```
typedef OSStatus(* PMIOModuleReadProcPtr) (
    IOMContext ioModuleContext,
    Ptr buffer,
    UInt32 *size,
    Boolean *eoj
);
```

If you name your function `MyCallback`, you would declare it like this:

```
OSStatus MyCallback (
IOMContext ioModuleContext,
Ptr buffer,
UInt32 *size,
Boolean *eoj
);
```

PMIOModuleSetAttributeProcPtr

```
typedef OSStatus(* PMIOModuleSetAttributeProcPtr) (
    IOMContext ioModuleContext,
    CFStringRef attribute,
    CFTypeRef data
);
```

If you name your function `MyCallback`, you would declare it like this:

```
OSStatus MyCallback (
IOMContext ioModuleContext,
CFStringRef attribute,
CFTypeRef data
);
```

PMIOModuleStatusProcPtr

```
typedef OSStatus(* PMIOModuleStatusProcPtr) (
    IOMContext ioModuleContext,
    CFStringRef *status
);
```

If you name your function `MyCallback`, you would declare it like this:

```
OSStatus MyCallback (
IOMContext ioModuleContext,
CFStringRef *status
);
```

PMIOModuleTerminateProcPtr

```
typedef OSStatus(* PMIOModuleTerminateProcPtr) (
    IOMContext *ioModuleContextPtr
);
```

If you name your function `MyCallback`, you would declare it like this:

```
OSStatus MyCallback (
IOMContext *ioModuleContextPtr
);
```

PMIOModuleWriteProcPtr

```
typedef OSStatus(* PMIOModuleWriteProcPtr) (
    IOMContext ioModuleContext,
    Ptr buffer,
    UInt32 *size,
    Boolean eoj
);
```

If you name your function `MyCallback`, you would declare it like this:

```
OSStatus MyCallback (
IOMContext ioModuleContext,
Ptr buffer,
UInt32 *size,
Boolean eoj
);
```

PMIOOpenProcPtr

```
typedef OSStatus(* PMIOOpenProcPtr) (
    const void *jobContext
);
```

If you name your function `MyPMIOOpenCallback`, you would declare it like this:

```
OSStatus MyPMIOOpenCallback (
   const void *jobContext
);
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`PMPrinterModuleDeprecated.h`

PMIOReadProcPtr

```
typedef OSStatus(* PMIOReadProcPtr) (
    const void *jobContext,
    Ptr buffer,
    UInt32 *size,
    Boolean *eoj
);
```

If you name your function `MyPMIOReadCallback`, you would declare it like this:

```
OSStatus MyPMIOReadCallback (
   const void *jobContext,
   Ptr buffer,
   UInt32 *size,
   Boolean *eoj
);
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`PMPrinterModuleDeprecated.h`

PMIOSetAttributeProcPtr

```
typedef OSStatus(* PMIOSetAttributeProcPtr) (
    const void *jobContext,
    CFStringRef attribute,
    CFTypeRef data
);
```

If you name your function `MyPMIOSetAttributeCallback`, you would declare it like this:

```
OSStatus MyPMIOSetAttributeCallback (
   const void *jobContext,
   CFStringRef attribute,
   CFTypeRef data
);
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`PMPrinterModuleDeprecated.h`

PMIOStatusProcPtr

```
typedef OSStatus(* PMIOStatusProcPtr) (
    const void *jobContext,
    CFStringRef *status
);
```

If you name your function `MyPMIOStatusCallback`, you would declare it like this:

```
OSStatus MyPMIOStatusCallback (
   const void *jobContext,
   CFStringRef *status
);
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`PMPrinterModuleDeprecated.h`

PMIOWriteProcPtr

```
typedef OSStatus(* PMIOWriteProcPtr) (
    const void *jobContext,
    Ptr buffer,
    UInt32 *size,
    Boolean eoj
);
```

If you name your function `MyPMIOWriteCallback`, you would declare it like this:

```
OSStatus MyPMIOWriteCallback (
   const void *jobContext,
   Ptr buffer,
   UInt32 *size,
   Boolean eoj
);
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`PMPrinterModuleDeprecated.h`

PMJobStreamGetNextBandProcPtr

```
typedef OSStatus(* PMJobStreamGetNextBandProcPtr) (
    const void *jobContext,
    PMRasterBand *pmRasterBand
);
```

If you name your function `MyPMJobStreamGetNextBandCallback`, you would declare it like this:

```
OSStatus MyPMJobStreamGetNextBandCallback (
   const void *jobContext,
   PMRasterBand *pmRasterBand
);
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`PMPrinterModuleDeprecated.h`

PMJobStreamGetPosProcPtr

```
typedef OSStatus(* PMJobStreamGetPosProcPtr) (
    const void *jobContext,
    UInt32 *markerPos
);
```

If you name your function `MyPMJobStreamGetPosCallback`, you would declare it like this:

```
OSStatus MyPMJobStreamGetPosCallback (
   const void *jobContext,
   UInt32 *markerPos
);
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`PMPrinterModuleDeprecated.h`

PMJobStreamOpenProcPtr

```
typedef OSStatus(* PMJobStreamOpenProcPtr) (
    const void *jobContext
);
```

If you name your function `MyPMJobStreamOpenCallback`, you would declare it like this:

```
OSStatus MyPMJobStreamOpenCallback (
   const void *jobContext
);
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`PMPrinterModuleDeprecated.h`

PMJobStreamReadWriteProcPtr

```
typedef OSStatus(* PMJobStreamReadWriteProcPtr) (
    const void *jobContext,
    void *buffPtr,
    UInt32 *size
);
```

If you name your function `MyPMJobStreamReadWriteCallback`, you would declare it like this:

```
OSStatus MyPMJobStreamReadWriteCallback (
```

```
    const void *jobContext,
    void *buffPtr,
    UInt32 *size
);
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`PMPrinterModuleDeprecated.h`

PMJobStreamSetPosProcPtr

```
typedef OSStatus(* PMJobStreamSetPosProcPtr) (
    const void *jobContext,
    SInt16 posMode,
    UInt32 markerPos
);
```

If you name your function `MyPMJobStreamSetPosCallback`, you would declare it like this:

```
OSStatus MyPMJobStreamSetPosCallback (
    const void *jobContext,
    SInt16 posMode,
    UInt32 markerPos
);
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`PMPrinterModuleDeprecated.h`

PMNotificationProcPtr

```
typedef OSStatus(* PMNotificationProcPtr) (
    const void *jobContext,
    CFDictionaryRef notificationDict,
    CFDictionaryRef *notificationReplyDict
);
```

If you name your function `MyPMNotificationCallback`, you would declare it like this:

```
OSStatus MyPMNotificationCallback (
    const void *jobContext,
    CFDictionaryRef notificationDict,
    CFDictionaryRef *notificationReplyDict
);
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`PMPrinterModuleDeprecated.h`

PMPDECloseProcPtr

```
typedef OSStatus(* PMPDECloseProcPtr) (
    PMPDEContext context
);
```

If you name your function `MyPMPDEClose`, you would declare it like this:

```
OSStatus MyPMPDEClose (
   PMPDEContext context
);
```

Parameters

context

A pointer to a custom data structure that contains state information shared among the functions in a printing dialog extension. This is the same context defined by the prologue function—see `PMPDEPrologueProcPtr` (page 2911).

Return Value
A result code.

Discussion
If your pane is visible, your close function is called when the user

- switches to another pane (including the Summary pane)
- changes printers
- dismisses the dialog by clicking Preview, Print, or Save

If your pane is visible and the user cancels the dialog, your close function is not called. The close function is not required to provide any services, and does not return any information to the caller.

PMPDEGetSummaryTextProcPtr

```
typedef OSStatus(* PMPDEGetSummaryTextProcPtr) (
    PMPDEContext context,
    CFArrayRef *titleArray,
    CFArrayRef *summaryArray
);
```

If you name your function `MyPMPDEGetSummary`, you would declare it like this:

```
OSStatus MyPMPDEGetSummary (
   PMPDEContext context,
   CFArrayRef *titleArray,
   CFArrayRef *summaryArray
```

```
);
```

Parameters

context

A pointer to a custom data structure that contains state information shared among the functions in a printing dialog extension. This is the same context defined by the prologue function—see `PMPDEPrologueProcPtr` (page 2911).

titleArray

A pointer to an array of strings that contain brief, localized descriptions of the settings in your user interface. The array and strings must be Core Foundation types.

On entry, the array is undefined. Before returning, you should assign an array of type `CFArrayRef`. The printing system assumes ownership of the array and is responsible for releasing it.

If your printing dialog extension does not supply summary text—or your user interface does not have any settings—you should assign `NULL` before returning.

summaryArray

A pointer to an array of strings that contain brief, localized descriptions of the current values of the settings in your user interface. The array and strings must be Core Foundation types.

On entry, the array is undefined. Before returning, you should assign an array of type `CFArrayRef`. The printing system assumes ownership of the array and is responsible for releasing it.

If your printing dialog extension does not supply summary text—or your user interface does not have any settings—you should assign `NULL` before returning.

Return Value
A result code.

Discussion
Whenever a user switches to the Summary pane in a printing dialog, the printing system calls the summary function of each active printing dialog extension.

The printing system expects *titleArray* and *summaryArray* to be in a one-to-one correspondence. Here's a formal description of what that means. Your user interface has associated with it *n* pairs of strings, each pair consisting of textual descriptions of a single setting and its current value. The two arrays are in a one-to-one correspondence if for all *k* from 0 to *n*-1, the strings *titleArray[k]* and *summaryArray[k]* form a valid pair.

PMPDEInitializeProcPtr

```
typedef OSStatus(* PMPDEInitializeProcPtr) (
    PMPDEContext context,
    PMPDEFlags *flags,
    PMPDERef ref,
    ControlRef embedderUserPane,
    PMPrintSession printSession
);
```

If you name your function `MyPMPDEInitialize`, you would declare it like this:

```
OSStatus MyPMPDEInitialize (
   PMPDEContext context,
   PMPDEFlags *flags,
   PMPDERef ref,
```

```
    ControlRef embedderUserPane,
    PMPrintSession printSession
);
```

Parameters

context

A pointer to a custom data structure that contains state information shared among the functions in a printing dialog extension. This is the same context defined by the prologue function—see `PMPDEPrologueProcPtr` (page 2911).

flags

A pointer to an integer flag that provides information about the capabilities of your printing dialog extension. On entry, the flag is undefined. Before returning, your initialization function should assign a valid feature request flag. For information about the defined flags, see "PDE Feature Flags" (page 2933).

ref

Reserved for future use.

embedderUserPane

The control provided by the printing system to host the user interface of your printing dialog extension. This parameter is a standard Carbon user pane—a root control into which your printing dialog extension embeds the various controls and other elements in your user interface.

Before returning, your initialization function should create static user interface elements—such as controls, icons, and images—and embed them in the user pane provided by the printing system. You should position the user interface elements with respect to the coordinate system of the dialog window, and make them visible. You should also set the initial values of your controls, based on either your default settings or previous settings saved by the user. Do not modify the user pane itself.

If any user interface elements require special handling, you can install Carbon event handlers for them. For example, the Duplex pane in the Print dialog uses an event handler to enable or disable the two binding selection buttons, based on the current checkbox setting.

Do not do any custom drawing at this time, since the user pane is still invisible. If the user decides to display your user interface, the printing system makes it visible and the Control Manager automatically draws your static elements.

printSession

A pointer to a print session object. This is the same print session created by the application prior to displaying the dialog. You can use this parameter to gain access to one of the specialized printing tickets, such as page format or print settings.

Return Value

A result code.

Discussion

If the printing system calls your initialization function, you can safely assume that your prologue function was called first.

If your initialization function returns a nonzero status code, the printing system does not call your summary function.

PMPDEOpenProcPtr

```
typedef OSStatus(* PMPDEOpenProcPtr) (
    PMPDEContext context
);
```

If you name your function `MyPMPDEOpen`, you would declare it like this:

```
OSStatus MyPMPDEOpen (
   PMPDEContext context
);
```

Parameters

context

A pointer to a custom data structure that contains state information shared among the functions in a printing dialog extension. This is the same context defined by the prologue function—see `PMPDEPrologueProcPtr` (page 2911).

Return Value

A result code.

Discussion

If the user selects your pane for display in the dialog, your open function is called immediately before the pane is made visible.

An open function is not required to provide any services, and does not return any information to the caller.

PMPDEPrologueProcPtr

```
typedef OSStatus(* PMPDEPrologueProcPtr) (
    PMPDEContext *context,
    OSType *creator,
    CFStringRef *userOptionKind,
    CFStringRef *title,
    UInt32 *maxH,
    UInt32 *maxV
);
```

If you name your function `MyPMPDEPrologue`, you would declare it like this:

```
OSStatus MyPMPDEPrologue (
   PMPDEContext *context,
   OSType *creator,
   CFStringRef *userOptionKind,
   CFStringRef *title,
   UInt32 *maxH,
   UInt32 *maxV
);
```

Parameters

context

A pointer to a custom data structure that contains state information shared among the functions in a printing dialog extension. On entry, the caller's `PMPDEContext` variable is undefined. Before returning, your prologue function should assign a new context, or `NULL` to indicate that no context exists.

The printing system does not assume ownership of memory used for a context, so your printing dialog extension should release it when it is no longer needed. Typically this is done in your termination function.

creator

Reserved for future use.

userOptionKind

A pointer to a string that contains a unique identifier for the pane implemented by your printing dialog extension. The printing system uses this identifier to determine whether your printing dialog extension implements an Apple-defined pane, or a custom pane defined by you. On entry, the string is undefined. Before returning, your prologue function should assign the appropriate identifier.

If your printing dialog extension implements or overrides an Apple-defined pane, assign one of the identifiers listed in "PDE Pane Kind Identifiers" (page 2935). If your printing dialog extension implements a custom pane, assign a Core Foundation string that contains your own custom identifier.

By convention, a custom identifier takes the form `<domain>.print.pde.<signature>`, where `<domain>` is your vendor-specific domain and `<signature>` is a short name or signature for the pane you are implementing.

The printing system does not assume ownership of this string, so your printing dialog extension should release it when it is no longer needed.

title

A pointer to a string that contains the localized title of the pane implemented by your printing dialog extension. The printing system displays this title in two places—the pane selection pop-up menu and the Summary pane.

On entry, the string is undefined. Before returning, your prologue function should assign a Core Foundation string containing the localized title.

If you are implementing an Apple-defined pane, the printing system may ignore your localized title and use an Apple-defined title instead.

Your printing dialog extension retains ownership of the string, and you should release it when it is no longer needed. Typically this is done in your terminate function.

maxH

A pointer to a value that represents the maximum number of horizontal pixels your user interface requires. When your user interface is made visible, the printing system might use this value to adjust the width of the dialog.

On entry, the value is undefined. Before returning, your prologue function should assign the maximum horizontal extent of your user interface in pixels.

maxV

A pointer to a value that represents the maximum number of vertical pixels your user interface requires. When your user interface is made visible, the printing system uses this value to adjust the height of the dialog.

On entry, the value is undefined. Before returning, your prologue function should assign the maximum vertical extent of your user interface in pixels.

Return Value

A result code.

Discussion
When the printing system displays a printing dialog, it calls the prologue function for each registered dialog extension.

If your prologue function returns a nonzero status code, the printing system does not include your pane in the dialog, and does not call your terminate function. In this circumstance, your prologue function should release any allocated memory or resources before exiting.

While the dialog is open, if some user action causes the printing system to scan for plug-ins again—choosing a different printer, for example—then your prologue function is called again.

PMPDESyncProcPtr

```
typedef OSStatus(* PMPDESyncProcPtr) (
    PMPDEContext context,
    PMPrintSession printSession,
    Boolean reinitializePlugIn
);
```

If you name your function `MyPMPDESync`, you would declare it like this:

```
OSStatus MyPMPDESync (
   PMPDEContext context,
   PMPrintSession printSession,
   Boolean reinitializePlugIn
);
```

Parameters

context

A pointer to a custom data structure that contains state information shared among the functions in a printing dialog extension. This is the same context defined by the prologue function—see `PMPDEPrologueProcPtr` (page 2911).

printSession

A pointer to a printing session object. Your synchronization function should use this parameter to gain access to the job ticket its uses to save user interface settings. These settings are stored as extended data in either the print settings ticket (for Print dialog extensions) or the page format ticket (for Page Setup dialog extensions).

reinitializePlugIn

A `Boolean` value that indicates which synchronization operation to perform. If the value is `true`, you should locate and retrieve the ticket settings and use them to update your user interface settings. If the value is `false`, you should use your current user interface settings to update the ticket settings.

Return Value
A result code.

Discussion
The printing system calls your synchronization function at certain times—in response to user actions—to update either the user interface or the ticket. Also, your printing dialog extension can call your synchronization function directly. For example, your initialization function may need to synchronize your user interface to its default settings.

Here are some calling conditions where the *reinitializePlugin* parameter might be `true`:

- Your own intialization function wants to synchronize your user interface to its default settings.
- The printing system wants you to initialize your user interface from the print settings ticket associated with a different printer.

Here are some calling conditions where the *reinitializePlugin* parameter might be `false`:

- Your user interface is visible, and the user tries to switch to another pane.
- The user saves the current dialog settings.
- The user clicks the Preview button or the Print button.

You may want to check the validity of the current settings in your pane at this time, to avoid recording invalid or inconsistent settings in a job ticket. To prevent the user from switching to another pane until a problem is corrected, return the result code `kPMDontSwitchPDEError`.

PMPDETerminateProcPtr

```
typedef OSStatus(* PMPDETerminateProcPtr) (
    PMPDEContext context,
    OSStatus status
);
```

If you name your function `MyPMPDETerminate`, you would declare it like this:

```
OSStatus MyPMPDETerminate (
   PMPDEContext context,
   OSStatus status
);
```

Parameters

context

A pointer to a custom data structure that contains state information shared among the functions in a printing dialog extension. This is the same context defined by the prologue function—see `PMPDEPrologueProcPtr` (page 2911).

status

Reserved for future use to indicate the conditions under which the user dismissed the dialog.

Return Value

A result code.

Discussion

The printing system calls your termination function when the printing system decides to tear down or rebuild the dialog, or when your pane is no longer needed.

Unless your prologue function returns a nonzero status code, your termination function is always called when the user dismisses the dialog.

Your termination function is also called if your printing feature is no longer relevant—for example, if your feature applies only to postscript printers and the user switches to a raster printer from within the dialog.

Your termination function should release Core Foundation objects, deallocate your context block, and perform any other actions necessary before your user interface is reinitialized or unloaded.

PMPluginGetAPIVersionProcPtr

```
typedef OSStatus(* PMPluginGetAPIVersionProcPtr) (
   PMPlugInHeaderInterface *obj,
   PMPlugInAPIVersion *versionPtr
);
```

If you name your function `MyPMPluginGetAPIVersion`, you would declare it like this:

```
OSStatus MyPMPluginGetAPIVersion (
   PMPlugInHeaderInterface *obj,
   PMPlugInAPIVersion *versionPtr
);
```

Parameters

obj

A pointer to a generic instance of a plug-in interface. You may ignore this parameter, as it isn't useful here.

versionPtr

A pointer to a data structure supplied by the caller for version information about your printing plug-in. Before returning, you should provide the correct version information for your plug-in by assigning version constants to the fields in this structure.

Return Value
A result code.

PMPluginReleaseProcPtr

Defines a pointer to the retain function in a printing plug-in interface. Your custom retain function increments the reference count of an instance of one of these interfaces.

