Control Manager Reference

Carbon > User Experience

2007-03-26

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

Simultaneously published in the United States and Canada.

Contents

Control Manager Reference

Framework:	Carbon/Carbon.h
Declared in	ControlDefinitions.h Controls.h HIButtonViews.h HIClockView.h HIContainerViews.h HIDataBrowser.h HIDisclosureViews.h HIImageViews.h HILittleArrows.h HIObject.h HIPopupButton.h HIProgressViews.h HIRelevanceBar.h HIScrollView.h HISeparator.h HISlider.h HITabbedView.h HITextViews.h HIToolboxDebugging.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 32)
Creates a bevel button control.

`CreateChasingArrowsControl` (page 33)
Creates a chasing arrows control.

`CreateCheckBoxControl` (page 33)
Creates a checkbox control.

`CreateCheckGroupBoxControl` (page 34)
Creates a group box control that has a check box as its title.

`CreateClockControl` (page 35)
Creates a clock control.

`CreateDisclosureButtonControl` (page 36)
Creates a new instance of the Disclosure Button Control.

`CreateDisclosureTriangleControl` (page 37)
Creates a disclosure triangle control.

`CreateEditUnicodeTextControl` (page 38)
Creates a new edit text control.

`CreateGroupBoxControl` (page 39)
Creates a group box control.

`CreateIconControl` (page 40)
Creates an icon control.

`CreateImageWellControl` (page 41)
Creates an image well control.

`CreateListBoxControl` (page 41)
Creates a list box control.

`CreateLittleArrowsControl` (page 43)
Creates a little arrows control.

`CreatePictureControl` (page 44)
Creates a picture control.

`CreatePlacardControl` (page 45)
Creates a placard control.

`CreatePopupArrowControl` (page 45)
Creates a pop-up arrow control.

`CreatePopupButtonControl` (page 46)
Creates a pop-up button control.

`CreatePopupGroupBoxControl` (page 47)
Creates a group box control that has a pop-up button as its title.

`CreateProgressBarControl` (page 49)
Creates a progress bar control.

`CreatePushButtonControl` (page 49)
Creates a push button control.

`CreatePushButtonWithIconControl` (page 50)
Creates a push button control containing an icon or other graphical content.

`CreateRadioButtonControl` (page 51)
Creates a radio button control.

`CreateRadioGroupControl` (page 52)
Creates a radio group control.

`CreateRelevanceBarControl` (page 52)
Creates a relevance bar control.

`CreateRoundButtonControl` (page 55)
Creates a new instance of the round button control.

`CreateScrollBarControl` (page 55)
Creates a scroll bar control.

`CreateSeparatorControl` (page 58)
Creates a separator control.

`CreateSliderControl` (page 58)
Creates a slider control.

`CreateStaticTextControl` (page 60)
Creates a new static text control.

`CreateTabsControl` (page 60)
Creates a tabs control.

`CreateUserPaneControl` (page 62)
Creates a user pane control.

`CreateWindowHeaderControl` (page 62)
Creates a window header control.

`DisposeControl` (page 64)
Decrements a control's reference count and destroys it if the reference count becomes 0.

`GetNewControl` (page 100)
Creates a control from a control resource.

`KillControls` (page 120)
Removes all of the controls from a window that you wish to keep.

`RegisterControlDefinition` (page 126)
Registers an old-style control definition.

`CreateCustomControl` (page 314) 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 28)
Automatically embeds a control in the smallest appropriate embedder control.

`CountSubControls` (page 31)
Obtains the number of embedded controls within a control.

`CreateRootControl` (page 53)
Creates the root control for a specified window.

`DumpControlHierarchy` (page 73)
Writes a textual representation of the control hierarchy for a specified window into a file.

`EmbedControl` (page 73)
Embeds one control inside another.

`GetControlOwner` (page 91)
Returns the window to which a control is bound.

`GetIndexedSubControl` (page 98)
Obtains a handle to a specified embedded control.

`GetRootControl` (page 101)
Obtains a handle to a window's root control.

`GetSuperControl` (page 101)
Obtains a handle to an embedder control.

Displaying Controls

`DisableControl` (page 64)
Disables a control.

`Draw1Control` (page 71)
Draws a control and any embedded controls that are currently visible in the specified window.

DrawControlInCurrentPort (page 71)
Draws a control in the current graphics port.

DrawControls (page 72)
Draws all controls currently visible in the specified window.

GetControlViewSize (page 97)
Obtains the size of the content to which a control's size is proportioned.