```
typedef OSStatus (*PMPluginReleaseProcPtr) (
   PMPlugInHeaderInterface **objPtr
);
```

If you name your function `MyPMPluginRelease`, you would declare it like this:

```
OSStatus MyPMPluginRelease (
   PMPlugInHeaderInterface **objPtr
);
```

Parameters

objPtr

The address of a pointer to a generic instance of a plug-in interface. Before returning, your retain function should decrement the reference count of the instance, and assign `NULL` to the pointer whose address is provided by this parameter. If the reference count reaches zero, you should delete the instance and any related storage.

Return Value
A result code.

PMPluginRetainProcPtr

```
typedef OSStatus (*PMPluginRetainProcPtr) (
    PMPlugInHeaderInterface *obj
);
```

If you name your function `MyPMPluginRetain`, you would declare it like this:

```
OSStatus MyPMPluginRetain (
    PMPlugInHeaderInterface *obj
);
```

Parameters

obj

A pointer to a generic instance of a plug-in interface. Before returning, your retain function should increment the reference count of the instance.

Return Value

A result code.

PMPrBrowserAPIVersionProcPtr

```
typedef UInt32(* PMPrBrowserAPIVersionProcPtr) ();
```

If you name your function `MyPMPrBrowserAPIVersionCallback`, you would declare it like this:

```
UInt32 MyPMPrBrowserAPIVersionCallback ();
```

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`PMPrinterBrowsers.h`

PMPrBrowserGetLookupSpecProcPtr

```
typedef OSStatus(* PMPrBrowserGetLookupSpecProcPtr) (
    PMPrBrowserRef ref,
    UInt32 specIndex,
    CFDictionaryRef *lookupSpec
);
```

If you name your function `MyPMPrBrowserGetLookupSpecCallback`, you would declare it like this:

```
OSStatus MyPMPrBrowserGetLookupSpecCallback) (
    PMPrBrowserRef ref,
    UInt32 specIndex,
    CFDictionaryRef *lookupSpec
);
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`PMPrinterBrowsers.h`

PMPrBrowserGetSelectedPrintersProcPtr

```
typedef OSStatus(* PMPrBrowserGetSelectedPrintersProcPtr) (
    PMPrBrowserContext context,
    CFArrayRef *printers
);
```

If you name your function `MyPMPrBrowserGetSelectedPrintersCallback`, you would declare it like this:

```
OSStatus MyPMPrBrowserGetSelectedPrintersCallback (
PMPrBrowserContext context,
CFArrayRef *printers
);
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`PMPrinterBrowsers.h`

PMPrBrowserInitializeProcPtr

```
typedef OSStatus(* PMPrBrowserInitializeProcPtr) (
    PMPrBrowserContext context,
    PMPrBrowserRef ref,
    PMPrBrowserCallbacks *callbacks,
    ControlRef pbUserPaneCtlHdl,
    UInt32 numLookupSpecs
);
```

If you name your function `MyPMPrBrowserInitializeCallback`, you would declare it like this:

```
OSStatus MyPMPrBrowserInitializeCallback (
PMPrBrowserContext context,
PMPrBrowserRef ref,
PMPrBrowserCallbacks *callbacks,
ControlRef pbUserPaneCtlHdl,
UInt32 numLookupSpecs
);
```

Availability
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
PMPrinterBrowsers.h

PMPrBrowserPrologueProcPtr

```
typedef OSStatus(* PMPrBrowserPrologueProcPtr) (
    PMPrBrowserContext *context,
    PMPrBrowserFlags prologueFlags,
    CFStringRef *title,
    UInt32 *minH,
    UInt32 *minV,
    UInt32 *maxH,
    UInt32 *maxV
);
```

If you name your function `MyPMPrBrowserPrologueCallback`, you would declare it like this:

```
OSStatus MyPMPrBrowserPrologueCallback (
PMPrBrowserContext *context,
PMPrBrowserFlags prologueFlags,
CFStringRef *title,
UInt32 *minH,
UInt32 *minV,
UInt32 *maxH,
UInt32 *maxV
);
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
PMPrinterBrowsers.h

PMPrBrowserResizeProcPtr

```
typedef OSStatus(* PMPrBrowserResizeProcPtr) (
    PMPrBrowserContext context,
    const Rect *frameRect
);
```

If you name your function `MyPMPrBrowserResizeCallback`, you would declare it like this:

```
OSStatus MyPMPrBrowserResizeCallback (
PMPrBrowserContext context,
const Rect *frameRect
);
```

Availability
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
PMPrinterBrowsers.h

PMPrBrowserSelectionStatusProcPtr

```
typedef OSStatus(* PMPrBrowserSelectionStatusProcPtr) (
    PMPrBrowserRef ref,
    Boolean selected,
    Boolean addNow
);
```

If you name your function `MyPMPrBrowserSelectionStatusCallback`, you would declare it like this:

```
OSStatus MyPMPrBrowserSelectionStatusCallback (
PMPrBrowserRef ref,
Boolean selected,
Boolean addNow
);
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
PMPrinterBrowsers.h

PMPrBrowserSyncProcPtr

```
typedef OSStatus(* PMPrBrowserSyncProcPtr) (
    PMPrBrowserContext context
);
```

If you name your function `MyPMPrBrowserSyncCallback`, you would declare it like this:

```
OSStatus MyPMPrBrowserSyncCallback (
PMPrBrowserContext context
);
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
PMPrinterBrowsers.h

PMPrBrowserSyncRequestProcPtr

```
typedef OSStatus(* PMPrBrowserSyncRequestProcPtr) (
    PMPrBrowserRef ref
);
```

If you name your function `MyPMPrBrowserSyncRequestCallback`, you would declare it like this:

```
OSStatus MyPMPrBrowserSyncRequestCallback (
PMPrBrowserRef ref
);
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`PMPrinterBrowsers.h`

PMPrBrowserTerminateProcPtr

```
typedef OSStatus(* PMPrBrowserTerminateProcPtr) (
    PMPrBrowserContext context,
    OSStatus status
);
```

If you name your function `MyPMPrBrowserTerminateCallback`, you would declare it like this:

```
OSStatus MyPMPrBrowserTerminateCallback (
PMPrBrowserContext context,
OSStatus status
);
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`PMPrinterBrowsers.h`

PMPrBrowserWorksetPrintersProcPtr

```
typedef OSStatus(* PMPrBrowserWorksetPrintersProcPtr) (
    PMPrBrowserContext context,
    CFArrayRef printers
);
```

If you name your function `MyPMPrBrowserWorksetPrintersCallback`, you would declare it like this:

```
OSStatus MyPMPrBrowserWorksetPrintersCallback (
PMPrBrowserContext context,
```

```
CFArrayRef printers
);
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`PMPrinterBrowsers.h`

PMPrintJobProcPtr

```
typedef OSStatus(* PMPrintJobProcPtr) (
    PMContext printerModuleContext,
    const void *jobContext,
    PMTicketRef jobTicket,
    const PMJobStreamProcs *inDataProcs
);
```

If you name your function `MyPMPrintJobCallback`, you would declare it like this:

```
OSStatus MyPMPrintJobCallback (
   PMContext printerModuleContext,
   const void *jobContext,
   PMTicketRef jobTicket,
   const PMJobStreamProcs *inDataProcs
);
```

PMPrintPageProcPtr

```
typedef OSStatus(* PMPrintPageProcPtr) (
    PMContext printerModuleContext,
    const void *jobContext,
    PMTicketRef jobTicket,
   const PMJobStreamGetNextBandProcPtr pmJobStreamGetNextBandProc
);
```

If you name your function `MyPMPrintPageCallback`, you would declare it like this:

```
OSStatus MyPMPrintPageCallback (
   PMContext printerModuleContext,
   const void *jobContext,
   PMTicketRef jobTicket,
   const PMJobStreamGetNextBandProcPtr pmJobStreamGetNextBandProc
);
```

PMTerminateProcPtr

```
typedef OSStatus(* PMTerminateProcPtr) (
    PMContext *printerModuleContext,
    const void *jobContext
);
```

If you name your function `MyPMTerminateCallback`, you would declare it like this:

```
OSStatus MyPMTerminateCallback (
   PMContext *printerModuleContext,
   const void *jobContext
);
```

Data Types

Printing Plug-in Data Types

PMPlugInHeader

Defines the table of callback functions in the printing plug-in interface.

```
struct PMPlugInHeader
{
    OSStatus (*Retain) (
        PMPlugInHeaderInterface *obj
    );

    OSStatus (*Release) (
        PMPlugInHeaderInterface **objPtr
    );

    OSStatus (*GetAPIVersion) (
        PMPlugInHeaderInterface *obj,
        PMPlugInAPIVersion *versionPtr
    );
};
typedef struct PMPlugInHeader PMPlugInHeader;
```

Fields

`Retain`
: A pointer to a function that satisfies the requirements for a printing plug-in retain function, as described in `PMPluginRetainProcPtr` (page 2916).

`Release`
: A pointer to a function that satisfies the requirements for a printing plug-in release function, as described in `PMPluginReleaseProcPtr` (page 2915).

`GetAPIVersion`
: A pointer to a function that satisfies the requirements for a printing plug-in version function, as described in `PMPluginGetAPIVersionProcPtr` (page 2915).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`PMPluginHeader.h`

PMPlugInHeaderInterface

Defines a generic instance of the `PMPlugInHeader` interface—the printing system passes a parameter of this type when calling the three interface functions.

```
struct PMPlugInHeaderInterface {
    const PMPlugInHeader *vtable;
};
typedef struct PMPlugInHeaderInterface PMPlugInHeaderInterface;
```

Fields
`vtable`
A pointer to the function table for an implementation of the `PMPlugInHeader` interface.

Discussion
When the printing system calls one of the three functions in the `PMPlugInHeader` interface, a pointer to this generic data type is supplied as a parameter. It's really the address of the plug-in instance supplied by your query interface function (see *Core Foundation Plug-in Services*).

To gain access to your instance data—including the field that holds the reference count—you need to cast this pointer to the actual instance type defined in your plug-in.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`PMPluginHeader.h`

PMPlugInAPIVersion

Contains `PMPlugInHeader` interface version information that printing plug-ins must provide to the printing system.

```
struct PMPlugInAPIVersion {
    UInt32 buildVersionMajor;
    UInt32 buildVersionMinor;
    UInt32 baseVersionMajor;
    UInt32 baseVersionMinor;
};
```

Fields
`buildVersionMajor`
An integer value that specifies the major component of the API version with which the plug-in was compiled.

`buildVersionMinor`
An integer value that specifies the minor component of the API version with which the plug-in was compiled.

`baseVersionMajor`
: An integer value that specifies the major component of the base API version with which this plug-in is forward compatible.

`baseVersionMinor`
: An integer value that specifies the minor component of the base API version with which this plug-in is forward compatible.

Printing Dialog Extension Data Types

PlugInIntfVTable

Defines the table of callback functions in the printing dialog extension plug-in interface.

```
struct PlugInIntfVTable
{
    PMPlugInHeader plugInHeader;

    OSStatus (*Prologue) (
        PMPDEContext *context,
        OSType *creator,
        CFStringRef *userOptionKind,
        CFStringRef *title,
        UInt32 *maxH,
        UInt32 *maxV
    );

    OSStatus (*Initialize) (
        PMPDEContext context,
        PMPDEFlags *flags,
        PMPDERef ref,
        ControlRef embedderUserPane,
        PMPrintSession printSession
    );

    OSStatus (*Sync) (
        PMPDEContext context,
        PMPrintSession printSession,
        Boolean reinitializePlugIn
    );

    OSStatus (*GetSummaryText) (
        PMPDEContext context,
        CFArrayRef *titleArray,
        CFArrayRef *summaryArray
    );

    OSStatus (*Open) (
        PMPDEContext context
    );

    OSStatus (*Close) (
        PMPDEContext context
    );
```

```
    OSStatus (*Terminate) (
        PMPDEContext context,
        OSStatus status
    );
};
typedef struct PlugInIntfVTable PlugInIntfVTable;
```

Fields

`plugInHeader`
A table of pointers to the three functions in the printing plug-in interface.

`Prologue`
A pointer to a function that satisfies the requirements for the prologue function in a printing dialog extension, as described in `PMPDEPrologueProcPtr` (page 2911).

`Initialize`
A pointer to a function that satisfies the requirements for the initialization function in a printing dialog extension, as described in `PMPDEInitializeProcPtr` (page 2909).

`Sync`
A pointer to a function that satisfies the requirements for the synchronization function in a printing dialog extension, as described in `PMPDESyncProcPtr` (page 2913).

`GetSummaryText`
A pointer to a function that satisfies the requirements for the summary function in a printing dialog extension, as described in `PMPDEGetSummaryTextProcPtr` (page 2908).

`Open`
A pointer to a function that satisfies the requirements for the open function in a printing dialog extension, as described in `PMPDEOpenProcPtr` (page 2911).

`Close`
A pointer to a function that satisfies the requirements for the close function in a printing dialog extension, as described in `PMPDECloseProcPtr` (page 2908).

`Terminate`
A pointer to a function that satisfies the requirements for the termination function in a printing dialog extension, as described in `PMPDETerminateProcPtr` (page 2914).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`PMPrintingDialogExtensionsDeprecated.h`

PlugInIntf

Defines a generic instance of the printing dialog extension plug-in interface.

```
struct PlugInIntf {
    PlugInIntfVTable *vtable;
};
```

Fields

`vtable`
A pointer to the table of callback functions in a printing dialog extension.

PMPDEContext

Defines a generic private context for a printing dialog extension—the printing system passes a parameter of this type when calling functions in the `PlugInIntfVTable` interface.

```
typedef struct OpaquePMPDEContext* PMPDEContext;
```

Discussion
A **context** is a pointer to a custom data structure shared by the functions in a printing dialog extension. In Mac OS X, Carbon applications can open the same printing dialog in several document windows concurrently. Printing dialog extensions must support this possibility, which means they must be reentrant.

To ensure reentrancy, your printing dialog extension uses dynamically allocated memory for state information specific to one dialog window. Your prologue function allocates this memory and supplies its address—called a context—to the printing system. The printing system makes sure that the context associated with the current session (the active dialog) is passed in to your other functions as a calling parameter.

The context you provide is a generic pointer of type `PMPDEContext`. You can gain access to your actual context data by casting this parameter to the actual context type defined in your printing dialog extension.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`PMPrintingDialogExtensionsDeprecated.h`

PMPDEFlags

Defines an integer flag that provides additional information about a printing dialog extension to the printing system. A value of this data type is passed back to the printing system by the initialize function.

```
typedef UInt32 PMPDEFlags;
enum {
    kPMPDENoFlags = 0,
    kPMPDENoSummary = 1,
    kPMPDEAllFlags = -1
};
```

Discussion
The printing system examines this integer flag to learn more about its capabilities. For a description of the initialize function, see `PMPDEInitializeProcPtr` (page 2909).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`PMPrintingDialogExtensionsDeprecated.h`

PMPDERef

Defines a generic instance of the printing dialog extension plug-in interface.

```
typedef struct OpaquePMPDERef* PMPDERef;
```

Discussion
A value of this type is passed to the initialize function. For a description of the initialize function, see `PMPDEInitializeProcPtr` (page 2909).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`PMPrintingDialogExtensionsDeprecated.h`

Printer Module Data Types

PMProcs

Defines the table of callback functions in the printer module plug-in interface.

```
struct PMProcs
{
    PMPlugInHeader                  pluginHeader;
    PMCreatePBMInfoProcPtr          CreatePBMInfo;
    PMInitializeProcPtr             Initialize;
    PMCreatePDEPathsProcPtr         CreatePDEPaths;
    PMCreatePrinterTicketsProcPtr   CreatePrinterTickets;
    PMBeginJobProcPtr               BeginJob;
    PMPrintJobProcPtr               PrintJob;
    PMPrintPageProcPtr              PrintPage;
    PMImageAccessProcPtr            ImageAccess;
    PMCancelJobProcPtr              CancelJob;
    PMEndJobProcPtr                 EndJob;
    PMTerminateProcPtr              Terminate;
};
```

PMInterface

Defines a generic instance of the printer module plug-in interface.

```
struct PMInterface {
    const PMProcs *vtable;
};
typedef struct PMInterface PMInterface;
```

Fields
`vtable`
A pointer to the table of callback functions in a printer module.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
PMPrinterModuleDeprecated.h

PMInterfaceRef

Defines a pointer to a generic instance of the printer module plug-in interface.

```
typedef PMInterface* PMInterfaceRef;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
PMPrinterModuleDeprecated.h

PMIOProcs

Defines the table of callback functions in the `PMIOProcs` interface.

```
struct PMIOProcs
{
    CFIndex                 version;
    GetConnInfoProcPtr      GetConnInfoProc;
    PMIOOpenProcPtr         PMIOOpenProc;
    PMIOReadProcPtr         PMIOReadProc;
    PMIOWriteProcPtr        PMIOWriteProc;
    PMIOStatusProcPtr       PMIOStatusProc;
    PMIOGetAttributeProcPtr PMIOGetAttributeProc;
    PMIOSetAttributeProcPtr PMIOSetAttributeProc;
    PMIOCloseProcPtr        PMIOCloseProc;
};
```

PMJobStreamProcs

Defines the table of callback functions in the printer module data access interface.

```
struct PMJobStreamProcs
{
    CFIndex                         version;
    PMJobStreamOpenProcPtr          PMJobStreamOpenProc;
    PMJobStreamReadWriteProcPtr     PMJobStreamReadProc;
    PMJobStreamReadWriteProcPtr     PMJobStreamWriteProc;
    PMJobStreamGetPosProcPtr        PMJobStreamGetPosProc;
    PMJobStreamSetPosProcPtr        PMJobStreamSetPosProc;
    PMJobStreamGetPosProcPtr        PMJobStreamGetEOFProc;
};
```

PMContext

Defines an opaque type for a private context in a printer module.

```
typedef struct OpaquePMContext* PMContext;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`PMPrinterModuleDeprecated.h`

PMDrawingCtx

Defines an opaque type for a drawing context in a printer module.

```
typedef struct OpaquePMDrawingCtx* PMDrawingCtx;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`PMPrinterModuleDeprecated.h`

PMImageRef

Defines an opaque type for an image in a printer module.

```
typedef struct OpaquePMImageRef* PMImageRef;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`PMPrinterModuleDeprecated.h`

Printer Browser Module Data Types

PMInterfacePrBrowser

Defines the table of callback functions in the printer browser module plug-in interface.

```
struct PMInterfacePrBrowser
{
    IUnknownVTbl                            u;
    PMPrBrowserGetSelectedPrintersProcPtr   getSelectedPrinters;
    PMPrBrowserInitializeProcPtr            initialize;
    PMPrBrowserPrologueProcPtr              prologue;
    PMPrBrowserResizeProcPtr                resize;
    PMPrBrowserSyncProcPtr                  sync;
    PMPrBrowserTerminateProcPtr             terminate;
    PMPrBrowserWorksetPrintersProcPtr       worksetPrinters;
};
```

PMInterfacePrBrowserPtr

Defines a pointer to a function table for the `PMInterfacePrBrowser` interface.

```
typedef PMInterfacePrBrowser* PMInterfacePrBrowserPtr;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`PMPrinterBrowsers.h`

PMPrBrowserCallbacks

Defines the table of Print Center callback routines for printer browser modules.

```
struct PMPrBrowserCallbacks
{
    CFIndex                             version;
    PMPrBrowserGetLookupSpecProcPtr     getLookupSpec;
    PMPrBrowserSyncRequestProcPtr       syncRequest;
    PMPrBrowserSelectionStatusProcPtr   selStatus;
};
```

PMPrBrowserCallbacksPtr

Defines a pointer to a table of Print Center callbacks for printer browser modules.

```
typedef PMPrBrowserCallbacks* PMPrBrowserCallbacksPtr;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`PMPrinterBrowsers.h`

PMPrBrowserContext

Defines an opaque type for a private context in a printer browser module.

```
typedef struct OpaquePMPrBrowserContext* PMPrBrowserContext;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`PMPrinterBrowsers.h`

PMPrBrowserFlags

Defines an integer flag that provides additional information about a printer browser module to the printing system.

```
typedef UInt32 PMPrBrowserFlags;
enum {
    kPMPrBrowserPCNoFlags = 0,
    kPMPrBrowserPCNoUI = 1,
    kPMPrBrowserPCAllFlags = -1
};
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`PMPrinterBrowsers.h`

PMPrBrowserRef

Defines an opaque type for an instance of the printer browser module plug-in interface.

```
typedef struct OpaquePMPrBrowserRef* PMPrBrowserRef;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`PMPrinterBrowsers.h`

PMInterfaceAPIVersion

Defines the table of callback functions in the `PMInterfaceAPIVersion` interface.

```
struct PMInterfaceAPIVersion
{
    IUnknownVTbl                    u;
    PMPrBrowserAPIVersionProcPtr    apiVersion;
};
```

PMInterfaceAPIVersionPtr

Defines a pointer to a function table for the `PMInterfaceAPIVersion` interface.

```
typedef PMInterfaceAPIVersion* PMInterfaceAPIVersionPtr;
```

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`PMPrinterBrowsers.h`

I/O Module Data Types

IOMContext

Defines an opaque type for a private context in an I/O module.

```
typedef struct OpaqueIOMContext* IOMContext;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`PMIOModule.h`

IOMInterface

Defines a generic instance of the I/O module plug-in interface.

```
struct IOMInterface {
    const IOMProcs *vtable;
};
```

Fields
`vtable`
A pointer to the table of callback functions in an I/O module.

IOMInterfaceRef

Defines a pointer to a generic instance of the I/O module plug-in interface.

```
typedef const IOMInterface* IOMInterfaceRef;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
PMIOModule.h

IOMProcs

Defines the table of callback functions in the I/O module plug-in interface.

```
struct IOMProcs
{
    PMPlugInHeader pluginHeader;

    PMIOModuleGetConnectionInfoProcPtr  GetConnectionInfo;
    PMIOModuleInitializeProcPtr         Initialize;
    PMIOModuleOpenProcPtr               Open;
    PMIOModuleReadProcPtr               Read;
    PMIOModuleWriteProcPtr              Write;
    PMIOModuleStatusProcPtr             Status;
    PMIOModuleGetAttributeProcPtr       GetAttribute;
    PMIOModuleSetAttributeProcPtr       SetAttribute;
    PMIOModuleCloseProcPtr              Close;
    PMIOModuleTerminateProcPtr          Terminate;
};
typedef struct IOMProcs IOMProcs;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
PMIOModule.h

Constants

PDE Feature Flags

Specify the flags returned by a printing dialog extension in its initialization function.

```
typedef UInt32 PMPDEFlags;
enum {
    kPMPDENoFlags = 0,
    kPMPDENoSummary = 1,
    kPMPDEAllFlags = -1
};
```

Constants

`kPMPDENoFlags`

Specifies that a printing dialog extension does not have any special capabilities to report to the printing system.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `PMPrintingDialogExtensionsDeprecated.h`.

`kPMPDENoSummary`

Specifies that a printing dialog extension does not provide summary information. Your initialization function can indicate this to the printing system by using the `flags` parameter to pass back this constant. For more information about the initialization function, see `PMPDEInitializeProcPtr` (page 2909). For more information about providing summary information, see `PMPDEGetSummaryTextProcPtr` (page 2908).

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `PMPrintingDialogExtensionsDeprecated.h`.

`kPMPDEAllFlags`

Not used.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `PMPrintingDialogExtensionsDeprecated.h`.

PDE Interface Identifier

Specifies the unique identifier for the printing dialog extension plug-in interface.

```
#define kDialogExtensionIntfIDStr
CFSTR("A996FD7E-B738-11D3-8519-0050E4603277")
```

Constants

`kDialogExtensionIntfIDStr`

A UUID that a printing dialog extension should use in its query interface function to verify that the caller wants an instance of a printing dialog extension plug-in interface, and not some other plug-in interface.

Discussion

For more information about the query interface function, see *Core Foundation Plug-in Services*..

PDE Interface Version

Specify the major and minor components of the version numbers for the interface supported by your printing dialog extension.

```
#define kPDEBuildVersionMajor   1
#define kPDEBuildVersionMinor   0
#define kPDEBaseVersionMajor    1
#define kPDEBaseVersionMinor    0
```

Constants

`kPDEBuildVersionMajor`
Specifies that Apple has designated the current release of the printing dialog extension plug-in interface to be major version 1, as in 1.0.

`kPDEBuildVersionMinor`
Specifies that Apple has designated the current release of the printing dialog extension plug-in interface to be minor version 0, as in 1.0.

`kPDEBaseVersionMajor`
Specifies that Apple has designated the first release of the printing dialog extension plug-in interface to be major version 1, as in 1.0.

`kPDEBaseVersionMinor`
Specifies that Apple has designated the first release of the printing dialog extension plug-in interface to be minor version 0, as in 1.0.

PDE Pane Kind Identifiers

Identifiers associated with the Apple-defined panes in a printing dialog.

```
#define kPMColorPDEKindID                   CFSTR("com.apple.print.pde.ColorKind")
#define kPMCopiesAndPagesPDEKindID
CFSTR("com.apple.print.pde.CopiesAndPagesKind")
#define kPMCoverPagePDEKindID
CFSTR("com.apple.print.pde.CoverPageKind")
#define kPMDuplexPDEKindID                 CFSTR("com.apple.print.pde.DuplexKind")
#define kPMLayoutPDEKindID
CFSTR("com.apple.print.pde.LayoutUserOptionKind")
#define kPMOutputOptionsPDEKindID
CFSTR("com.apple.print.pde.OutputOptionsKind")
#define kPMPageAttributesKindID
CFSTR("com.apple.print.pde.PageAttributesKind")
#define kPMPaperFeedPDEKindID
CFSTR("com.apple.print.pde.PaperFeedKind")
#define kPMPriorityPDEKindID
CFSTR("com.apple.print.pde.PriorityKind")
#define kPMRotationScalingPDEKindID
CFSTR("com.apple.print.pde.RotationScalingKind")
```

Constants

`kPMColorPDEKindID`
Identifies the Print dialog pane named Color Option.

`kPMCopiesAndPagesPDEKindID`
Identifies the Print dialog pane named Copies & Pages.

`kPMCoverPagePDEKindID`
Identifies the Print dialog pane named Cover Page.

`kPMDuplexPDEKindID`
Identifies the Print dialog pane named Duplex.

`kPMLayoutPDEKindID`
Identifies the Print dialog pane named Layout.

`kPMOutputOptionsPDEKindID`
Identifies the Print dialog pane named Output Options.

`kPMPageAttributesKindID`
Identifies the Page Setup dialog pane named Page Attributes.

`kPMPaperFeedPDEKindID`
Identifies the Print dialog pane named Paper Feed.

`kPMPriorityPDEKindID`
Identifies the Print dialog pane named Priority.

`kPMRotationScalingPDEKindID`
Identifies the Print dialog pane named Rotation & Scaling.

PDE Ticket Identifiers

Specify the types of printing job tickets used by printing dialog extensions.

```
#define kPDE_PMPrintSettingsRef     CFSTR("PMPrintSettingsTicket")
#define kPDE_PMPageFormatRef        CFSTR("PMPageFormatTicket")
#define kPDE_PMJobTemplateRef       CFSTR("PMJobTemplateTicket")
#define kPDE_PMPrinterInfoRef       CFSTR("PMPrinterInfoTicket")
```

Constants

`kPDE_PMPrintSettingsRef`
Specifies a print settings ticket.

`kPDE_PMPageFormatRef`
Specifies a page format ticket.

`kPDE_PMJobTemplateRef`
Specifies a job template ticket.

`kPDE_PMPrinterInfoRef`
Specifies a printer info ticket.

PDE Type Identifiers

Specify the different types of printing dialog extensions.

```
#define kAppPageSetupDialogTypeIDStr
CFSTR("B9A0DA98-E57F-11D3-9E83-0050E4603277")
#define kAppPrintDialogTypeIDStr
CFSTR("BCB07250-E57F-11D3-8CA6-0050E4603277")
#define kPrinterModuleTypeIDStr
CFSTR("BDB091F4-E57F-11D3-B5CC-0050E4603277")
```

Constants

`kAppPageSetupDialogTypeIDStr`
The identifier for a Page Setup dialog extension supplied with an application.

`kAppPrintDialogTypeIDStr`
The identifier for a Print dialog extension supplied with an application.

`kPrinterModuleTypeIDStr`
The identifier for a Print dialog extension supplied with a printer module.

Discussion
Apple associates each type of printing dialog extension with a UUID, represented as a constant Core Foundation string.

One or more of these constants is used in the `CFPlugInTypes` property list entry of a printing dialog extension, to declare a supported type and associate the type with a factory function that implements it.

For more information about factory functions, see *Core Foundation Plug-in Services*.

I/O Module Interface Version

Specify the major and minor components of the version numbers for the interface supported by your I/O module.

```
enum {
    kIOMBuildVersionMajor = 1,
    kIOMBuildVersionMinor = 0,
    kIOMBaseVersionMajor = 1,
    kIOMBaseVersionMinor = 0
};
```

Printer Module Interface Version

Specify the major and minor components of the version numbers for the interface supported by your printer module.

```
enum {
    kPMBuildVersionMajor = 1,
    kPMBuildVersionMinor = 0,
    kPMBaseVersionMajor = 1,
    kPMBaseVersionMinor = 0
};
```

Printer Module Status Codes

Specify the status and error event codes and keys reported by the printer module (used in PMNotificationProc).

```
enum {
    kPMEventPrinterStatus = 4000,
    kPMEventErrorOccurred = 4001,
    kPMEventRecoverableErrorOccurred = 4002,
    kPMEventRecoverableErrorCleared = 4003
};
```

Other Printer Module Constants

Defined in PMPrinterModule.h.

```
enum {
    kPMBrowserInfoNumValues = 4
};
```

Print Center Feature Flags

Specify the flags returned by Print Center in its prologue function.

```
typedef UInt32 PMPrBrowserFlags;
enum {
    kPMPrBrowserPCNoFlags = 0,
    kPMPrBrowserPCNoUI = 1,
    kPMPrBrowserPCAllFlags = -1
};
```

Print Center Signatures

Specify the Print Center creator code and icon signatures.

```
enum {
    kPMPrBrowserPCCreator = 'pctr',
    kPMPrBrowserWorksetPrinterIconType = 'wspr',
    kPMPrBrowserUnknownPrinterIconType = '?ptr'
};
```

Result Codes

The table below lists the result codes defined in the Carbon Printing Manager for functions in the printing plug-in interfaces.

Result Code	Value	Description
`kPMCloseFailed`	-9785	A file or connection could not be closed. Available in Mac OS X v10.0 and later.
`kPMDontSwitchPDEError`	-9531	Tells the printing system not to switch out of the current pane. Available in Mac OS X v10.0 and later.
`kPMEditRequestFailed`	-9544	Error handling request to update Edit menu. Available in Mac OS X v10.0 and later.
`kPMInvalidLookupSpec`	-9542	Error retrieving lookup specification. Available in Mac OS X v10.0 and later.
`kPMInvalidPBMRef`	-9540	Invalid printer browser module. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
kPMInvalidPDEContext	-9530	Invalid printing dialog extension context. Available in Mac OS X v10.0 and later.
kPMInvalidPrinterAddress	-9780	Invalid printer address. NetInfo printcap entry not found.Unable to open USB interface. Available in Mac OS X v10.0 and later.
kPMIOAttrNotAvailable	-9787	I/O attribute not available on current connection type. Available in Mac OS X v10.0 and later.
kPMNoSelectedPrinters	-9541	No selected printers or error getting selection. Available in Mac OS X v10.0 and later.
kPMOpenFailed	-9781	A file or connection could not be opened. Available in Mac OS X v10.0 and later.
kPMPrBrowserNoUI	-9545	User interface function call with no user interface present. Available in Mac OS X v10.0 and later.
kPMReadFailed	-9782	A file or connection read operation failed. Available in Mac OS X v10.0 and later.
kPMReadGotZeroData	-9788	A file or connection read operation returned no data. Available in Mac OS X v10.1 and later.
kPMStatusFailed	-9784	Connection status failed. Available in Mac OS X v10.0 and later.
kPMSyncRequestFailed	-9543	Error handling sync request. Available in Mac OS X v10.0 and later.
kPMUnsupportedConnection	-9786	Connection type not supported. Available in Mac OS X v10.0 and later.
kPMWriteFailed	-9783	A file or connection write operation failed. Available in Mac OS X v10.0 and later.

TextEdit Reference (Not Recommended)

Framework: Carbon/Carbon.h

Declared in TextEdit.h
TSMTE.h

Important: The information in this document is obsolete and should not be used for new development.

Overview

TextEdit was originally designed to handle editable text items in dialog boxes and other parts of the Mac OS system software. Although TextEdit was enhanced to provide more text-handling support, especially in its handling of multi-script text, it retained some of its original limitations. TextEdit was never intended to manipulate lengthy documents or text requiring more than rudimentary formatting.

TextEdit has been deprecated for deployment targets Mac OS X version 10.4 and later. The replacement API is Multilingual Text Engine (MLTE). MLTE offers additional features such as Unicode text editing, document-wide tabs, full justification of text, support for more than 32 KB of text, built-in scroll bar handling, built-in printing support, support for inline input, support for the advanced font features of Apple Type Services for Unicode Imaging (ATSUI), and support for multiple levels of undo.

You should use MLTE to replace TextEdit functions in your existing applications. With MLTE, you can significantly reduce the number of lines in your code because MLTE handles most of the low-level tasks you had to code in the past. MLTE provides a quick and easy solution for static display of Unicode text and for creating Unicode-compliant text-editing fields within an application. For more information, see *Handling Unicode Text Editing With MLTE*.

Functions by Task

Activating and Deactivating an Edit Structure

`TEActivate` (page 2968) Deprecated in Mac OS X v10.4

Activates the specified edit structure. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`TEDeactivate` (page 2974) Deprecated in Mac OS X v10.4

Deactivates the specified edit structure. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

Using Additional TextEdit Features

`TEFeatureFlag` (page 2976) Deprecated in Mac OS X v10.4
Turns a specified feature on or off or returns the current status of that feature. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

Checking, Setting, and Replacing Styles

`TEContinuousStyle` (page 2970) Deprecated in Mac OS X v10.4
Determines whether a given character attribute is continuous over the current selection range. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`TEGetStyle` (page 2981) Deprecated in Mac OS X v10.4
Gets character attributes for the specified text. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`TEGetStyleScrapHandle` (page 2983) Deprecated in Mac OS X v10.4
Creates a style scrap structure, copies the character attributes associated with the current selection range into it, and returns a handle to it. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`TENumStyles` (page 2987) Deprecated in Mac OS X v10.4
Returns the number of character attribute changes contained in the specified range, counting one for the start of the range. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`TEReplaceStyle` (page 2989) Deprecated in Mac OS X v10.4
Replaces any character attributes in the current selection range that match the specified existing character attributes with the specified new character attributes. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`TESetStyle` (page 2996) Deprecated in Mac OS X v10.4
Sets new character attributes, in the specified edit structure, for the current selection range. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`TEStyleInsert` (page 2998) Deprecated in Mac OS X v10.4
Inserts the specified text immediately before the selection range or the insertion point in the edit structure's text and applies the specified character attributes to the text, redrawing the text if necessary. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`TEUseStyleScrap` (page 3002) Deprecated in Mac OS X v10.4
Assigns new character attributes to the specified range of text in the designated edit structure. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

Customizing TextEdit

`TECustomHook` (page 2972) Deprecated in Mac OS X v10.4
Replaces a default TextEdit hook function with a customized function and returns the address of the replaced function. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`TEGetDoTextHook` (page 2977) Deprecated in Mac OS X v10.4
Obtains a universal procedure pointer to your do-text-hook callback. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`TEGetFindWordHook` (page 2977) Deprecated in Mac OS X v10.4

Obtains a universal procedure pointer to your set-find-word-hook callback. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`TEGetRecalcHook` (page 2980) Deprecated in Mac OS X v10.4

Obtains a universal procedure pointer to your recalculation callback. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`TESetClickLoop` (page 2992) Deprecated in Mac OS X v10.4

Installs the address of the application-supplied click loop function in the `clikLoop` field of the edit structure. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`TESetDoTextHook` (page 2993) Deprecated in Mac OS X v10.4

Sets your do-text-hook callback to be used by TextEdit. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`TESetFindWordHook` (page 2993) Deprecated in Mac OS X v10.4

Sets your set-find-word-hook callback to be used by TextEdit. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`TESetRecalcHook` (page 2994) Deprecated in Mac OS X v10.4

Sets your recalculation callback to be used by TextEdit. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

Displaying and Scrolling Text

`TEAutoView` (page 2968) Deprecated in Mac OS X v10.4

Enables and disables automatic scrolling of the text in the specified edit structure. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`TECalText` (page 2969) Deprecated in Mac OS X v10.4

Recalculates the beginnings of all lines of text in the specified edit structure. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`TEGetHeight` (page 2978) Deprecated in Mac OS X v10.4

Returns the total height of all of the lines in the text between and including the specified starting and ending lines. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`TEPinScroll` (page 2988) Deprecated in Mac OS X v10.4

Scrolls the text within the view rectangle of the specified edit structure by the designated number of pixels. Scrolling stops when the last line of text is scrolled into view. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`TEScroll` (page 2990) Deprecated in Mac OS X v10.4

Scrolls the text within the view rectangle of the specified edit structure by the designated number of pixels. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`TESelView` (page 2991) Deprecated in Mac OS X v10.4

Ensures, once automatic scrolling has been enabled by a call to the `TEAutoView` function or through the `TEFeatureFlag` function, that the selection range is visible, scrolling it into the view rectangle if necessary. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`TESetAlignment` (page 2992) Deprecated in Mac OS X v10.4
Sets the alignment of the specified text in an edit structure so that it is centered, right aligned, or left aligned, or aligned according to the line direction. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`TETextBox` (page 3000) Deprecated in Mac OS X v10.4
Draws the indicated text in a given rectangle, with the specified alignment. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`TEUpdate` (page 3002) Deprecated in Mac OS X v10.4
Draws the specified text within a given update rectangle. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

Initializing TextEdit, Creating an Edit Structure, and Disposing of an Edit Structure

`TEDispose` (page 2975) Deprecated in Mac OS X v10.4
Removes a specified edit structure and releases all memory associated with it. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`TENew` (page 2986) Deprecated in Mac OS X v10.4
Creates and initializes a monostyled edit structure and allocates a handle to it. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`TEStyleNew` (page 2999) Deprecated in Mac OS X v10.4
Creates a multistyled edit structure and allocates a handle to it. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

Managing the TextEdit Private Scrap

`TEGetScrapHandle` (page 2980) Deprecated in Mac OS X v10.4
Returns a handle to the TextEdit private scrap. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`TEGetScrapLength` (page 2981) Deprecated in Mac OS X v10.4
Returns the size of the TextEdit private scrap, in bytes. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`TEScrapHandle` (page 2990) Deprecated in Mac OS X v10.4
Returns a handle to the TextEdit private scrap. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`TESetScrapHandle` (page 2994) Deprecated in Mac OS X v10.4
Sets a handle to the TextEdit private scrap. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`TESetScrapLength` (page 2994) Deprecated in Mac OS X v10.4
Sets the size of the TextEdit private scrap to the specified number of bytes. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

Modifying the Text of an Edit Structure

`TECopy` (page 2971) Deprecated in Mac OS X v10.4

Copies the text selection range from the edit structure, leaving the selection range intact. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`TECut` (page 2973) Deprecated in Mac OS X v10.4

Removes the current selection range from the text of the designated edit structure, redrawing the text as necessary. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`TEDelete` (page 2974) Deprecated in Mac OS X v10.4

Removes the selected range of text from the designated edit structure, redrawing the remaining text as necessary. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`TEFromScrap` (page 2976) Deprecated in Mac OS X v10.4

Copies the contents of the desk scrap to the TextEdit private scrap. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`TEInsert` (page 2985) Deprecated in Mac OS X v10.4

Inserts the specified text immediately before the selection range or the insertion point in the text of the designated edit structure, redrawing the text as necessary. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`TEPaste` (page 2988) Deprecated in Mac OS X v10.4

Replaces the edit structure's selected text with the contents of the private scrap and leaves an insertion point after the inserted text. If the selection range is an insertion point, `TEPaste` inserts the contents of the private scrap there. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`TEStylePaste` (page 3000) Deprecated in Mac OS X v10.4

Pastes text and its associated character attribute information from the desk scrap into the edit structure's text at the insertion point—if the current selection range is an insertion point—or it replaces the current selection range. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`TEToScrap` (page 3001) Deprecated in Mac OS X v10.4

Copies the contents of the TextEdit private scrap to the desk scrap. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

Setting and Getting an Edit Structure's Text and Character Attribute Information

`TEGetStyleHandle` (page 2982) Deprecated in Mac OS X v10.4

Returns the style handle stored in the designated edit structure's `txFont` and `txFace` fields. The style handle points to the associated style structure, not to a copy of it. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`TEGetText` (page 2983) Deprecated in Mac OS X v10.4

Returns a handle to the text of the specified edit structure. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`TEKey` (page 2985) Deprecated in Mac OS X v10.4
Replaces the selection range in the text of the specified edit structure with the input character and positions the insertion point just past the inserted character. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`TESetStyleHandle` (page 2997) Deprecated in Mac OS X v10.4
Sets an edit structure's style handle, which is stored in the `txFont` and `txFace` fields. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`TESetText` (page 2997) Deprecated in Mac OS X v10.4
Incorporates a copy of the specified text into the designated edit structure. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

Setting the Caret and Selection Range

`TEClick` (page 2969) Deprecated in Mac OS X v10.4
Controls placement and highlighting of the selection range as determined by mouse events. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`TEGetHiliteRgn` (page 2978) Deprecated in Mac OS X v10.4
Obtains the highlight region for the specified edit structure. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`TEIdle` (page 2984) Deprecated in Mac OS X v10.4
When called repeatedly, displays a blinking caret at the insertion point, if any exists, in the text of the specified edit structure of an active window. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`TESetSelect` (page 2995) Deprecated in Mac OS X v10.4
Sets the selection range (or denotes the insertion point) within the text of the specified edit structure. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

Using Byte Offsets and Corresponding Points

`TEGetOffset` (page 2979) Deprecated in Mac OS X v10.4
Finds the byte offset of a character in an edit structure's text that corresponds to the specified point. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`TEGetPoint` (page 2979) Deprecated in Mac OS X v10.4
Determines the point that corresponds to the specified byte offset of a character and returns the coordinates of that point. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

Handling TSM Dialogs

`GetTSMTEDialogDocumentID` (page 2954) Deprecated in Mac OS X v10.4
Returns a TSM document ID for the specified dialog. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`GetTSMTEDialogTSMTERecHandle` (page 2954) Deprecated in Mac OS X v10.4
Returns a handle to a TSM record for the specified dialog. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`IsTSMTEDialog` (page 2961) Deprecated in Mac OS X v10.4
Checks to see if the specified dialog is a TSMTE dialog. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`SetTSMTEDialogDocumentID` (page 2967) Deprecated in Mac OS X v10.4
Sets the document ID for the specified dialog. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`SetTSMTEDialogTSMTERecHandle` (page 2967) Deprecated in Mac OS X v10.4
Sets a handle to a TSMTE record for the specified dialog. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

Working With UPPs for TextEdit Callback Functions

`DisposeCaretHookUPP` (page 2950) Deprecated in Mac OS X v10.4
Disposes of a universal procedure pointer (UPP) to a caret-hook callback. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`DisposeDrawHookUPP` (page 2950) Deprecated in Mac OS X v10.4
Disposes of a universal procedure pointer (UPP) to a draw-hook callback. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`DisposeEOLHookUPP` (page 2950) Deprecated in Mac OS X v10.4
Disposes of a universal procedure pointer (UPP) to an EOL-hook callback. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`DisposeHighHookUPP` (page 2951) Deprecated in Mac OS X v10.4
Disposes of a universal procedure pointer (UPP) to a high-hook callback. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`DisposeHitTestHookUPP` (page 2951) Deprecated in Mac OS X v10.4
Disposes of a universal procedure pointer (UPP) to a hit-test hook callback. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`DisposeNWidthHookUPP` (page 2951) Deprecated in Mac OS X v10.4
Disposes of a universal procedure pointer (UPP) to a width-hook callback. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`DisposeTEClickLoopUPP` (page 2952) Deprecated in Mac OS X v10.4
Disposes of a universal procedure pointer (UPP) to a click-loop callback. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`DisposeTEDoTextUPP` (page 2952) Deprecated in Mac OS X v10.4
Disposes of a universal procedure pointer (UPP) to a do-text callback. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`DisposeTEFindWordUPP` (page 2952) Deprecated in Mac OS X v10.4
Disposes of a universal procedure pointer (UPP) to a find-word callback. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`DisposeTERecalcUPP` (page 2952) Deprecated in Mac OS X v10.4
Disposes of a universal procedure pointer (UPP) to a recaluclation callback. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`DisposeTextWidthHookUPP` (page 2953) Deprecated in Mac OS X v10.4
Disposes of a universal procedure pointer (UPP) to a text-width-hook callback. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`DisposeTSMTEPostUpdateUPP` (page 2953) Deprecated in Mac OS X v10.4

Disposes of a universal procedure pointer (UPP) to a post-update callback. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`DisposeTSMTEPreUpdateUPP` (page 2953) Deprecated in Mac OS X v10.4

Disposes of a universal procedure pointer (UPP) to a pre-update callback. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`DisposeWidthHookUPP` (page 2954) Deprecated in Mac OS X v10.4

Disposes of a universal procedure pointer (UPP) to a width-hook callback. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`InvokeCaretHookUPP` (page 2955) Deprecated in Mac OS X v10.4

Calls a caret-hook callback. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`InvokeDrawHookUPP` (page 2955) Deprecated in Mac OS X v10.4

Calls a draw-hook callback. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`InvokeEOLHookUPP` (page 2956) Deprecated in Mac OS X v10.4

Calls an EOL-hook callback. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`InvokeHighHookUPP` (page 2956) Deprecated in Mac OS X v10.4

Calls a high-hook callback. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`InvokeHitTestHookUPP` (page 2957) Deprecated in Mac OS X v10.4

Calls a hit-test hook callback. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`InvokeNWidthHookUPP` (page 2957) Deprecated in Mac OS X v10.4

Calls a width-hook callback. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`InvokeTEClickLoopUPP` (page 2958) Deprecated in Mac OS X v10.4

Calls a click-loop callback. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`InvokeTEDoTextUPP` (page 2958) Deprecated in Mac OS X v10.4

Calls a do-text callback. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`InvokeTEFindWordUPP` (page 2959) Deprecated in Mac OS X v10.4

Calls a find-word callback. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`InvokeTERecalcUPP` (page 2959) Deprecated in Mac OS X v10.4

Calls a recalculation callback. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`InvokeTextWidthHookUPP` (page 2959) Deprecated in Mac OS X v10.4

Calls a text-width-hook callback. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`InvokeTSMTEPostUpdateUPP` (page 2960) Deprecated in Mac OS X v10.4

Calls a post-update callback. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`InvokeTSMTEPreUpdateUPP` (page 2960) Deprecated in Mac OS X v10.4

Calls a pre-update callback. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`InvokeWidthHookUPP` (page 2961) Deprecated in Mac OS X v10.4

Calls a width-hook callback. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`NewCaretHookUPP` (page 2962) Deprecated in Mac OS X v10.4

Creates a new universal procedure pointer (UPP) to a caret-hook callback. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`NewDrawHookUPP` (page 2962) Deprecated in Mac OS X v10.4

Creates a new universal procedure pointer (UPP) to a draw-hook callback. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`NewEOLHookUPP` (page 2962) Deprecated in Mac OS X v10.4

Creates a new universal procedure pointer (UPP) to an EOL-hook callback. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`NewHighHookUPP` (page 2963) Deprecated in Mac OS X v10.4

Creates a new universal procedure pointer (UPP) to a high-hook callback. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`NewHitTestHookUPP` (page 2963) Deprecated in Mac OS X v10.4

Creates a new universal procedure pointer (UPP) to a hit-test hook callback. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`NewNWidthHookUPP` (page 2964) Deprecated in Mac OS X v10.4

Creates a new universal procedure pointer (UPP) to a width-hook callback. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`NewTEClickLoopUPP` (page 2964) Deprecated in Mac OS X v10.4

Creates a new universal procedure pointer (UPP) to a click-loop callback. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`NewTEDoTextUPP` (page 2964) Deprecated in Mac OS X v10.4

Creates a new universal procedure pointer (UPP) to a do-text callback. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`NewTEFindWordUPP` (page 2965) Deprecated in Mac OS X v10.4

Creates a new universal procedure pointer (UPP) to a find-word callback. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`NewTERecalcUPP` (page 2965) Deprecated in Mac OS X v10.4

Creates a new universal procedure pointer (UPP) to a recalculation callback. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`NewTextWidthHookUPP` (page 2965) Deprecated in Mac OS X v10.4

Creates a new universal procedure pointer (UPP) to a text-width-hook callback. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`NewTSMTEPostUpdateUPP` (page 2966) Deprecated in Mac OS X v10.4

Creates a new universal procedure pointer (UPP) to a post-update callback. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`NewTSMTEPreUpdateUPP` (page 2966) Deprecated in Mac OS X v10.4

Creates a new universal procedure pointer (UPP) to a pre-update callback. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

`NewWidthHookUPP` (page 2966) Deprecated in Mac OS X v10.4
Creates a new universal procedure pointer (UPP) to a width-hook callback. (Deprecated. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

Functions

DisposeCaretHookUPP

Disposes of a universal procedure pointer (UPP) to a caret-hook callback. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void DisposeCaretHookUPP (
   CaretHookUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`TextEdit.h`

DisposeDrawHookUPP

Disposes of a universal procedure pointer (UPP) to a draw-hook callback. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void DisposeDrawHookUPP (
   DrawHookUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`TextEdit.h`

DisposeEOLHookUPP

Disposes of a universal procedure pointer (UPP) to an EOL-hook callback. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void DisposeEOLHookUPP (
   EOLHookUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
TextEdit.h

DisposeHighHookUPP

Disposes of a universal procedure pointer (UPP) to a high-hook callback. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void DisposeHighHookUPP (
   HighHookUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
TextEdit.h

DisposeHitTestHookUPP

Disposes of a universal procedure pointer (UPP) to a hit-test hook callback. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void DisposeHitTestHookUPP (
   HitTestHookUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
TextEdit.h

DisposeNWidthHookUPP

Disposes of a universal procedure pointer (UPP) to a width-hook callback. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void DisposeNWidthHookUPP (
   NWidthHookUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
TextEdit.h

DisposeTEClickLoopUPP

Disposes of a universal procedure pointer (UPP) to a click-loop callback. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void DisposeTEClickLoopUPP (
   TEClickLoopUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`TextEdit.h`

DisposeTEDoTextUPP

Disposes of a universal procedure pointer (UPP) to a do-text callback. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void DisposeTEDoTextUPP (
   TEDoTextUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`TextEdit.h`

DisposeTEFindWordUPP

Disposes of a universal procedure pointer (UPP) to a find-word callback. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void DisposeTEFindWordUPP (
   TEFindWordUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`TextEdit.h`

DisposeTERecalcUPP

Disposes of a universal procedure pointer (UPP) to a recaluclation callback. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void DisposeTERecalcUPP (
   TERecalcUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
TextEdit.h

DisposeTextWidthHookUPP

Disposes of a universal procedure pointer (UPP) to a text-width-hook callback. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void DisposeTextWidthHookUPP (
   TextWidthHookUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
TextEdit.h

DisposeTSMTEPostUpdateUPP

Disposes of a universal procedure pointer (UPP) to a post-update callback. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void DisposeTSMTEPostUpdateUPP (
   TSMTEPostUpdateUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
TSMTE.h

DisposeTSMTEPreUpdateUPP

Disposes of a universal procedure pointer (UPP) to a pre-update callback. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void DisposeTSMTEPreUpdateUPP (
   TSMTEPreUpdateUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`TSMTE.h`

DisposeWidthHookUPP

Disposes of a universal procedure pointer (UPP) to a width-hook callback. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void DisposeWidthHookUPP (
   WidthHookUPP userUPP
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`TextEdit.h`

GetTSMTEDialogDocumentID

Returns a TSM document ID for the specified dialog. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
TSMDocumentID GetTSMTEDialogDocumentID (
   DialogRef dialog
);
```