IsControlVisible (page 119)
Determines whether a control is visible.

SetControlViewSize (page 147)
Informs the Control Manager of the size of the content to which a control's size is proportioned.

SetUpControlBackground (page 151)
Applies the proper background color for the given control to the current port.

SetUpControlTextColor (page 152)
Applies the proper text color for the given control to the current port.¬¬

UpdateControls (page 156)
Draws controls in the specified update region of a window.

HideControl (page 107)
Makes a control, and any latent embedded controls, invisible.

ShowControl (page 153)
Makes an invisible control, and any latent embedded controls, visible.

SetControlVisibility (page 147)
Sets the visibility of a control, and any embedded controls, and specifies whether it should be drawn.

Handling Events in Controls

FindControl (page 75)
Obtains the location of a mouse-down event in a control.

FindControlUnderMouse (page 76)
Obtains the location of a mouse-down event in a control.

GetControlAction (page 82)
Returns a pointer to the action function associated with a control structure.

GetControlClickActivation (page 84)
Gets the control's preferred behavior for responding to particular click.

GetControlCommandID (page 84)
Gets the command ID for a control.

HandleControlClick (page 103)
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 104)
Allows a control to display a contextual menu.

HandleControlDragReceive (page 105)
Tells a control to accept the data from a drag.

HandleControlDragTracking (page 105)
Tells a control to respond visually to a drag.

`HandleControlSetCursor` (page 106)
Requests that a control set the cursor based on the mouse location.

`IsAutomaticControlDragTrackingEnabledForWindow` (page 117)
Indicates whether automatic drag tracking is enabled for the specified window.

`IsControlDragTrackingEnabled` (page 118)
Indicates whether a control's drag track and receive support is enabled.

`SetAutomaticControlDragTrackingEnabledForWindow` (page 129)
Enables or disables automatic drag tracking for a window.

`SetControlAction` (page 135)
Sets the action function for a control.

`SetControlCommandID` (page 137)
Sets the command ID for a control.

`SetControlDragTrackingEnabled` (page 139)
Sets the drag tracking state for a control.

`TestControl` (page 154)
Obtains the control part in which a mouse-down event occurred.

`HandleControlKey` (page 316) 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 27)
Activates a control and any latent embedded controls.

`DeactivateControl` (page 63)
Deactivates a control and any latent embedded controls.

`EnableControl` (page 75)
Enables a control.

`GetControlRegion` (page 95)
Obtains the region corresponding to a given control part.

`IsControlActive` (page 117)
Returns whether a control is active.

`IsControlEnabled` (page 118)
Indicates whether a control is enabled.

`IsControlHilited` (page 119)
Indicates whether or not the control is highlighted.

`MoveControl` (page 121)
Moves a control within its window.

`SizeControl` (page 154)
Changes the size of a control's rectangle.

Handling Keyboard Focus

`AdvanceKeyboardFocus` (page 28)
Advances the keyboard focus to the next focusable control in a window.

`ClearKeyboardFocus` (page 30)
Removes the keyboard focus for the currently focused control in a window.

`GetKeyboardFocus` (page 99)
Obtains a handle to the control with the current keyboard focus for a specified window.

`ReverseKeyboardFocus` (page 127)
Returns keyboard focus to the prior focusable control in a window.

`SetKeyboardFocus` (page 150)
Sets the current keyboard focus to a specified control part for a window.

Accessing Control Settings and Data

`ChangeControlPropertyAttributes` (page 29)
Changes a property attribute.

`CopyControlTitleAsCFString` (page 30)
Makes a copy of the control's title as a Core Foundation string.

`GetBestControlRect` (page 77)
Obtains a control's optimal size and text placement.

`GetControl32BitMaximum` (page 80)
Obtains the maximum setting of a control.

`GetControl32BitMinimum` (page 80)
Obtains the minimum setting of a control.

`GetControl32BitValue` (page 81)
Obtains the current setting of a control.

`GetControlBounds` (page 82)
Gets the bounds of a control.

`GetControlByID` (page 83)
Finds a control in a window by its unique ID.

`GetControlData` (page 85)
Obtains control-specific data.

`GetControlDataSize` (page 86)
Obtains the size of a control's tagged data.

`GetControlHilite` (page 88)
Gets the highlight status of a control.

`GetControlID` (page 88)
Gets the control ID for a control.

`GetControlKind` (page 89)
Returns the kind of the given control.

`GetControlProperty` (page 92)
Obtains a piece of data that has been previously associated with a control.

GetControlPropertySize (page 94)
Obtains the size of a piece of data that has previously been associated with a control.