Return Value
See the Text Services Manager documentation for a description of the `TSMDocumentID` data type.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TSMTE.h`

GetTSMTEDialogTSMTERecHandle

Returns a handle to a TSM record for the specified dialog. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
TSMTERecHandle GetTSMTEDialogTSMTERecHandle (
   DialogRef dialog
);
```

Return Value
See the description of the `TSMTERecHandle` data type.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TSMTE.h`

InvokeCaretHookUPP

Calls a caret-hook callback. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void InvokeCaretHookUPP (
   const Rect *r,
   TEPtr pTE,
   CaretHookUPP userUPP
);
```

Discussion
You should not need to use this function, as the system invokes your callback for you.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`TextEdit.h`

InvokeDrawHookUPP

Calls a draw-hook callback. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void InvokeDrawHookUPP (
   unsigned short textOffset,
   unsigned short drawLen,
   void *textBufferPtr,
   TEPtr pTE,
   TEHandle hTE,
   DrawHookUPP userUPP
);
```

Discussion
You should not need to use this function, as the system invokes your callback for you.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`TextEdit.h`

InvokeEOLHookUPP

Calls an EOL-hook callback. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
Boolean InvokeEOLHookUPP (
   char theChar,
   TEPtr pTE,
   TEHandle hTE,
   EOLHookUPP userUPP
);
```

Return Value
See the Mac Types documentation for a description of the `Boolean` data type.

Discussion
You should not need to use this function, as the system invokes your callback for you.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`TextEdit.h`

InvokeHighHookUPP

Calls a high-hook callback. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void InvokeHighHookUPP (
   const Rect *r,
   TEPtr pTE,
   HighHookUPP userUPP
);
```

Discussion
You should not need to use this function, as the system invokes your callback for you.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`TextEdit.h`

InvokeHitTestHookUPP

Calls a hit-test hook callback. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
Boolean InvokeHitTestHookUPP (
   unsigned short styleRunLen,
   unsigned short styleRunOffset,
   unsigned short slop,
   void *textBufferPtr,
   TEPtr pTE,
   TEHandle hTE,
   unsigned short *pixelWidth,
   unsigned short *charOffset,
   Boolean *pixelInChar,
   HitTestHookUPP userUPP
);
```

Return Value
See the Mac Types documentation for a description of the `Boolean` data type.

Discussion
You should not need to use this function, as the system invokes your callback for you.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`TextEdit.h`

InvokeNWidthHookUPP

Calls a width-hook callback. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
unsigned short InvokeNWidthHookUPP (
   unsigned short styleRunLen,
   unsigned short styleRunOffset,
   short slop,
   short direction,
   void *textBufferPtr,
   short *lineStart,
   TEPtr pTE,
   TEHandle hTE,
   NWidthHookUPP userUPP
);
```

Discussion
You should not need to use this function, as the system invokes your callback for you.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`TextEdit.h`

InvokeTEClickLoopUPP

Calls a click-loop callback. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
Boolean InvokeTEClickLoopUPP (
   TEPtr pTE,
   TEClickLoopUPP userUPP
);
```

Return Value
See the Mac Types documentation for a description of the `Boolean` data type.

Discussion
You should not need to use this function, as the system invokes your callback for you.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`TextEdit.h`

InvokeTEDoTextUPP

Calls a do-text callback. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void InvokeTEDoTextUPP (
   TEPtr pTE,
   unsigned short firstChar,
   unsigned short lastChar,
   short selector,
   GrafPtr *currentGrafPort,
   short *charPosition,
   TEDoTextUPP userUPP
);
```

Discussion
You should not need to use this function, as the system invokes your callback for you.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`TextEdit.h`

InvokeTEFindWordUPP

Calls a find-word callback. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void InvokeTEFindWordUPP (
   unsigned short currentPos,
   short caller,
   TEPtr pTE,
   TEHandle hTE,
   unsigned short *wordStart,
   unsigned short *wordEnd,
   TEFindWordUPP userUPP
);
```

Discussion
You should not need to use this function, as the system invokes your callback for you.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
TextEdit.h

InvokeTERecalcUPP

Calls a recalculation callback. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void InvokeTERecalcUPP (
   TEPtr pTE,
   unsigned short changeLength,
   unsigned short *lineStart,
   unsigned short *firstChar,
   unsigned short *lastChar,
   TERecalcUPP userUPP
);
```

Discussion
You should not need to use this function, as the system invokes your callback for you.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
TextEdit.h

InvokeTextWidthHookUPP

Calls a text-width-hook callback. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
unsigned short InvokeTextWidthHookUPP (
   unsigned short textLen,
   unsigned short textOffset,
   void *textBufferPtr,
   TEPtr pTE,
   TEHandle hTE,
   TextWidthHookUPP userUPP
);
```

Discussion
You should not need to use this function, as the system invokes your callback for you.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`TextEdit.h`

InvokeTSMTEPostUpdateUPP

Calls a post-update callback. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void InvokeTSMTEPostUpdateUPP (
   TEHandle textH,
   long fixLen,
   long inputAreaStart,
   long inputAreaEnd,
   long pinStart,
   long pinEnd,
   long refCon,
   TSMTEPostUpdateUPP userUPP
);
```

Discussion
You should not need to use this function, as the system invokes your callback for you.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`TSMTE.h`

InvokeTSMTEPreUpdateUPP

Calls a pre-update callback. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void InvokeTSMTEPreUpdateUPP (
   TEHandle textH,
   long refCon,
   TSMTEPreUpdateUPP userUPP
);
```

Discussion
You should not need to use this function, as the system invokes your callback for you.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`TSMTE.h`

InvokeWidthHookUPP

Calls a width-hook callback. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
unsigned short InvokeWidthHookUPP (
   unsigned short textLen,
   unsigned short textOffset,
   void *textBufferPtr,
   TEPtr pTE,
   TEHandle hTE,
   WidthHookUPP userUPP
);
```

Discussion
You should not need to use this function, as the system invokes your callback for you.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`TextEdit.h`

IsTSMTEDialog

Checks to see if the specified dialog is a TSMTE dialog. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
Boolean IsTSMTEDialog (
   DialogRef dialog
);
```

Return Value
See the Mac Types documentation for a description of the `Boolean` data type.

Availability
Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TSMTE.h`

NewCaretHookUPP

Creates a new universal procedure pointer (UPP) to a caret-hook callback. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
CaretHookUPP NewCaretHookUPP (
   CaretHookProcPtr userRoutine
);
```

Return Value
See the description of the `CaretHookUPP` data type.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`TextEdit.h`

NewDrawHookUPP

Creates a new universal procedure pointer (UPP) to a draw-hook callback. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
DrawHookUPP NewDrawHookUPP (
   DrawHookProcPtr userRoutine
);
```

Return Value
See the description of the `DrawHookUPP` data type.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`TextEdit.h`

NewEOLHookUPP

Creates a new universal procedure pointer (UPP) to an EOL-hook callback. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
EOLHookUPP NewEOLHookUPP (
   EOLHookProcPtr userRoutine
);
```

Return Value
See the description of the `EOLHookUPP` data type.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`TextEdit.h`

NewHighHookUPP

Creates a new universal procedure pointer (UPP) to a high-hook callback. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
HighHookUPP NewHighHookUPP (
   HighHookProcPtr userRoutine
);
```

Return Value
See the description of the `HighHookUPP` data type.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`TextEdit.h`

NewHitTestHookUPP

Creates a new universal procedure pointer (UPP) to a hit-test hook callback. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
HitTestHookUPP NewHitTestHookUPP (
   HitTestHookProcPtr userRoutine
);
```

Return Value
See the description of the `HitTestHookUPP` data type.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`TextEdit.h`

NewNWidthHookUPP

Creates a new universal procedure pointer (UPP) to a width-hook callback. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
NWidthHookUPP NewNWidthHookUPP (
   NWidthHookProcPtr userRoutine
);
```

Return Value
See the description of the `NWidthHookUPP` data type.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`TextEdit.h`

NewTEClickLoopUPP

Creates a new universal procedure pointer (UPP) to a click-loop callback. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
TEClickLoopUPP NewTEClickLoopUPP (
   TEClickLoopProcPtr userRoutine
);
```

Return Value
See the description of the `TEClickLoopUPP` data type.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`TextEdit.h`

NewTEDoTextUPP

Creates a new universal procedure pointer (UPP) to a do-text callback. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
TEDoTextUPP NewTEDoTextUPP (
   TEDoTextProcPtr userRoutine
);
```

Return Value
See the description of the `TEDoTextUPP` data type.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
TextEdit.h

NewTEFindWordUPP

Creates a new universal procedure pointer (UPP) to a find-word callback. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
TEFindWordUPP NewTEFindWordUPP (
   TEFindWordProcPtr userRoutine
);
```

Return Value
See TEFindWordUPP (page 3020) for a description of the TEFindWordUPP data type.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
TextEdit.h

NewTERecalcUPP

Creates a new universal procedure pointer (UPP) to a recalculation callback. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
TERecalcUPP NewTERecalcUPP (
   TERecalcProcPtr userRoutine
);
```

Return Value
See the description of the TERecalcUPP data type.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
TextEdit.h

NewTextWidthHookUPP

Creates a new universal procedure pointer (UPP) to a text-width-hook callback. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
TextWidthHookUPP NewTextWidthHookUPP (
   TextWidthHookProcPtr userRoutine
);
```

Return Value
See the description of the TextWidthHookUPP data type.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`TextEdit.h`

NewTSMTEPostUpdateUPP

Creates a new universal procedure pointer (UPP) to a post-update callback. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
TSMTEPostUpdateUPP NewTSMTEPostUpdateUPP (
   TSMTEPostUpdateProcPtr userRoutine
);
```

Return Value
See `TSMTEPostUpdateUPP` (page 3029) for a description of the `TSMTEPostUpdateUPP` data type.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`TSMTE.h`

NewTSMTEPreUpdateUPP

Creates a new universal procedure pointer (UPP) to a pre-update callback. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
TSMTEPreUpdateUPP NewTSMTEPreUpdateUPP (
   TSMTEPreUpdateProcPtr userRoutine
);
```

Return Value
See the description of the `TSMTEPreUpdateUPP` data type.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`TSMTE.h`

NewWidthHookUPP

Creates a new universal procedure pointer (UPP) to a width-hook callback. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
WidthHookUPP NewWidthHookUPP (
   WidthHookProcPtr userRoutine
);
```

Return Value
See the description of the `WidthHookUPP` data type.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.

Declared In
`TextEdit.h`

SetTSMTEDialogDocumentID

Sets the document ID for the specified dialog. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void SetTSMTEDialogDocumentID (
   DialogRef dialog,
   TSMDocumentID documentID
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TSMTE.h`

SetTSMTEDialogTSMTERecHandle

Sets a handle to a TSMTE record for the specified dialog. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void SetTSMTEDialogTSMTERecHandle (
   DialogRef dialog,
   TSMTERecHandle tsmteRecHandle
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TSMTE.h`

TEActivate

Activates the specified edit structure. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void TEActivate (
   TEHandle hTE
);
```

Parameters

hTE

A handle to the specified edit structure.

Discussion

When your application receives notification of an activate event, it can call the `TEActivate` function, which activates an edit structure and highlights the selection range. If the selection range is an insertion point, `TEActivate` simply displays a caret there. Call this function every time the Event Manager function `WaitNextEvent` reports that the window containing the edit structure has become active.

If you do not call `TEActivate` before you call `TEClick`, `TEIdle`, or `TESetSelect`, the selection range is not highlighted, or, if the selection range is set to an insertion point, a caret is not displayed at the insertion point. However, if you have turned on outline highlighting through the `TEFeatureFlag` function for the edit structure, the text of the selection range is framed or a dimmed or an unblinking caret is displayed at the insertion point.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`TextEdit.h`

TEAutoView

Enables and disables automatic scrolling of the text in the specified edit structure. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void TEAutoView (
   Boolean fAuto,
   TEHandle hTE
);
```

Parameters

fAuto

A flag indicating whether to enable or disable automatic scrolling. A value of `TRUE` enables automatic scrolling; a value of `FALSE` disables automatic scrolling.

hTE

A handle to the edit structure for which automatic scrolling is to be enabled or disabled.

Discussion

The `TEAutoView` function does not actually scroll the text automatically: `TESelView` does. However, when `fAuto` is set to `FALSE` , a call to `TESelView` has no effect.

If there is a scroll bar associated with the edit structure, your application must manage scrolling of it. You can replace the default click loop function, which scrolls the text only, with a customized version that also updates the scroll bar.

You can also enable or disable automatic scrolling for an edit structure through the `teFAutoScroll` feature of the `TEFeatureFlag` function. For more information, see `TEFeatureFlag` (page 2976).

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextEdit.h`

TECalText

Recalculates the beginnings of all lines of text in the specified edit structure. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void TECalText (
   TEHandle hTE
);
```

Parameters

hTE

A handle to the edit structure whose text lines are to be recalculated.

Discussion
The `TECalText` function updates elements of the `lineStarts` array in an edit structure. Call `TECalText` if you've changed the destination rectangle, the `hText` field, or any other property of the edit structure that pertains to line breaks and the number of characters per line—for example, font, size, style, and so on.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextEdit.h`

TEClick

Controls placement and highlighting of the selection range as determined by mouse events. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void TEClick (
   Point pt,
   Boolean fExtend,
   TEHandle h
);
```

Parameters

pt

The mouse location in local coordinates at the time the mouse button was pressed, obtainable from the event structure (in global coordinates).

fExtend

A flag denoting the state of the Shift key at the time of the click as indicated by the Event Manager. If the Shift key was held down at the time of the click to extend the selection, pass a value of `TRUE`.

h

A handle to the edit structure whose text is displayed in the view rectangle where the click occurred.

Discussion

Call `TEClick` whenever a mouse-down event occurs in the view rectangle of the edit structure and the window associated with that edit structure is active. The `TEClick` function keeps control until the mouse button is released. Use the QuickDraw function `GlobalToLocal` to convert the global coordinates of the mouse location given in the event structure to the local coordinate system for `pt`.

The `TEClick` function removes highlighting of the old selection range unless the selection range is being extended. If the mouse moves, meaning that a drag is occurring, `TEClick` expands or shortens the selection range accordingly a character at a time. In the case of a double-click, the word where the cursor is positioned becomes the selection range.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`TextEdit.h`

TEContinuousStyle

Determines whether a given character attribute is continuous over the current selection range. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
Boolean TEContinuousStyle (
   short *mode,
   TextStyle *aStyle,
   TEHandle hTE
);
```

Parameters

mode

On input, a pointer to a selector specifying the attributes to be checked. On output, `mode` identifies only those attributes determined to be continuous over the selection range. Possible values for the `mode` parameter are defined in "Text Styling Constants" (page 3036).

aStyle

On input, a pointer to a text style structure. On output, this structure contains the values for the `mode` attributes determined to be continuous over the selection.

hTE

A handle to the edit structure containing the selected text whose attributes are to be checked. If the value of `hTE` is a handle to a monostyled edit structure, `TEContinuousStyle` returns the set of character attributes that are consistent for the entire structure.

Return Value
`TRUE` if all of the attributes to be checked are continuous; `FALSE` if none or some are continuous. See the Mac Types documentation for a description of the `Boolean` data type.

Discussion
This function does not modify the text selection. If the current selection range is an insertion point, `TEContinuousStyle` first checks the null scrap. If the null scrap contains character attributes, then they are used based on the value of the `mode` parameter. Otherwise, if the null scrap is empty, `TEContinuousStyle` returns the attributes of the character preceding the insertion point. The `TEContinuousStyle` function always returns `TRUE` in this case, and each field of the text style structure is set if the corresponding bit in the `mode` parameter is set.

Note that fields in the text style structure specified by `aStyle` are only valid if the corresponding bits are set in the `mode` variable.

How the `tsFace` field of the `aStyle` structure is used requires some consideration. For example, if `TEContinuousStyle` returns a `mode` parameter that contains `doFace` and the text style structure `tsFace` field is bold, it means that the selected text is all bold, but may contain other text styles, such as italic, as well. Italic does not apply to all of the selected text, or it would have been included in the `tsFace` field. If the `tsFace` field is an empty set, then all of the selected text is plain.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextEdit.h`

TECopy

Copies the text selection range from the edit structure, leaving the selection range intact. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void TECopy (
   TEHandle hTE
);
```

Parameters

hTE

A handle to the edit structure containing the text to be copied.

Discussion
The `TECopy` function copies the text to the private scrap. For text of a monostyled edit structure, the text is written to the private scrap only. For text of a multistyled edit structure, the text is written to the TextEdit private scrap, the character attribute information is written to the TextEdit style scrap, and both are written to the Scrap Manager's desk scrap. Anything previously in the private scrap is deleted before the copied text is written to it.

For both multistyled and monostyled text, if the selection range is an insertion point, `TECopy` empties the TextEdit private scrap. When the selection range is an insertion point and the text is multistyled, `TECopy` has no effect on the null scrap, the style scrap, or the Scrap Manager's desk scrap.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextEdit.h`

TECustomHook

Replaces a default TextEdit hook function with a customized function and returns the address of the replaced function. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void TECustomHook (
   TEIntHook which,
   UniversalProcPtr *addr,
   TEHandle hTE
);
```

Parameters

which
The hook whose default function is to be replaced.

addr
On input, the address of your customized function.
On output, the `addr` parameter contains the address of the function that was previously installed in the field identified by the `which` parameter. This address is returned so that you can daisy-chain functions.

hTE
A handle to the edit structure to be modified.

Discussion
The `TECustomHook` function lets you alter the behavior of TextEdit to better suit your application's requirements and those of the script systems installed. If you replace a default hook function with a customized version that you write in a high-level language, such as Pascal or C, you need to provide assembly-language glue code that utilizes the registers for your high-level language function.

The end-of-line hook, width measurement hook, new width measurement hook, text width measurement hook, draw hook, and hit test hook fields are hook fields in the TextEdit dispatch structure. The `which` parameter identifies the hook whose default function is to be replaced. You use the constants described in "Text Custom Hook Constants" (page 3033) to specify a value for this parameter.

Certain precautions are critical in replacing default functions. Before placing the address of your function in the TextEdit dispatch structure, strip the addresses, using the Operating System Utilities `StripAddress` function, to guarantee that your application is 32-bit clean.

Before replacing a TextEdit function with a customized one, determine whether more than one script system is installed, and if so, ensure that your customized function accommodates all of the installed script systems. This avoids the problem of your customized function producing results that are incompatible with the Script Manager.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextEdit.h`

TECut

Removes the current selection range from the text of the designated edit structure, redrawing the text as necessary. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void TECut (
   TEHandle hTE
);
```

Parameters

hTE
A handle to the edit structure containing the text to be cut.

Discussion
For monostyled text, the `TECut` function writes the cut text to the private scrap.

For multistyled text, `TECut` writes the cut text to the private scrap and its character attributes to the style scrap it also writes both to the Scrap Manager's desk scrap. For multistyled text, the `TECut` function removes the character attributes from the style structure's style table when the text is cut.

For both monostyled and multistyled text, if the selection range is an insertion point, TextEdit deletes everything from the private scrap. When the selection range is an insertion point and the text is multistyled, `TECut` has no effect on the style scrap or the Scrap Manager's desk scrap.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextEdit.h`

TEDeactivate

Deactivates the specified edit structure. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void TEDeactivate (
   TEHandle hTE
);
```

Parameters

hTE

A handle to the specified edit structure.

Discussion

When the activate event flag is set to deactivate the window, your application can call the `TEDeactivate` function, which changes an edit structure's status from active to inactive and removes the selection range highlighting or the caret.

However, if you turned on outline highlighting through the `TEFeatureFlag` function for the edit structure, the text of the selection range is framed or a dimmed or an unblinking caret is displayed at the insertion point when the structure is deactivated.

Call this function every time the Event Manager function `WaitNextEvent` reports that the window containing the edit structure has become inactive.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`TextEdit.h`

TEDelete

Removes the selected range of text from the designated edit structure, redrawing the remaining text as necessary. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void TEDelete (
   TEHandle hTE
);
```

Parameters

hTE

A handle to the edit structure containing the text to be deleted.

Discussion

When the `TEDelete` function deletes a selected range of text, it does not transfer the text to either the private scrap or the Scrap Manager's desk scrap.

For multistyled structures, when you use `TEDelete` to delete a selected range of text, the associated character attributes are saved in the null scrap to be applied to characters entered after the text is deleted. When the user clicks in some other area of the text, the character attributes are removed from the null scrap. You can use `TEDelete` to implement the Clear command. The `TEDelete` function recalculates line starts and line heights.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextEdit.h`

TEDispose

Removes a specified edit structure and releases all memory associated with it. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void TEDispose (
   TEHandle hTE
);
```

Parameters

hTE

A handle to the edit structure for which the allocated memory should be released.

Discussion
Call the `TEDispose` function only when you're completely through with an edit structure.

Note that if your program retains a handle to text associated with the edit structure that you are destroying with `TEDispose`, the handle becomes invalid because the `TEDispose` function disposes of it, as well as the dispatch structure handle. If the structure is multistyled, `TEDispose` also disposes all of the style-related handles: `STHandle`, `LHHandle`, `STScrpHandle`, `nullSTHandle`, and `TEStyleHandle`.

To continue to refer to the text after you've destroyed the edit structure, you need to make a copy of the handle in the `hText` field of the edit structure using the Operating System Utilities `HandToHand` function before you call `TEDispose`.

In addition to disposing of the edit structure, the edit structure handle, and the dispatch structure handle, the `TEDispose` function destroys the null scrap associated with the edit structure and releases the memory used for it.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextEdit.h`

TEFeatureFlag

Turns a specified feature on or off or returns the current status of that feature. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
short TEFeatureFlag (
   short feature,
   short action,
   TEHandle hTE
);
```

Parameters

feature

The feature for which the action is to be performed. See "Text Feature Constants" (page 3035) for a description of the available values.

action

A selector stipulating that the feature, specified by the `feature` parameter, is to be turned on or off, or that the current status of the feature is to be returned. See "Text Feature Action Constants" (page 3034) for a description of the available values.

hTE

A handle to the edit structure for which the action should be performed.

Return Value

The status of the specified feature (if the selector is set to `teBitTest`).

Discussion

You can use the `TEFeatureFlag` function to check the status of additional TextEdit features—automatic scrolling, outline highlighting, and text buffering—and to enable or disable the feature. You can also use this function to disable inline input in a particular edit structure and to enable several features that have been provided so that inline input works correctly with TextEdit.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`TextEdit.h`

TEFromScrap

Copies the contents of the desk scrap to the TextEdit private scrap. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
OSErr TEFromScrap (
   void
);
```

Return Value

A result code. See "TextEdit Result Codes" (page 3037).

Discussion

You use this function to move monostyled text across applications or between an application and a desk accessory.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextEdit.h`

TEGetDoTextHook

Obtains a universal procedure pointer to your do-text-hook callback. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
TEDoTextUPP TEGetDoTextHook (
   void
);
```

Return Value
See the description of the `TEDoTextUPP` data type.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextEdit.h`

TEGetFindWordHook

Obtains a universal procedure pointer to your set-find-word-hook callback. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
TEFindWordUPP TEGetFindWordHook (
   void
);
```

Return Value
See the description of the `TEFindWordUPP` data type.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextEdit.h`

TEGetHeight

Returns the total height of all of the lines in the text between and including the specified starting and ending lines. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
long TEGetHeight (
   long endLine,
   long startLine,
   TEHandle hTE
);
```

Parameters

endLine

The number of the last line of text whose height is to be included in the total height. You can specify a value that is greater than or equal to 1 for this parameter.

startLine

The number of the first line of text whose height is to be included in the total height. You can specify a value that is greater than or equal to 1 for this parameter.

hTE

A handle to the edit structure containing the lines of text whose height is to be returned.

Return Value

The total height of all of the designated text lines.

Discussion

For monostyled text, the `TEGetHeight` function uses the value of the edit structure's `lineHeight` field. For multistyled text, it uses the line height element (`LHElement`) of the line height table (`LHTable`). Note that `TEGetHeight` does not take into account the height of any blank lines at the end of the text. You need to consider this when scrolling text.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`TextEdit.h`

TEGetHiliteRgn

Obtains the highlight region for the specified edit structure. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
OSErr TEGetHiliteRgn (
   RgnHandle region,
   TEHandle hTE
);
```

Return Value

A result code. See "TextEdit Result Codes" (page 3037).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextEdit.h`

TEGetOffset

Finds the byte offset of a character in an edit structure's text that corresponds to the specified point. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
short TEGetOffset (
   Point pt,
   TEHandle hTE
);
```

Parameters

pt

A point in the displayed text of the specified edit structure.

hTE

A handle to the edit structure containing the text.

Return Value
The byte offset of the character at the specified point. In the case of a 2-byte character, the function returns the byte offset of the first byte.

Discussion
The `TEGetOffset` function works for both monostyled and multistyled edit structures.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextEdit.h`

TEGetPoint

Determines the point that corresponds to the specified byte offset of a character and returns the coordinates of that point. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
Point TEGetPoint (
   short offset,
   TEHandle hTE
);
```

Parameters

offset

A byte offset into the text buffer of an edit structure.

hTE

A handle to the edit structure containing the text.

Return Value
The coordinates of the point that corresponds to the specified byte offset. The `TEGetPoint` function returns a valid result even when the edit structure does not contain any text. The point returned is based on the values in the structure's destination rectangle.

In the case of an offset being equal to a line end, which is also the start of the next line, `TEGetPoint` returns a point corresponding to the line start of the next line. In the case of a dual caret, the primary caret position, the one corresponding to the primary line direction, is returned.

See the Mac Types documentation for a description of the `Point` data type.

Discussion
The line height, taken either from the `lineHeight` field for a monostyled edit structure or from the line-height array, `LHElement` , for a multistyled edit structure, is also used to determine the vertical component. Both the text direction and the primary line direction are used to determine the horizontal component.

The `TEGetPoint` function works for both monostyled and multistyled edit structures.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextEdit.h`

TEGetRecalcHook

Obtains a universal procedure pointer to your recalculation callback. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
TERecalcUPP TEGetRecalcHook (
   void
);
```

Return Value
See the description of the `TERecalcUPP` data type.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextEdit.h`

TEGetScrapHandle

Returns a handle to the TextEdit private scrap. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
Handle TEGetScrapHandle (
   void
);
```

Return Value
See the Mac Types documentation for a description of the `Handle` data type.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextEdit.h`

TEGetScrapLength

Returns the size of the TextEdit private scrap, in bytes. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
long TEGetScrapLength (
   void
);
```

Return Value
The size of the TextEdit private scrap, in bytes.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextEdit.h`

TEGetStyle

Gets character attributes for the specified text. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void TEGetStyle (
   short offset,
   TextStyle *theStyle,
   short *lineHeight,
   short *fontAscent,
   TEHandle hTE
);
```

Parameters
offset
The offset to the text whose character attributes you want to obtain.

theStyle
On output, points to a structure of type TextStyle that contains the character attributes for the current selection range.

lineHeight
A pointer to a value that specifies the line height.

fontAscent
A pointer to a value that specifies the font ascent.

hTE
A handle to the multistyled edit structure containing the text whose character attributes you want to obtain.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextEdit.h`

TEGetStyleHandle

Returns the style handle stored in the designated edit structure's `txFont` and `txFace` fields. The style handle points to the associated style structure, not to a copy of it. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
TEStyleHandle TEGetStyleHandle (
   TEHandle hTE
);
```

Parameters

hTE
A handle to the multistyled edit structure containing the style handle to be returned.

Return Value
A handle to the style structure contained in the specified edit structure (of type `TEStyleRec`). Because only multistyled edit structures have style structures, `TEGetStyleHandle` returns `NULL` when used with a monostyled edit structure. See the description of the `TEStyleHandle` data type.

Discussion
To ensure future compatibility, your application should always use this function rather than manipulate the fields of the edit structure directly.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextEdit.h`

TEGetStyleScrapHandle

Creates a style scrap structure, copies the character attributes associated with the current selection range into it, and returns a handle to it. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
StScrpHandle TEGetStyleScrapHandle (
   TEHandle hTE
);
```

Parameters

hTE

The handle to the edit structure containing the text selection range whose character attributes are to be copied.

Return Value

A handle to the created style scrap structure, or `NULL` if called with a handle to a monostyled structure. See the description of the `StScrpHandle` data type.

Discussion

The `TEGetStyleScrapHandle` function creates a style scrap structure of type `StScrpRec` and copies the character attributes associated with the current selection range of the designated edit structure into it. If the current selection range is an insertion point, `TEGetStyleScrapHandle` first checks the null scrap. If the null scrap contains character attributes, they are written to the newly created style scrap structure. If the null scrap is empty, the attributes associated with the character preceding the insertion point are copied to the style scrap structure.

The `TEGetStyleScrapHandle` function has no impact on the Scrap Manager's desk scrap.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`TextEdit.h`

TEGetText

Returns a handle to the text of the specified edit structure. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
CharsHandle TEGetText (
   TEHandle hTE
);
```

Parameters

hTE

A handle to the edit structure containing the text whose handle you want returned. You pass this handle as an input parameter.

Return Value

A handle to the text contained in the specified edit structure. See page 82 for a description of the `CharsHandle` data type.

Discussion
Given an edit structure that contains text, you can use the `TEGetText` function to get a handle to the text itself. The `TEGetText` function doesn't make a copy of the text. Rather, it returns the handle to the text which is stored as a packed array of characters. (This handle belongs to TextEdit your application must not destroy it.) The `teLength` field of the edit structure contains the length of the text whose handle is returned.

The handle of type `CharsHandle` that is returned by `TEGetText` corresponds to the `hText` field of the `TERec` (page 3021) structure.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextEdit.h`

TEIdle

When called repeatedly, displays a blinking caret at the insertion point, if any exists, in the text of the specified edit structure of an active window. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void TEIdle (
   TEHandle hTE
);
```

Parameters
hTE
A handle to the edit structure.

Discussion
You need to call `TEIdle` only when the window containing the text is active; the caret is blinked only then. TextEdit observes a minimum blink interval, initially set to 32 ticks. No matter how often you call `TEIdle`, the time between blinks is never less than the minimum interval. (The user can adjust the minimum interval setting with the General Controls control panel.)

To maintain a constant frequency of blinking, you need to call `TEIdle` at least once each time through your main event loop. Call it more than once if your application does an unusually large amount of processing each time through the loop.

Call the Event Manager's `GetCaretTime` function to get the blink rate.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextEdit.h`

TEInsert

Inserts the specified text immediately before the selection range or the insertion point in the text of the designated edit structure, redrawing the text as necessary. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void TEInsert (
   const void *text,
   long length,
   TEHandle hTE
);
```

Parameters

text

A pointer to the text to be inserted.

length

The number of characters to be inserted.

hTE

A handle to the edit structure containing the text buffer into which the new text is to be inserted.

Discussion

When you call the `TEInsert` function and a range of text is selected, `TEInsert` doesn't affect the selection range. The `TEInsert` function does not check for a 32 KB limit, so your application must ensure that the inserted text does not exceed this text size limit of 32 KB. The `TEInsert` function recalculates line starts and line heights to adjust for the inserted text.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`TextEdit.h`

TEKey

Replaces the selection range in the text of the specified edit structure with the input character and positions the insertion point just past the inserted character. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void TEKey (
   CharParameter key,
   TEHandle hTE
);
```

Parameters

key

The input character.

hTE

A handle to the edit structure in whose text the character is to be entered.

Discussion
The TextEdit function `TEKey` allows you to handle key-down events and enter text input through the keyboard. If the selection range is an insertion point, `TEKey` inserts the character. (Two-byte characters are passed one byte at a time.)

If the `key` parameter contains a backspace character, the selection range or the character immediately before the insertion point is deleted. When the primary line direction is right-to-left, the character to the right of the insertion point is deleted. When the primary line direction is left-to-right, the character to the left of the insertion point is deleted.

When the user deletes text up to the beginning of a set of character attributes, `TEKey` saves the attributes in the null scrap's style scrap structure. The attributes are saved temporarily to be applied to characters inserted after the deletion. As soon as the user clicks in another area of the text, `TEKey` removes the attributes. `TEKey` redraws the text as necessary.

Call `TEKey` every time the Event Manager function `WaitNextEvent` reports a keyboard event that your application determines should be handled by TextEdit.

Because `TEKey` inserts every character passed in the `key` parameter, your application must filter all characters which aren't actual text, such as keys typed in conjunction with the Command key.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextEdit.h`

TENew

Creates and initializes a monostyled edit structure and allocates a handle to it. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
TEHandle TENew (
   const Rect *destRect,
   const Rect *viewRect
);
```

Parameters

destRect
A pointer to the destination rectangle for the new edit structure, specified in the local coordinates of the current graphics port. This is the area in which text is laid out.

viewRect
A pointer to the view, or visible, rectangle for the new edit structure, specified in the local coordinates of the current graphics port. This is the area of the window in which text is actually displayed.