GetControlReference (page 95)
Obtains a control's current reference value.

GetImageWellContentInfo (page 98)
Gets information about the content of an image well.

GetControlPropertyAttributes (page 93)
Gets the property attributes for a control.

GetTabContentRect (page 102)
Gets the content rectangle for a tab.

IsValidControlHandle (page 120)
Reports whether a given handle is a control handle.

RemoveControlProperty (page 127)
Removes a piece of data that has been previously associated with a control.

SetControl32BitMaximum (page 133)
Changes the maximum setting of a control and, if appropriate, redraws it accordingly.

SetControl32BitMinimum (page 133)
Changes the minimum setting of a control and, if appropriate, redraws it accordingly.

SetControl32BitValue (page 134)
Changes the current setting of a control and redraws it accordingly.

SetControlData (page 137)
Sets control-specific data.

SetControlBounds (page 135)
Sets the bounds of a control.

SetControlID (page 140)
Sets a control's ID.

SetControlTitleWithCFString (page 145)
Sets the title for a control to the specified Core Foundation string.

SetDisclosureTriangleLastValue (page 148)
Sets the last value of a disclosure triangle.

SetImageWellContentInfo (page 149)
Sets the content information for an image well.

SetImageWellTransform (page 149)
Sets an image well transform.

SetTabEnabled (page 150)
Enables and disables a tab control.

SetControlFontStyle (page 140)
Sets the font style for a control.

SetControlProperty (page 143)
Associates data with a control.

SetControlReference (page 144)
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 91)
Gets the menu handle for a pop-up control.

`GetControlPopupMenuID` (page 92)
Gets the menu ID of a pop-up menu.

`SetControlPopupMenuHandle` (page 142)
Sets the menu handle for a pop-up control.

`SetControlPopupMenuID` (page 143)
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 78)
Gets the content information for a bevel button.

`GetBevelButtonMenuHandle` (page 79)
Gets the menu handle for a bevel button.

`GetBevelButtonMenuValue` (page 79)
Gets the value of a bevel button menu.

`SetBevelButtonContentInfo` (page 129)
Sets the content information for a bevel button.

`SetBevelButtonGraphicAlignment` (page 130)
Sets the alignment for a bevel button.

`SetBevelButtonMenuValue` (page 131)
Sets the value of a bevel button menu.

`SetBevelButtonTextAlignment` (page 131)
Sets the alignment of the text for a bevel button.

`SetBevelButtonTextPlacement` (page 132)
Sets the placement for bevel button text.

`SetBevelButtonTransform` (page 132)
Sets the transform for a bevel button.

Managing Control UPPs

`DisposeControlActionUPP` (page 65)
Disposes of a control action UPP.

`DisposeControlCNTLToCollectionUPP` (page 65)
Disposes of a CNLT to collection UPP.

`DisposeControlEditTextValidationUPP` (page 66)
Disposes of an edit text validation UPP.

`InvokeEditUnicodePostUpdateUPP` (page 116)
Invokes a Unicode post update UPP.

`NewControlActionUPP` (page 122)
Creates a UPP for a control action callback function.

`NewControlCNTLToCollectionUPP` (page 122)
Creates a UPP for a control-to-collection callback function.

`NewControlEditTextValidationUPP` (page 123)
Creates a UPP for a control edit text validation callback function.

`NewControlKeyFilterUPP` (page 123)

`NewControlUserPaneActivateUPP` (page 123)

`NewControlUserPaneBackgroundUPP` (page 124)

`NewControlUserPaneDrawUPP` (page 124)

`NewControlUserPaneFocusUPP` (page 124)

`NewControlUserPaneHitTestUPP` (page 124)

`NewControlUserPaneIdleUPP` (page 125)

`NewControlUserPaneKeyDownUPP` (page 125)

`NewControlUserPaneTrackingUPP` (page 125)

`NewEditUnicodePostUpdateUPP` (page 126)

`DisposeControlDefUPP` (page 315) 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 317) Deprecated in Mac OS X v10.5
Invokes a control definition UPP. (Deprecated. Use a custom HIView to draw a custom control.)

`NewControlDefUPP` (page 319) 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 57)
Creates a scrolling text box control.

`DisposeControlColorUPP` (page 66)

`GetControlFeatures` (page 87)
Obtains the features a control supports.

`GetControlMaximum` (page 90)
Obtains a control's maximum setting. (Deprecated. Use `GetControl32BitMaximum` (page 80) instead.)