Return Value
A handle to the newly created edit structure. Your application needs to store the handle to the edit structure that is returned; many functions require it as an input parameter. See the description of the `TEHandle` data type.

Discussion
A monostyled edit structure is one in which all text is restricted to a single font, size, and style. Use the `TENew` function when the text is to be rendered in attributes that are consistent from character to character. Otherwise, use the `TEStyleNew` (page 2999) function.

Call `TENew` once for every edit structure you want allocated. Your application should store the handle to the edit structure that is returned; many functions require it as an input parameter. The edit structure assumes the drawing environment of the graphics port.

If your application contains more than one window where text editing occurs, you need to create an edit structure for each window.

Before this function is called, the window must be in the current graphics port.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextEdit.h`

TENumStyles

Returns the number of character attribute changes contained in the specified range, counting one for the start of the range. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
long TENumStyles (
   long rangeStart,
   long rangeEnd,
   TEHandle hTE
);
```

Parameters

rangeStart
The beginning of the range of text for which the number of style runs (sets of character attributes) or changes is counted and returned.

rangeEnd
The end of the range of text for which the number of style runs (sets of character attributes) or changes is counted and returned.

hTE
A handle to the edit structure containing the range of text.

Return Value
The number of character attribute changes contained in the specified range. This does not necessarily represent the number of unique sets of attributes for the range, because some sets of attributes may be repeated. For monostyled edit structures, `TENumStyles` always returns 1.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextEdit.h`

TEPaste

Replaces the edit structure's selected text with the contents of the private scrap and leaves an insertion point after the inserted text. If the selection range is an insertion point, `TEPaste` inserts the contents of the private scrap there. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void TEPaste (
   TEHandle hTE
);
```

Parameters

hTE

A handle to the edit structure into which the text is to be pasted.

Discussion
When you call `TEPaste`, after it pastes the text from the private scrap, it redraws all of the text as necessary. If the private scrap is empty, `TEPaste` deletes the selection range. If you call `TEPaste` for a multistyled edit structure, it pastes only the text in the private scrap. In this case, `TEPaste` ignores any associated character attribute information stored in the style scrap; instead, it applies the character attributes of the first character of the selection range being replaced to the text. If the selection range is an insertion point, `TEPaste` applies the character attributes of the character preceding the insertion point.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextEdit.h`

TEPinScroll

Scrolls the text within the view rectangle of the specified edit structure by the designated number of pixels. Scrolling stops when the last line of text is scrolled into view. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void TEPinScroll (
   short dh,
   short dv,
   TEHandle hTE
);
```

Parameters

dh

The distance in pixels that the text is to be scrolled horizontally. A positive value moves the text to the right; a negative value moves the text to the left.

dv

The distance in pixels that the text is to be scrolled vertically. A positive value moves the text down; a negative value moves the text up.

hTE

A handle to the edit structure whose text is to be scrolled.

Discussion

The `TEPinScroll` function updates the text on the screen automatically to reflect the new scroll position, as does the `TEScroll` function. The destination rectangle is offset by the amount scrolled. When the edit structure is longer than the text it contains, `TEPinScroll` displays up to the last line of text inclusive, and not beyond it.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`TextEdit.h`

TEReplaceStyle

Replaces any character attributes in the current selection range that match the specified existing character attributes with the specified new character attributes. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void TEReplaceStyle (
   short mode,
   const TextStyle *oldStyle,
   const TextStyle *newStyle,
   Boolean fRedraw,
   TEHandle hTE
);
```

Parameters

mode

A selector that specifies which attributes to replace. It corresponds to any additive combination of the "Text Styling Constants" (page 3036) for font, character style, type size, color, and so forth.

oldStyle

A pointer to a text style structure that specifies the current character attributes to search for in the selected text.

newStyle

A pointer to a text style structure that specifies the new attributes to be set. This structure contains the character attributes to be applied to the current selection range based on the value of `mode`.

fRedraw

A flag that specifies whether or not TextEdit should immediately redraw the text to reflect the attribute changes. A value of `FALSE` delays redrawing until another event forces the update. A value of `TRUE` causes the text to be redrawn immediately using the new character attributes.

hTE

A handle to the multistyled edit structure containing the text selection whose character attributes are to be changed.

Discussion
The `TEReplaceStyle` function replaces any attribute in the current selection range that matches the attribute specified by `oldStyle` with that given by `newStyle`. Only the character attributes specified by `mode` are affected.

Attribute changes are made directly to the style elements (`STElement`) within the style table itself (`TEStyleTable`). If you specify the value `doAll` for the `mode` parameter, `newStyle` replaces `oldStyle` outright. The `TEReplaceStyle` function has no effect on a monostyled edit structure.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextEdit.h`

TEScrapHandle

Returns a handle to the TextEdit private scrap. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
Handle TEScrapHandle (
   void
);
```

Return Value
A handle to the TextEdit private scrap. See the Mac Types documentation for a description of the `Handle` data type.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextEdit.h`

TEScroll

Scrolls the text within the view rectangle of the specified edit structure by the designated number of pixels. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void TEScroll (
   short dh,
   short dv,
   TEHandle hTE
);
```

Parameters

dh

The distance in pixels that the text is to be scrolled horizontally. A positive value moves the text to the right; a negative value moves the text to the left.

dv

The distance in pixels that the text is to be scrolled vertically. A positive value moves the text down; a negative value moves the text up.

hTE

A handle to the edit structure whose text is to be scrolled.

Discussion

The `TEScroll` function updates the text on the screen automatically to reflect the new scroll position. The destination rectangle is offset by the amount scrolled. The `TEScroll` and `TEPinScroll` functions behave the same, except that `TEPinScroll` stops scrolling when the last line of text is scrolled into view.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`TextEdit.h`

TESelView

Ensures, once automatic scrolling has been enabled by a call to the `TEAutoView` function or through the `TEFeatureFlag` function, that the selection range is visible, scrolling it into the view rectangle if necessary. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void TESelView (
   TEHandle hTE
);
```

Parameters

hTE

A handle to the edit structure containing the text selection range.

Discussion

The top left part of the selection range is scrolled into view. If the text is displayed in a rectangle that is not high enough, automatic scrolling can cause text to appear to flicker. If automatic scrolling is disabled, `TESelView` has no effect. For more information, see `TEFeatureFlag` (page 2976).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`TextEdit.h`

TESetAlignment

Sets the alignment of the specified text in an edit structure so that it is centered, right aligned, or left aligned, or aligned according to the line direction. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void TESetAlignment (
   short just,
   TEHandle hTE
);
```

Parameters

just

The alignment for the specified text.The default value of the `just` field of the edit structure is `teFlushDefault`. This means that text alignment is based on the primary line direction which is set by default according to the system script.

For a description of the values you can use in this parameter, see "Text Alignment Constants" (page 3032).

hTE

A handle to the edit structure containing the text.

Discussion
For languages that are read from right to left, text is right aligned by default. For languages that are read from left to right, text is left aligned by default. If you change the alignment, call the Window Manager function `InvalRect` after `TESetAlignment` to redraw the text with the new alignment.

TextEdit does not support justified alignment. To draw justified text, use the QuickDraw Text functions.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextEdit.h`

TESetClickLoop

Installs the address of the application-supplied click loop function in the `clikLoop` field of the edit structure. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void TESetClickLoop (
   TEClickLoopUPP clikProc,
   TEHandle hTE
);
```

Parameters

clikProc

A universal procedure pointer to the customized click loop function.

hTE

A handle to the edit structure whose `clikLoop` field is to be modified.

Discussion

The `TESetClickLoop` function lets you replace the default click loop function. The `TEClick` function repeatedly calls the function that the click loop field points to as long as the user holds down the mouse button within the text of the view rectangle. The default click loop function scrolls only the text. However, you can provide a customized click loop function that scrolls the text and the scroll bars in tandem.

If automatic scrolling is enabled, the default click loop function checks to see if the mouse has been dragged out of the view rectangle; if it has, the function scrolls the text using `TEPinScroll` (page 2988). The amount by which `TEPinScroll` scrolls the text vertically is determined by the `lineHeight` field of the edit structure for monostyled text and the `LHTable` for multistyled text.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`TextEdit.h`

TESetDoTextHook

Sets your do-text-hook callback to be used by TextEdit. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void TESetDoTextHook (
   TEDoTextUPP value
);
```

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`TextEdit.h`

TESetFindWordHook

Sets your set-find-word-hook callback to be used by TextEdit. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void TESetFindWordHook (
   TEFindWordUPP value
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextEdit.h`

TESetRecalcHook

Sets your recalculation callback to be used by TextEdit. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void TESetRecalcHook (
   TERecalcUPP value
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextEdit.h`

TESetScrapHandle

Sets a handle to the TextEdit private scrap. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void TESetScrapHandle (
   Handle value
);
```

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextEdit.h`

TESetScrapLength

Sets the size of the TextEdit private scrap to the specified number of bytes. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void TESetScrapLength (
   long length
);
```

Parameters

length

The size of the private scrap, in bytes.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`TextEdit.h`

TESetSelect

Sets the selection range (or denotes the insertion point) within the text of the specified edit structure. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void TESetSelect (
   long selStart,
   long selEnd,
   TEHandle hTE
);
```

Parameters

selStart

The byte offset at the start of the text selection range. The `selStart` field can range from 0 to 32767.

If `selStart` equals `selEnd`, the new selection range is an insertion point, and a caret is displayed. If `selEnd` is anywhere beyond the last character of the text, `TESetSelect` uses the first position past the last character.

selEnd

The byte offset at the end of the text selection range. The `selEnd` field can range from 0 to 32767.

hTE

A handle to the edit structure.

Discussion

The `TESetSelect` function removes highlighting of the old selection range and highlights the new one.

When only the Roman script system is used, the selection range is always displayed and highlighted as a continuous range of text. However, when one or more script systems requiring mixed-directional display of text are installed, a continuous sequence of characters in memory may appear as a discontinuous selection when displayed.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`TextEdit.h`

TESetStyle

Sets new character attributes, in the specified edit structure, for the current selection range. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void TESetStyle (
   short mode,
   const TextStyle *newStyle,
   Boolean fRedraw,
   TEHandle hTE
);
```

Parameters

mode

A selector that specifies which character attributes are to be changed. The value for `mode` can be any additive combination of the `mode` constants for font, style, type size, color, and so forth. It corresponds to any additive combination of the "Text Styling Constants" (page 3036) for font, character style, type size, color, and so forth.

The value of `mode` specifies which existing character attributes are to be changed to the new character attributes specified by `newStyle`. If `doToggle` is specified along with `doFace` and if an attribute specified in the given `newStyle` parameter exists across the entire selected range of text, then `TESetStyle` removes that attribute. Otherwise, if the attribute doesn't exist across the entire selection range, all of the selected text is set to include that character attribute.

newStyle

A pointer to a structure of type `TextStyle` that specifies the new attributes to be set. This structure contains the character attributes to be applied to the current selection range based on the value of `mode`.

fRedraw

A flag that specifies whether or not TextEdit should immediately redraw the affected text to reflect the new character attribute changes. A value of `TRUE` causes the text to be redrawn immediately. Line breaks, line heights, and line ascents are recalculated. A value of `FALSE` delays redrawing until another event forces the update.

If the `fRedraw` parameter is set to `TRUE`, TextEdit redraws the current selection range using the new character attributes, recalculating line breaks, line heights, and line ascents.

If the `fRedraw` parameter is set to `FALSE`, TextEdit does not redraw the text or recalculate line breaks, line heights, and line ascents. Consequently, when you call a function that uses any of this information, such as `TEGetHeight` (which returns a total height between two specified lines), it does not reflect the new character attributes set with `TESetStyle`. Instead, the function uses the information that was available before `TESetStyle` was called. To update this information, call the `TECalText` (page 2969) function. To be certain that the new information is always reflected, call `TESetStyle` with the `fRedraw` parameter set to `TRUE`.

hTE

A handle to the multistyled edit structure containing the selected text.

Discussion
The `TESetStyle` function has no effect on a monostyled structure.

If you call the `TESetStyle` function when the value of the `selStart` field of an edit structure equals the value of the `selEnd` field (specifying an insertion point), TextEdit stores the input character attributes in the null scrap structure pointed to by the null style handle.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextEdit.h`

TESetStyleHandle

Sets an edit structure's style handle, which is stored in the `txFont` and `txFace` fields. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void TESetStyleHandle (
   TEStyleHandle theHandle,
   TEHandle hTE
);
```

Parameters

`theHandle`
The style handle to be set in the combined `txFont` and `txFace` fields of the specified edit structure.

`hTE`
A handle to the edit structure.

Discussion
The `TESetStyleHandle` function has no effect on monostyled edit structures.

Your application should always use `TESetStyleHandle` rather than manipulate the fields of the edit structure directly.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextEdit.h`

TESetText

Incorporates a copy of the specified text into the designated edit structure. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void TESetText (
   const void *text,
   long length,
   TEHandle hTE
);
```

Parameters

text

A pointer to the text to be copied and incorporated.

length

The number of characters in the text to be incorporated.

hTE

A handle to the edit structure into which the text is to be copied.

Discussion

The function `TESetText` lets you incorporate existing text into the text buffer of an edit structure. The function copies the specified text into the existing `hText` handle of the edit structure, resizing the buffer, if necessary it doesn't bring in the original text. The copied text is wrapped to the destination rectangle, and its `lineStarts` and `nLines` fields are calculated accordingly. The selection range is set to an insertion point at the end of the incorporated text. The `TESetText` function does not display the copied text on the screen. To do this, call `TEUpdate`.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`TextEdit.h`

TEStyleInsert

Inserts the specified text immediately before the selection range or the insertion point in the edit structure's text and applies the specified character attributes to the text, redrawing the text if necessary. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void TEStyleInsert (
   const void *text,
   long length,
   StScrpHandle hST,
   TEHandle hTE
);
```

Parameters

text

A pointer to the text to be inserted.

length

The length, in bytes, of the text to be inserted.

hST

A handle to the style scrap structure containing the character attribute information to be applied to the inserted text.

hTE

A handle to the edit structure into which the text is to be inserted.

Discussion
You should create your own style scrap structure, specifying the character attributes to be inserted and applied to the text, and pass its handle to `TEStyleInsert` as the value of the `hST` parameter. The character attributes are copied directly into the style structure's (`TEStyleRec`) style table.

The `TEStyleInsert` function does not affect the current selection range.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextEdit.h`

TEStyleNew

Creates a multistyled edit structure and allocates a handle to it. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
TEHandle TEStyleNew (
   const Rect *destRect,
   const Rect *viewRect
);
```

Parameters

destRect

A pointer to the destination rectangle for the new edit structure, specified in the local coordinates of the current graphics port. This is the area in which text is laid out.

viewRect

A pointer to the view rectangle for the new edit structure, specified in the local coordinates of the current graphics port. This is the area of the window in which text is actually displayed.

Return Value
A handle to the newly created edit structure. Your application needs to store the handle to the edit structure that is returned; many functions require it as an input parameter. See the description of the `TEHandle` data type.

Discussion
A multistyled edit structure contains text whose attributes, including font, size, and style, can vary from character to character. Always use the `TEStyleNew` function to create an edit structure for text that uses varying character attributes. The `TEStyleNew` function sets the `txSize`, `lineHeight`, and `fontAscent` fields of the edit structure to –1, allocates a style structure, and stores a handle to the style structure in the `txFont` and `txFace` fields. The `TEStyleNew` function creates and initializes a null scrap that is used by TextEdit functions throughout the life of the edit structure.

Call `TEStyleNew` once for every edit structure you want allocated. Your application needs to store the handle to the edit structure that is returned; many functions require it as an input parameter.

If your application contains more than one window where text editing occurs, you need to create an edit structure for each window.

Before this function is called, the window must be in the current graphics port.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextEdit.h`

TEStylePaste

Pastes text and its associated character attribute information from the desk scrap into the edit structure's text at the insertion point—if the current selection range is an insertion point—or it replaces the current selection range. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void TEStylePaste (
   TEHandle hTE
);
```

Parameters

hTE

A handle to the edit structure into which the text is to be pasted.

Discussion
When you call `TEStylePaste` and there is no character attribute information associated with text in the desk scrap, `TEStylePaste` first checks the null scrap. If the null scrap contains character attribute information, this is used. If the null scrap is empty, `TEStylePaste` gives the text the same attributes as those of the first character of the replaced selection range or that of the preceding character if the selection is an insertion point.

For a monostyled edit structure, `TEStylePaste` pastes the text only; there is no associated character attribute information because all the text uses the same attributes.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextEdit.h`

TETextBox

Draws the indicated text in a given rectangle, with the specified alignment. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void TETextBox (
   const void *text,
   long length,
   const Rect *box,
   short just
);
```

Parameters

text

A pointer to the text to be drawn.

length

The number of bytes comprising the text.

box

A pointer to the rectangle where the text is to be drawn. The rectangle is specified in local coordinates (of the current graphics port) and must be at least as wide as the first character drawn. (A good rule of thumb is to make the rectangle at least 20 pixels wide.

just

The kind of justification (alignment) used for the specified text.

Discussion

The `TETextBox` function provides you with an easy way to display static text to a user. It creates its own monostyled edit structure, which it deletes when finished with it, so you cannot edit the text it draws. The `TETextBox` function breaks a line of text correctly. You can specify how text is aligned in the box using any of the "Text Alignment Constants" (page 3032).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`TextEdit.h`

TEToScrap

Copies the contents of the TextEdit private scrap to the desk scrap. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
OSErr TEToScrap (
   void
);
```

Return Value

A result code. See "TextEdit Result Codes" (page 3037).

Discussion

You use the `TEToScrap` function to move monostyled text across applications or between an application and a desk accessory. Call the Scrap Manager function `ZeroScrap` to initialize the desk scrap or clear its contents before calling `TEToScrap`.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In
`TextEdit.h`

TEUpdate

Draws the specified text within a given update rectangle. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void TEUpdate (
   const Rect *rUpdate,
   TEHandle hTE
);
```

Parameters

rUpdate

The update rectangle, given in the coordinates of the current graphics port, where the specified text is to be drawn.

hTE

A handle to the edit structure containing the text to be drawn.

Discussion
Call `TEUpdate` every time the Event Manager function `WaitNextEvent` reports an update event for a text editing window—after you call the Window Manager function `BeginUpdate` , and before you call the `EndUpdate` function. You also need to erase the update region with the `EraseRect` function. If you don't, the caret can sometimes remain visible when the window is deactivated.

Availability
Available in Mac OS X v10.0 and later.
Deprecated in Mac OS X v10.4.
Not available to 64-bit applications.

Declared In
`TextEdit.h`

TEUseStyleScrap

Assigns new character attributes to the specified range of text in the designated edit structure. (Deprecated in Mac OS X v10.4. Use Multilingual Text Engine instead; see *Handling Unicode Text Editing With MLTE*.)

```
void TEUseStyleScrap (
   long rangeStart,
   long rangeEnd,
   StScrpHandle newStyles,
   Boolean fRedraw,
   TEHandle hTE
);
```

Parameters

rangeStart

The offset of the first character in the text of the edit structure to which the character attributes are to be applied.

rangeEnd

The offset of the last character in the text of the edit structure to which the character attributes are to be applied.

newStyles

A handle to a style scrap structure. The style scrap structure contains the attributes to be applied to the specified range of text. If the value of `newStyles` is `NULL`, no action is performed. Each element in the style scrap structure contains a field that is the offset of the beginning of the element's character attributes. This field (`scrpStartChar`) defines the boundaries for the scrap's style runs.

Depending on the requirements of your application, you can create a style scrap structure directly and pass its handle to `TEUseStyleScrap` as the value of `newStyles` or you can use a style scrap structure created by `TEGetStyleScrapHandle`.

fRedraw

A flag that specifies whether TextEdit should immediately redraw the selection range using the new character attributes. If the `fRedraw` parameter is set to `TRUE`, the attributes are applied immediately to the specified range of text, and line breaks, line heights, and line ascents are recalculated. If `fRedraw` is set to `FALSE`, the new character attributes are not reflected in the view rectangle until the next update event occurs.

hTE

A handle to the edit structure containing the range of text to which the character attributes are to be applied. If the handle points to a monostyled edit structure (created using `TENew`), no action is performed.

Discussion

The `TEUseStyleScrap` function writes the character attribute information into the style structure's style table and updates the style run table.

Regardless of whether the text is redrawn, the current selection range is not changed; if characters are highlighted before `TEUseStyleScrap` is called, they remain highlighted after it is called. However, if characters within the current selection range also fall within the specified range of text, they are rendered in the new character attributes when the text is redrawn.

The `TEUseStyleScrap` function applies the first element's attributes to the characters from `rangeStart` up to the `scrpStartChar` field of the next element. The function terminates without error if it prematurely reaches the end of the range or if there are not enough scrap style elements to cover the whole range. In the latter case, the function applies the last set of character attributes in the style scrap structure to the remainder of the range.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.4.

Not available to 64-bit applications.

Declared In

`TextEdit.h`

Callbacks

CaretHookProcPtr

Defines a pointer to a caret-hook callback.

```
typedef void (*CaretHookProcPtr) (
    const Rect * r,
    TEPtr pTE
);
```

If you name your function `MyCaretHookProc`, you would declare it like this:

```
void CaretHookProcPtr (
    const Rect * r,
    TEPtr pTE
);
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

`TextEdit.h`

DrawHookProcPtr

Defines a pointer to a draw-hook callback.

```
typedef void (*DrawHookProcPtr) (
    unsigned short textOffset,
    unsigned short drawLen,
    void * textBufferPtr,
    TEPtr pTE,
    TEHandle hTE
);
```

If you name your function `MyDrawHookProc`, you would declare it like this:

```
void DrawHookProcPtr (
    unsigned short textOffset,
    unsigned short drawLen,
    void * textBufferPtr,
    TEPtr pTE,
    TEHandle hTE
);
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

`TextEdit.h`

EOLHookProcPtr

Defines a pointer to an EOL-hook callback.

```
typedef Boolean (*EOLHookProcPtr) (
    char theChar,
    TEPtr pTE,
    TEHandle hTE
);
```

If you name your function `MyEOLHookProc`, you would declare it like this:

```
Boolean EOLHookProcPtr (
    char theChar,
    TEPtr pTE,
    TEHandle hTE
);
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TextEdit.h`

HighHookProcPtr

Defines a pointer to a high-hook callback.

```
typedef void (*HighHookProcPtr) (
    const Rect * r,
    TEPtr pTE
);
```

If you name your function `MyHighHookProc`, you would declare it like this:

```
void HighHookProcPtr (
    const Rect * r,
    TEPtr pTE
);
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TextEdit.h`

HitTestHookProcPtr

Defines a pointer to a hit-test hook callback.

```
typedef Boolean (*HitTestHookProcPtr) (
    unsigned short styleRunLen,
    unsigned short styleRunOffset,
    unsigned short slop,
    void * textBufferPtr,
    TEPtr pTE,
    TEHandle hTE,
    unsigned short * pixelWidth,
    unsigned short * charOffset,
    Boolean * pixelInChar
);
```

If you name your function `MyHitTestHookProc`, you would declare it like this:

```
Boolean HitTestHookProcPtr (
    unsigned short styleRunLen,
    unsigned short styleRunOffset,
    unsigned short slop,
    void * textBufferPtr,
    TEPtr pTE,
    TEHandle hTE,
    unsigned short * pixelWidth,
    unsigned short * charOffset,
    Boolean * pixelInChar
);
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TextEdit.h`

NWidthHookProcPtr

Defines a pointer to a width-hook callback.

```
typedef unsigned short (*NWidthHookProcPtr) (
    unsigned short styleRunLen,
    unsigned short styleRunOffset,
    short slop,
    short direction,
    void * textBufferPtr,
    short * lineStart,
    TEPtr pTE,
    TEHandle hTE
);
```

If you name your function `MyNWidthHookProc`, you would declare it like this:

```
unsigned short NWidthHookProcPtr (
    unsigned short styleRunLen,
    unsigned short styleRunOffset,
    short slop,
    short direction,
    void * textBufferPtr,
    short * lineStart,
```

```
    TEPtr pTE,
    TEHandle hTE
);
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TextEdit.h`

TEClickLoopProcPtr

Defines a pointer to a click-loop callback.

```
typedef Boolean (*TEClickLoopProcPtr) (
    TEPtr pTE
);
```

If you name your function `MyTEClickLoopProc`, you would declare it like this:

```
Boolean TEClickLoopProcPtr (
    TEPtr pTE
);
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TextEdit.h`

TEDoTextProcPtr

Defines a pointer to a do-text callback.

```
typedef void (*TEDoTextProcPtr) (
    TEPtr pTE,
    unsigned short firstChar,
    unsigned short lastChar,
    short selector,
    GrafPtr * currentGrafPort,
    short * charPosition
);
```

If you name your function `MyTEDoTextProc`, you would declare it like this:

```
void TEDoTextProcPtr (
    TEPtr pTE,
    unsigned short firstChar,
    unsigned short lastChar,
    short selector,
    GrafPtr * currentGrafPort,
    short * charPosition
```

```
);
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TextEdit.h`

TEFindWordProcPtr

Defines a pointer to a find-word callback.

```
typedef void (*TEFindWordProcPtr) (
    unsigned short currentPos,
    short caller,
    TEPtr pTE,
    TEHandle hTE,
    unsigned short * wordStart,
    unsigned short * wordEnd
);
```

If you name your function `MyTEFindWordProc`, you would declare it like this:

```
void TEFindWordProcPtr (
    unsigned short currentPos,
    short caller,
    TEPtr pTE,
    TEHandle hTE,
    unsigned short * wordStart,
    unsigned short * wordEnd
);
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TextEdit.h`

TERecalcProcPtr

Defines a pointer to a recalculation callback.

```
typedef void (*TERecalcProcPtr) (
    TEPtr pTE,
    unsigned short changeLength,
    unsigned short * lineStart,
    unsigned short * firstChar,
    unsigned short * lastChar
);
```

If you name your function `MyTERecalcProc`, you would declare it like this:

```
void TERecalcProcPtr (
```

```
    TEPtr pTE,
    unsigned short changeLength,
    unsigned short * lineStart,
    unsigned short * firstChar,
    unsigned short * lastChar
);
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TextEdit.h`

TextWidthHookProcPtr

Defines a pointer to a width-hook callback.

```
typedef unsigned short (*TextWidthHookProcPtr) (
    unsigned short textLen,
    unsigned short textOffset,
    void * textBufferPtr,
    TEPtr pTE,
    TEHandle hTE
);
```

If you name your function `MyTextWidthHookProc`, you would declare it like this:

```
unsigned short TextWidthHookProcPtr (
    unsigned short textLen,
    unsigned short textOffset,
    void * textBufferPtr,
    TEPtr pTE,
    TEHandle hTE
);
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TextEdit.h`

TSMTEPostUpdateProcPtr

Defines a pointer to a post-update callback.

```
typedef void (*TSMTEPostUpdateProcPtr) (
    TEHandle textH,
    long fixLen,
    long inputAreaStart,
    long inputAreaEnd,
    long pinStart,
    long pinEnd,
    long refCon
);
```

If you name your function `MyTSMTEPostUpdateProc`, you would declare it like this:

```
void TSMTEPostUpdateProcPtr (
    TEHandle textH,
    long fixLen,
    long inputAreaStart,
    long inputAreaEnd,
    long pinStart,
    long pinEnd,
    long refCon
);
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TSMTE.h`

TSMTEPreUpdateProcPtr

Defines a pointer to a pre-udate callback.

```
typedef void (*TSMTEPreUpdateProcPtr) (
    TEHandle textH,
    long refCon
);
```

If you name your function `MyTSMTEPreUpdateProc`, you would declare it like this:

```
void TSMTEPreUpdateProcPtr (
    TEHandle textH,
    long refCon
);
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TSMTE.h`

WidthHookProcPtr

Defines a pointer to a width-hook callback.

```
typedef unsigned short (*WidthHookProcPtr) (
    unsigned short textLen,
    unsigned short textOffset,
    void * textBufferPtr,
    TEPtr pTE,
    TEHandle hTE
);
```

If you name your function `MyWidthHookProc`, you would declare it like this:

```
unsigned short WidthHookProcPtr (
    unsigned short textLen,
    unsigned short textOffset,
    void * textBufferPtr,
    TEPtr pTE,
    TEHandle hTE
);
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TextEdit.h`

Data Types

CaretHookUPP

Defines a universal procedure pointer (UPP) to a caret-hook callback.

```
typedef CaretHookProcPtr CaretHookUPP;
```

Discussion
For more information, see the description of the CaretHookUPP () callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TextEdit.h`

Chars

Defines an array of characters.

```
typedef  char Chars[32001];
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TextEdit.h`

CharsPtr

Defines a data type for a character pointer.

```
typedef char* CharsPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
TextEdit.h

CharsHandle

Defines a handle to a character pointer.

```
typedef CharsPtr* CharsHandle;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
TextEdit.h

DrawHookUPP

Defines a universal procedure pointer (UPP) to a draw-hook callback.

```
typedef DrawHookProcPtr DrawHookUPP;
```

Discussion
For more information, see the description of the DrawHookUPP () callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
TextEdit.h

EOLHookUPP

Defines a universal procedure pointer (UPP) to an EOL-hook callback.

```
typedef EOLHookProcPtr EOLHookUPP;
```

Discussion
For more information, see the description of the EOLHookUPP () callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
TextEdit.h

HighHookUPP

Defines a universal procedure pointer (UPP) to a high-hook callback.

```
typedef HighHookProcPtr HighHookUPP;
```

Discussion
For more information, see the description of the HighHookUPP () callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TextEdit.h`

HitTestHookUPP

Defines a universal procedure pointer (UPP) to a hit-test hook callback.

```
typedef HitTestHookProcPtr HitTestHookUPP;
```

Discussion
For more information, see the description of the HitTestHookUPP () callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TextEdit.h`

LHHandle

Defines a handle to a line-height table pointer.

```
typedef LHPtr *     LHHandle;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TextEdit.h`

LHElement

Contains height and ascent information.

```
struct LHElement {
    short lhHeight;
    short lhAscent;
};
typedef struct LHElement LHElement;
typedef LHElement * LHPtr;
```

Fields

`lhHeight`

The line height, in points. This is the maximum value for any individual character attribute in the line.

`lhAscent`

The font ascent, in points; this is the maximum value for any individual character attribute in a line.

Discussion

The line-height table, defined by the `LHTable` data type, provides an array of line heights to hold the vertical spacing information for a given edit structure. It also contains line ascent information. The null style structure, defined by the `NullStRec` data type, contains the null scrap which is used to store character attribute information for a null selection.

The line height table holds vertical spacing information for the text of an edit structure. This table parallels the `lineStarts` array in the edit structure itself. Its length equals the edit structure's `nLines` field plus 1 for a dummy entry at the end, just as the `lineStarts` array ends with a dummy entry that has the same value as the length of the text. The table's contents are recalculated whenever the line starting values are themselves recalculated with the `TECalText` function or whenever an editing action causes recalibration.

The line height table is used only if the `lineHeight` and `fontAscent` fields in the edit structure are negative; positive values in those fields specify fixed vertical spacing, overriding the information in the table. The line height table is of type `LHTable`, which is an array of elements of `LHElement`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`TextEdit.h`

LHTable

Defines an array of line-height elements.

```
typedef LHElement LHTable[8001];
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

`TextEdit.h`

NullStHandle

Defines a handle to a null scrap record pointer.

```
typedef NullStPtr *      NullStHandle;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TextEdit.h`

NullStRec

Contains the null scrap.

```
struct NullStRec {
    long teReserved;
    StScrpHandle nullScrap;
};
typedef struct NullStRec NullStRec;
typedef NullStRec * NullStPtr;
```

Fields

`teReserved`
This field is reserved for future expansion.

`nullScrap`
A handle to the style scrap structure.

Discussion
The `NullSTRec` data type defines the null style structure.

The null style structure contains the null scrap, which is used to store the character attribute information for a null selection (insertion point). A number of functions either write this character attribute information to the null scrap or read it from this scrap (to be applied to inserted text). The null scrap is created and initialized when an application calls `TEStyleNew` to create a multistyled edit structure. The null scrap is retained for the life of the edit structure; it is destroyed when `TEDispose` destroys the edit structure and releases the memory allocated for it.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TextEdit.h`

NWidthHookUPP

Defines a universal procedure pointer (UPP) to a width-hook callback.

```
typedef NWidthHookProcPtr NWidthHookUPP;
```

Discussion
For more information, see the description of the NWidthHookUPP () callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
TextEdit.h

ScrpSTElement

Contains the scrap style table.

```
struct ScrpSTElement {
    long scrpStartChar;
    short scrpHeight;
    short scrpAscent;
    short scrpFont;
    StyleField scrpFace;
    short scrpSize;
    RGBColor scrpColor;
};
typedef struct ScrpSTElement ScrpSTElement;
typedef ScrpSTElement ScrpSTTable[1601];
```

Fields

`scrpStartChar`
The offset to the beginning of a style structure in the scrap.

`scrpHeight`
The line height. You can determine the line height and the font ascent using the QuickDraw function `GetFontInfo`.

`scrpAscent`
The font ascent. See `scrpHeight`.

`scrpFont`
The font family ID.

`scrpFace`
The character style (such as plain, bold, underline).

`scrpSize`
The size, in points.

`scrpColor`
The RGB (red, green, blue) color for the style scrap.

Discussion
The style scrap structure contains the scrap style table. Unlike the main style table for an edit structure, the scrap style table may contain duplicate elements; the entries in the table correspond one-to-one with the style runs in the text. The `scrpStartChar` field of each entry gives the starting position for the run.

The `scrpStyleTab` data type defines the scrap style table data structure, which is an array of scrap style element structures. The `ScrpSTElement` data type defines each scrap style element structure.

Availability
Available in Mac OS X v10.0 and later.

Declared In
TextEdit.h

ScrpSTTable

Contains an array of scrap style elements.

```
typedef ScrpSTElement ScrpSTTable[1601];
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TextEdit.h`

STElement

Contains one entry for each distinct set of character attributes used in the text of an edit structure.

```
struct STElement {
    short stCount;
    short stHeight;
    short stAscent;
    short stFont;
    StyleField stFace;
    short stSize;
    RGBColor stColor;
};
typedef struct STElement STElement;
typedef STElement * STPtr;
```

Fields

`stCount`
A reference count of character runs using this set of character attributes.

`stHeight`
The line height for this run, in points.

`stAscent`
The font ascent for this run, in points.

`stFont`
The font family ID.

`stFace`
The character style (bold, italic, and so forth). This field consists of two bytes. The low-order byte contains the character style. TextEdit uses the high bit (bit 15) of the high-order byte to store the style run direction: it uses 0 for left-to-right text, and 1 for right-to-left text.

`stSize`
The text size, in points.

`stColor`
The RGB (red, green, blue) color.

Discussion
The style table contains one entry for each distinct set of character attributes used in the text of an edit structure. Each entry is defined in a style element structure. The size of the table is given by the `nStyles` field of the style structure. There is no duplication; each set of character attributes appears exactly once in the table. A reference count tells how many times each set of attributes is used in the table. The `TEStyleTable` data type defines the style table. Th `eSTElement` data type defines the style element structure.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TextEdit.h`

STHandle

Defines a handle to a style table pointer.

```
typedef STPtr *     STHandle;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TextEdit.h`

StScrpHandle

Defines a handle to a scrap style table pointer.

```
typedef StScrpPtr *     StScrpHandle;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TextEdit.h`

StScrpRec

Contains information used by functions to store character attribute information temporarily.

```
struct StScrpRec {
    short scrpNStyles;
    ScrpSTTable scrpStyleTab;
};
typedef struct StScrpRec StScrpRec;
typedef StScrpRec * StScrpPtr;
```

Fields

`scrpNStyles`
The number of style runs (sets of character attributes) used in the text. This determines the size of the style table. When character attribute information is written to the null scrap, this field is set to 1; when the character attribute information is removed, this field is set to 0.

`scrpStyleTab`
The scrap style table containing an element for each style run (set of character attributes).

Discussion
The style scrap structure, defined by the `StScrpRec` data type, is used by functions to store character attribute information temporarily. The scrap style table, defined by the `scrpStyleTab` data type, is contained in the style scrap structure. The scrap style element structure, defined by the `ScrpSTElement` data type, contains the character attribute information for an element in the scrap style table. One scrap style element structure exists for each sequential attribute change in the associated text.

The style scrap is used for storing character attribute information associated with the current text selection or insertion point, character attribute information to be applied to text, or multistyled text that is cut or copied. When multistyled text is cut or copied, the character attribute information is written to both the style scrap and the desk scrap.

In most cases, the style scrap is created dynamically as needed by functions. However, a style scrap structure can be created directly without using the `TEGetStyleScrapHandle` function; the character attribute information written to it can be applied to inserted text through `TEStyleInsert` or to existing text through `TEUseStyleScrap`.

The format of the style scrap is defined by a style scrap structure of type `STScrpRec`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TextEdit.h`

StyleRun

Contains information for a style run.

```
struct StyleRun {
    short startChar;
    short styleIndex;
};
typedef struct StyleRun StyleRun;
```

Fields
`startChar`
The starting character position.
`styleIndex`
The run's index in the style table.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TextEdit.h`

TEClickLoopUPP

Defines a universal procedure pointer (UPP) to a click-loop callback.

```
typedef TEClickLoopProcPtr TEClickLoopUPP;
```

Discussion
For more information, see the description of the TEClickLoopUPP () callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TextEdit.h`

TEDoTextUPP

Defines a universal procedure pointer (UPP) to a do-text callback.

```
typedef TEDoTextProcPtr TEDoTextUPP;
```

Discussion
For more information, see the description of the TEDoTextUPP () callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TextEdit.h`

TEFindWordUPP

Defines a universal procedure pointer (UPP) to a find-word callback.

```
typedef TEFindWordProcPtr TEFindWordUPP;
```

Discussion
For more information, see the description of the TEFindWordUPP () callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TextEdit.h`

TEHandle

Defines a handle to a TextEdit record pointer.

```
typedef TEPtr* TEHandle;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TextEdit.h`

TEIntHook

Defines a data type for a TextEdit integer hook.

```
typedef  short TEIntHook;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TextEdit.h`

TEPtr

Defines a pointer to a TextEdit record.

```
typedef TTERec* TEPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TextEdit.h`

TERec

Stores display and editing information for TextEdit.

```
struct TERec {
    Rect destRect;
    Rect viewRect;
    Rect selRect;
    short lineHeight;
    short fontAscent;
    Point selPoint;
    short selStart;
    short selEnd;
    short active;
    WordBreakUPP wordBreak;
    TEClickLoopUPP clickLoop;
    long clickTime;
    short clickLoc;
    long caretTime;
    short caretState;
    short just;
    short teLength;
    Handle hText;
    long hDispatchRec;
    short clikStuff;
    short crOnly;
    short txFont;
    StyleField txFace;
    short txMode;
    short txSize;
    GrafPtr inPort;
    HighHookUPP highHook;
    CaretHookUPP caretHook;
    short nLines;
    short lineStarts[16001];
};
typedef struct TERec TERec;
typedef TERec * TEPtr;
```

Fields

`destRect`

The destination rectangle, in local coordinates.

`viewRect`

The view rectangle, in local coordinates.

`selRect`

The selection rectangle, whose boundaries are defined in local coordinates. This value is the current selection range or insertion point.

`lineHeight`

The vertical spacing of lines of text. Vertical spacing may be fixed or it may vary from line to line, depending upon specific text attributes. If the value of `lineHeight` is greater than 0, this field specifies the fixed vertical distance from the ascent line of one line of text down to the ascent line of the next.

If the value of `lineHeight` is less than 1, then this field specifies the vertical distance from the ascent line of one line of text down to the ascent line of the next calculated independently for each line, based on the maximum value for any individual character attribute on that line.

`fontAscent`

The font ascent line. If the value of `fontAscent` is greater than 0, this field specifies how far above the base line the pen is positioned to begin drawing the caret or highlighting.

For single-spaced text, this is the height of the text in pixels (the height of the tallest characters in the font from the base line). If the value of `fontAscent` is less than 1, this field specifies the font ascent calculated independently for each line, based on maximum value for any individual character attribute on that line.

`selPoint`

The point selected with the mouse, in the local coordinates of the current graphics port. The assembly-language offset for this field is named `teSelPoint`.

`selStart`

The byte offset of the beginning of a selection range. Note that byte offset 0 refers to the first byte in the text buffer.

`selEnd`

The byte offset of the end of a selection range. To include that byte, this value must be 1 greater than the position of the last byte offset of the text.

`active`

This field is used internally by TextEdit. It is set when an edit structure is activated through `TEActivate` and then reset when the edit structure is rendered inactive through `TEDeactivate`. To ensure future compatibility, use `TEActivate` or `TEDeactivate` to access this field.

`wordBreak`

A universal procedure pointer to the structure's word selection break function. This function determines the word that is highlighted when the user double-clicks in the text and the position at which text is wrapped at the end of a line.

`clickLoop`

A universal procedure pointer to the click loop function.The specified click loop function is called repeatedly by the `TEClick` function as long as the mouse button is held down within the text.

`clickTime`

This field is for internal use only.

`clickLoc`

This field is for internal use only.

`caretTime`

This field is for internal use only.

`caretState`

This field is for internal use only.

`just`

The type of text alignment: default (according to primary line direction), left, center, or right.

`teLength`

The number of bytes in the text to be edited. For two-byte systems, potentially twice the number of characters. Initially set to zero. The maximum length is 32767 bytes.

`hText`

A handle to the text. Initially, it points to a zero-length block of text in the heap.

`hDispatchRec`

A handle to the TextEdit dispatch structure. This field is for internal use only; do not modify this field, or copy it to another edit structure. Each edit structure has its own dispatch structure. Attempting to use the dispatch structure of one edit structure with another edit structure can cause TextEdit to crash.

`clikStuff`

This field is for internal use only. TextEdit sets this field to reflect whether the most recent mouse-down event occurred on the leading or trailing edge of a glyph. TextEdit uses this value in determining a caret position.

`crOnly`

A value specifying whether or not text wraps at the right edge of the destination rectangle. If `crOnly` is positive, text does wrap. Otherwise, new lines are displayed only at Carriage Returns.

If `crOnly` is negative, new lines are specified explicitly by `Return` characters only; text does not wrap at the edge of the destination rectangle. (This is useful in an application similar to a programming-language editor, where you may not want a single line of code to be split onto two lines.

`txFont`

The font of all the text in the edit structure, if the `txSize` field of this edit structure 0. If you change this value, the entire text of this edit structure has the new characteristic when it is redrawn also remember to change the `lineHeight` and `fontAscent` fields.

If the `txSize` field is –1, this field combines with `txFace` to hold a handle to the associated style structure.

`txFace`

The character attributes of all the text in an edit structure, if the `txSize` field of this edit structure 0. If you change this value, the entire text of this edit structure has the new characteristic when it is redrawn also, remember to change the `lineHeight` and `fontAscent` fields as well.

If the `txSize` field is –1, this field combines with `txFont` to hold a handle to the associated style structure.

`txMode`

The pen mode of all the text in the edit structure. If you change this value, the entire text of this edit structure has the new characteristic when it is redrawn; also, remember to change the `lineHeight` and `fontAscent` fields as well.

`txSize`

Depending on its value, `txSize` either contains the point size of all of the text or it acts as a flag indicating whether or not there is associated character attribute information. If `txSize` 0, this is a monostyled edit structure, that is, all text is set in a single font, size, and face, and the value of `txSize` is the size of the text. If `txSize` is –1, the edit structure contains associated character attribute information and the `txFont` and `txFace` fields combine to form a handle to the style structure.

`inPort`

A pointer to the graphics port associated with this edit structure.

`highHook`

A pointer to the function that deals with text highlighting. In assembly language, the `highHook` field is located at the offset `teHiHook`.

`caretHook`

A pointer to the function that controls the appearance of the caret. In assembly language, the `caretHook` field is located at the offset `teCarHook`.

`nLines`

The number of lines in the text.

`lineStarts`

An array containing the character position of the first character in each line. It is declared to have 16001 elements to comply with Pascal range checking. This is a dynamic data structure, having only as many elements as needed. TextEdit calculates these values internally, so do not change the elements of the `lineStarts` array. Because this data structure grows and shrinks, the size of the edit structure changes.

Discussion
The edit structure, defined by the `TERec` data type, stores the display and editing information for TextEdit. Along with various subsidiary data structures, the style structure, defined by the `TEStyleRec` data type, stores the character attribute information for the text of the edit structure.

The edit structure contains display, storage, styling, and other information. Although some fields are used differently for multistyled edit structures and monostyled edit structures, the structure of an edit structure is the same whether the text is multistyled or monostyled.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TextEdit.h`

TERecalcUPP

Defines a universal procedure pointer (UPP) to a recalculation callback.

```
typedef TERecalcProcPtr TERecalcUPP;
```

Discussion
For more information, see the description of the TERecalcUPP () callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TextEdit.h`

TEStyleRec

Stores the character attribute information for the text of a multistyled edit structure.

```
struct TEStyleRec {
    short nRuns;
    short nStyles;
    STHandle styleTab;
    LHHandle lhTab;
    long teRefCon;
    NullStHandle nullStyle;
    StyleRun runs[8001];
};
typedef struct TEStyleRec TEStyleRec;
typedef TEStyleRec * TEStylePtr;
```

Fields

`nRuns`
The number of style runs in the text.

`nStyles`
The number of distinct sets of character attributes used in the text; this forms the size of the style table.

`styleTab`
A handle to the style table.

`lhTab`
A handle to the line height table.

`teRefCon`
A reference constant for use by applications. The application can use this 32-bit field to suit its needs.

`nullStyle`
A handle to the style scrap structure used to store the character attribute information for a null selection.

`runs`
A table of style runs that is of indefinite length.

Discussion

The style structure stores the character attribute information for the text of a multistyled edit structure. If an edit structure has associated character attribute information, its `txFont` and `txFac` e fields combine to hold a style handle, of type `TEStyleHandle`, to its style structure. The text is divided into style runs, summarized in the style run table, of type `StyleRun`, which is part of the style structure. Each entry in the style run table gives the starting character position of a run and an index into the style table, of type `TEStyleTable`.

The style table element pointed to by the style run index describes the character attributes for that run.

To determine the length of a run, you subtract its start position from that of the next entry in the style run table. A dummy entry at the end of the style run table delimits the length of the last run; its start position is equal to the overall number of characters in the text, plus 1. The `TEStyleRec` data type defines the style structure.

The style run table, defined by the `StyleRun` data type, is an array that contains the boundaries of each style run and an index to its character attribute information in the style element array. The style table, defined by the `TEStyleTable` data type, contains one entry for each distinct set of character attributes used in the text of the edit structure.

Availability

Available in Mac OS X v10.0 and later.

Declared In
TextEdit.h

TEStyleTable

Defines and array of style elements.

```
typedef STElement TEStyleTable[1777];
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
TextEdit.h

TextStyle

Contains text style information.

```
struct TextStyle {
    short tsFont;
    StyleField tsFace;
    short tsSize;
    RGBColor tsColor;
};
typedef struct TextStyle TextStyle;
typedef TextStyle * TextStylePtr;
```

Fields

`tsFont`
The font family number.

`tsFace`
The character style (bold, italic, plain, and so forth).

`tsSize`
The text size in points.

`tsColor`
The RGB (red, green, blue) color.

`tsColor`

Discussion
Text style structures, which are passed as variables or reference parameters, are used for communicating character attribute information between the application and several TextEdit functions, such as `TEContinuousStyle` and `TEReplaceStyle`. They carry the same information as the style element structures in the style table, but without the reference count, line height, and font ascent.

Availability
Available in Mac OS X v10.0 and later.

Declared In
TextEdit.h

TextWidthHookUPP

Defines a universal procedure pointer (UPP) to a width-hook callback.

```
typedef TextWidthHookProcPtr TextWidthHookUPP;
```

Discussion
For more information, see the description of the TextWidthHookUPP () callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TextEdit.h`

TSMDialogPeek

Defines a data type for a TSM dialog pointer.

```
typedef TSMDialogPtr TSMDialogPeek;
```

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In
`TSMTE.h`

TSMDialogPtr

Defines a pointer to a TSM dialog record.

```
typedef TSMDialogRecord* TSMDialogPtr;
```

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In
`TSMTE.h`

TSMDialogRecord

Contains information for a TSM dialog record.

```
struct TSMDialogRecord {
  DialogRecord        fDialog;
  TSMDocumentID       fDocID;
  TSMTERecHandle      fTSMTERecH;
  long                fTSMTERsvd[3];
};
typedef struct TSMDialogRecord  TSMDialogRecord;
```

Availability
Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In
TSMTE.h

TSMTEPostUpdateUPP

Defines a universal procedure pointer (UPP) to a post-update callback.

```
typedef TSMTEPostUpdateProcPtr TSMTEPostUpdateUPP;
```

Discussion
For more information, see the description of the TSMTEPostUpdateUPP () callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
TSMTE.h

TSMTEPreUpdateUPP

Defines a universal procedure pointer (UPP) to a pre-update callback.

```
typedef TSMTEPreUpdateProcPtr TSMTEPreUpdateUPP;
```

Discussion
For more information, see the description of the TSMTEPreUpdateUPP () callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
TSMTE.h

TSMTERec

Defines a TSMTE record structure.

```
struct TSMTERec {
    TEHandle textH;
    TSMTEPreUpdateUPP preUpdateProc;
    TSMTEPostUpdateUPP postUpdateProc;
    long updateFlag;
    long refCon;
};
typedef struct TSMTERec TSMTERec;
typedef TSMTERec * TSMTERecPtr;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
TSMTE.h

TSMTERecHandle

Defines a handle to a TSMTE record pointer.

```
typedef TSMTERecPtr * TSMTERecHandle;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TSMTE.h`

WidthHookUPP

Defines a universal procedure pointer (UPP) to a width-hook callback.

```
typedef WidthHookProcPtr WidthHookUPP;
```

Discussion
For more information, see the description of the WidthHookUPP () callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`TextEdit.h`

Constants

Auto Idling Flag

Enables automatic idling in an event loop.

```
enum {
    teFIdleWithEventLoopTimer = 7
};
```

Auto Scroll Constant

Specifies automatic scrolling

```
enum {
        kTSMTEAutoScroll = 1
};
```

Do Text Selectors

Specify constants for identifying `TEDoTextSelectors`.

```
enum {
    teFind = 0,
    teHighlight = 1,
    teDraw = -1,
    teCaret = -2
};
```

Find Word Identification Constants

Specify constants for identifying the routine that called `FindWord`.

```
enum {
    teWordSelect = 4,
    teWordDrag = 8,
    teFromFind = 12,
    teFromRecal = 16
};
```

Hook Constants

Specify offsets into the `TEDispatchRec` data structure.

```
enum {
    EOLHook = 0,
    DRAWHook = 4,
    WIDTHHook = 8,
    HITTESTHook = 12,
    nWIDTHHook = 24,
    TextWidthHook = 28
};
```

Inline Input Flag

Specifies to use inline input service.

```
enum {
    teFUseTextServices = 4
};
```

Constants

`teFUseTextServices`

Use inline input service. This flag is no longer in use.

Available in Mac OS X v10.0 and later.

Declared in `TextEdit.h`.

Signature and Interface Constants

Specify a TSM TextEdit signature or interface.

```
enum {
    kTSMTESignature = 'tmTE',
    kTSMTEInterfaceType = 'tmTE'
};
```

Style Mode Constants

Used to set and replace style modes.

```
enum {
    fontBit = 0,
    faceBit = 1,
    sizeBit = 2,
    clrBit = 3,
    addSizeBit = 4,
    toggleBit = 5
};
```

Text Alignment Constants

Specify justification (word alignment) styles.

```
enum {
    teJustLeft = 0,
    teJustCenter = 1,
    teJustRight = -1,
    teForceLeft = -2,
    teFlushDefault = 0,
    teCenter = 1,
    teFlushRight = -1,
    teFlushLeft = -2
};
```

Constants

`teFlushDefault`
Align according to primary line direction
Available in Mac OS X v10.0 and later.
Declared in `TextEdit.h`.

`teCenter`
Centered for all scripts
Available in Mac OS X v10.0 and later.
Declared in `TextEdit.h`.

`teFlushRight`
Right aligned for all scripts
Available in Mac OS X v10.0 and later.
Declared in `TextEdit.h`.

`teFlushLeft`
Left aligned for all scripts
Available in Mac OS X v10.0 and later.
Declared in `TextEdit.h`.

Discussion
You can use these constants to specify the text alignment through the `align` parameter of the `TESetAlignment` (page 2992) and `TETextBox` (page 3000) functions. For compatibility, the previous names of these constants (`teJustLeft` , `teJustCenter` , `teJustRight` , and `teForceLeft`) are still supported.

Text Custom Hook Constants

Specify a selector for a TextEdit hook function.

```
enum {
    intEOLHook = 0,
    intDrawHook = 1,
    intWidthHook = 2,
    intHitTestHook = 3,
    intNWidthHook = 6,
    intTextWidthHook = 7,
    intInlineInputTSMTEPreUpdateHook = 8,
    intInlineInputTSMTEPostUpdateHook = 9
};
```

Constants

`intEOLHook`
End-of-line hook
Available in Mac OS X v10.0 and later.
Declared in `TextEdit.h`.

`intDrawHook`
Draw hook
Available in Mac OS X v10.0 and later.
Declared in `TextEdit.h`.

`intWidthHook`
Width measurement hook
Available in Mac OS X v10.0 and later.
Declared in `TextEdit.h`.

`intHitTestHook`
Hit test hook
Available in Mac OS X v10.0 and later.
Declared in `TextEdit.h`.

`intNWidthHook`
New width measurement hook
Available in Mac OS X v10.0 and later.
Declared in `TextEdit.h`.

`intTextWidthHook`
Text width measurement hook (low-memory global width measurement hook)
Available in Mac OS X v10.0 and later.
Declared in `TextEdit.h`.

`intInlineInputTSMTEPreUpdateHook`
Specifies a `TSMTEPreUpdateProcPtr` callback.

Available in Mac OS X v10.0 and later.

Declared in `TextEdit.h`.

`intInlineInputTSMTEPostUpdateHook`
Specifies a `TSMTEPostUpdateProcPtr` callback.

Available in Mac OS X v10.0 and later.

Declared in `TextEdit.h`.

Discussion
To specify a default TextEdit hook function with a customized function, you specify one of the following constants as the value of the `which` parameter to the `TECustomHook` (page 2972) function.

Text Feature Action Constants

Specify the action to be performed on a feature.

```
enum {
    teBitClear = 0,
    teBitSet = 1,
    teBitTest = -1
};
```

Constants
`teBitClear`
Disables the specified feature

Available in Mac OS X v10.0 and later.

Declared in `TextEdit.h`.

`teBitSet`
Enables the specified feature. If `teBitTest` returns `teBitSet`, the feature is enabled; if it returns `teBitClear`, it is disabled.

Available in Mac OS X v10.0 and later.

Declared in `TextEdit.h`.

`teBitTest`
Returns the current setting of the specified feature

Available in Mac OS X v10.0 and later.

Declared in `TextEdit.h`.

Discussion
To specify the action to be performed on a feature, you specify one of these constants as the value of the `action` parameter to the `TEFeatureFlag` (page 2976) function.

To test for the availability of these features, you can call the `Gestalt` function with the `gestaltTextEditVersion` selector. A result of `gestaltTE4` or greater returned in the response parameter indicates that outline highlighting and text buffering are available. A result of `gestaltTE5` or greater returned in the response parameter indicates that the two inline input features are available.

Version Notes
The inline input features are also available on version 6.0.7 systems with non-Roman script systems installed. However, there is no Gestalt constant that indicates this availability.

Text Feature Constants

Specify feature or bit definitions for the function `TEFeatureFlag`.

```
enum {
    teFAutoScroll = 0,
    teFTextBuffering = 1,
    teFOutlineHilite = 2,
    teFInlineInput = 3,
    teFUseWhiteBackground = 4,
    teFUseInlineInput = 5,
    teFInlineInputAutoScroll = 6
};
```

Constants

`teFAutoScroll`

Automatic scrolling. You can use the `TEFeatureFlag` function to turn automatic scrolling on and off as an alternative to calling `TEAutoView`. The effect is the same.

Available in Mac OS X v10.0 and later.

Declared in `TextEdit.h`.

`teFTextBuffering`

Text buffering. The `teFTextBuffering` selector enables or disables text buffering for performance improvements of 2-byte scripts. This is a global buffer, as opposed to the `TEKey` function's internal 2-byte buffer, and it is used across all active edit structures.

Exercise care when you enable the text-buffering capability in more than one active structure; otherwise, the bytes that are buffered from one edit structure may appear in another edit structure.

Ensure that buffering is not turned off in the middle of processing a 2-byte character. To guarantee the integrity of your structure, it is important that you wait for an idle event before you disable buffering or enable buffering in a second edit structure.

When text buffering is enabled, ensure that the `TEIdle` (page 2984) function is called before any pause of more than a few ticks—for example, before the Event Manager function `WaitNextEvent`. A possibility of a long delay before characters appear on the screen exists, especially in non-Roman systems. If you do not call `TEIdle`, the characters can end up in the edit structure of another application.

If text buffering is enabled on a non-Roman script system and the keyboard has changed, TextEdit flushes the text of the current script from the buffer before bringing characters of the new script into the buffer.

Available in Mac OS X v10.0 and later.

Declared in `TextEdit.h`.

`teFOutlineHilite`

Outline highlighting. The `teFOutlineHilite` selector specifies outline highlighting as the feature for which an action is to be performed. If a highlighted region exists in an edit structure and the window is inactive, then the highlighted region is outlined or framed.

In the case that outline highlighting is enabled and the current selection range is an insertion point, the caret is then drawn in a gray pattern so that it appears dimmed. To do the framing and caret dimming, TextEdit temporarily replaces the current address in the `highHook` and `caretHook` fields of the edit structure, redraws the caret or the highlighted region, and then immediately restores the hooks to their previous addresses.

Available in Mac OS X v10.0 and later.

Declared in `TextEdit.h`.

`teFInlineInput`

Inline input. You must deactivate an edit structure (using `TEDeactivate`) before changing the state of the feature bits or any fields in the edit structure.

In the future, other text services may use this same mechanism. If you follow the guidelines specified here, your application should also work with future text services. When an inline edit session begins, inline input also sets the `teFInlineInput` bit to provide the following features so that inline input works correctly with TextEdit: disabling font and keyboard synchronization, forcing a multiple-line selection to be highlighted line by line using a separate rectangle for each line rather than using a minimum number of rectangles for optimization, and highlighting a line only to the edge of the text rather than beyond the text to the edge of the view rectangle.

The `teFInlineInput` bit is cleared by inline input when an inline session ends. Use the `teFInlineInput` constant in the feature parameter of `TEFeatureFlag` to include these features in your application even when inline input is not installed. Be careful about changing the state of this bit if the `teFUseTextServices` bit is set. Again, the edit structure should always be deactivated before you change the state of the `teFInlineInput` bit. If you clear the `teFUseTextServices` bit and you set the `teFInlineInput` bit, inline input is disabled, but your application retains the features listed above.

Available in Mac OS X v10.0 and later.

Declared in `TextEdit.h`.

Discussion
To identify or adjust a feature, you specify one of these constants as the value of the `feature` parameter to the `TEFeatureFlag` (page 2976) function.

Text Styling Constants

Specify character attributes.

```
enum {
    doFont = 1,
    doFace = 2,
    doSize = 4,
    doColor = 8,
    doAll = 15,
    addSize = 16,
    doToggle = 32
};
```

Constants
`doFont`

Sets the font family ID

Available in Mac OS X v10.0 and later.

Declared in `TextEdit.h`.

`doFace`

Sets the character style

Available in Mac OS X v10.0 and later.

Declared in `TextEdit.h`.

`doSize`
Sets the type size
Available in Mac OS X v10.0 and later.
Declared in `TextEdit.h`.

`doColor`
Sets the color
Available in Mac OS X v10.0 and later.
Declared in `TextEdit.h`.

`doAll`
Sets all attributes
Available in Mac OS X v10.0 and later.
Declared in `TextEdit.h`.

`addSize`
Increases or decreases the current type size
Available in Mac OS X v10.0 and later.
Declared in `TextEdit.h`.

`doToggle`
Modifies the mode
Available in Mac OS X v10.0 and later.
Declared in `TextEdit.h`.

Discussion
You can use these constants (singly or in combination) to specify character attributes, through the `mode` parameter of the `TEContinuousStyle` (page 2970) , `TESetStyle` (page 2996) , and `TEReplaceStyle` (page 2989) functions.

Result Codes

In addition to `noErr`, the most common result code returned by TextEdit is listed below.

Result Code	Value	Description
`noScrapErr`	-100	Scrap does not exist (not initalized). Available in Mac OS X v10.0 and later.

Document Revision History

This table describes the changes to *Carbon Framework Reference*.

Date	Notes
2007-10-31	Added three reference documents (HIGeometry, Carbon Printing, Control Manager) and removed two (Carbon Printing Manager, ColorSync Manager).
2006-05-23	First publication of this content as a collection of previously published documents.

Document Revision History

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z