`GetControlMinimum` (page 90)
Obtains a control's minimum setting. (Deprecated. Use `GetControl32BitMinimum` (page 80) instead.)

`GetControlValue` (page 96)
Obtains a control's current setting. (Deprecated. Use `GetControl32BitValue` (page 81) instead.)

`GetControlVariant` (page 97)
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 110)

`NewControlColorUPP` (page 122)

`SetControlColorProc` (page 136)
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 86)
Obtains a handle to control-specific data. (Deprecated. Use custom HIViews instead of custom CDEFs. See *HIView Programming Guide*.)

`SetControlDataHandle` (page 138)
(Deprecated. Use custom HIViews instead of custom CDEFs. See *HIView Programming Guide*.)

`SetControlMaximum` (page 141)
Changes the maximum setting of a control and redraws its indicator or scroll box accordingly. (Deprecated. Use `SetControl32BitMaximum` (page 133) instead.)

`SetControlMinimum` (page 141)
Changes the minimum setting of a control and redraws its indicator or scroll box accordingly. (Deprecated. Use `SetControl32BitMinimum` (page 133) instead.)

`SetControlSupervisor` (page 144)
Routes mouse-down events to the embedder control.

`SetControlValue` (page 146)
Changes the current setting of a control and redraws it accordingly. (Deprecated. Use `SetControl32BitValue` (page 134) instead.)

`TrackControl` (page 155)
Responds to cursor movements in a control while the mouse button is down. (Deprecated. Use `HandleControlClick` (page 103) instead.)

`DragControl` (page 70)
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 108)
Changes the highlighting of a control.

`SendControlMessage` (page 128)
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 316) Deprecated in Mac OS X v10.5
Obtains the title of a control. (Deprecated. Use `HIViewCopyText` or `CopyControlTitleAsCFString` (page 30) instead.)

`NewControl` (page 318) 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 33)).)

`SetControlTitle` (page 320) Deprecated in Mac OS X v10.5
Changes the title of a control and redraws the control accordingly. (Deprecated. Use `HIViewSetText` or `SetControlTitleWithCFString` (page 145) instead.)

`CreateEditTextControl` (page 313) Deprecated in Mac OS X v10.4
Creates a new edit text control. (Deprecated. Use `CreateEditUnicodeTextControl` (page 38) instead.)

`IdleControls` (page 314) 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 193).

Return Value

A result code. See "Control Manager Result Codes" (page 308).

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 308).

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 161) 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 308).

Discussion

The Dialog Manager uses `AutoEmbedControl` (page 28) 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 73).

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 193).

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 308).

Discussion
If you have associated control properties with a control (by calling `SetControlProperty` (page 143), 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 308).

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 161) 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 193).

outString

A copy of the control's title.

Return Value

A result code. See "Control Manager Result Codes" (page 308).

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 193).

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 308).

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 98) 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 237).

behavior
The behavior the button is to have. For possible values, see "Bevel Button Behavior Constants" (page 205).

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 211).

menuPlacement
The placement of the menu. For possible values, see "Control Bevel Button Menu Placement Constants" (page 237).

outControl
On return, `outControl` points to the new control. For a description of this data type, see ControlRef (page 193).

Return Value
A result code. See "Control Manager Result Codes" (page 308).

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 193).

Return Value
A result code. See "Control Manager Result Codes" (page 308).

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 193).

Return Value
A result code. See "Control Manager Result Codes" (page 308).

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 193).

Return Value

A result code. See "Control Manager Result Codes" (page 308).

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 237).

clockFlags

Clock options. For possible values, see "Clock Value Flag Constants" (page 218).

outControl

On return, `outControl` points to the new control. For a description of this data type, see ControlRef (page 193).

Return Value

A result code. See "Control Manager Result Codes" (page 308).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`HIClockView.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 193).

Return Value

A result code. See "Control Manager Result Codes" (page 308).

Discussion

`CreateDisclosureButtonControl` is preferred over `NewControl` (page 318) 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 237).

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 193).

Return Value
A result code. See "Control Manager Result Codes" (page 308).

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`

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 193).

Return Value
A result code. See "Control Manager Result Codes" (page 308).

Discussion
This function is the preferred way of creating edit text controls. Use it instead of the `CreateEditTextControl` (page 313) 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 193).

Return Value
A result code. See "Control Manager Result Codes" (page 308).

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 193).

Return Value
A result code. See "Control Manager Result Codes" (page 308).

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 193).

Return Value
A result code. See "Control Manager Result Codes" (page 308).

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 193).

Return Value

A result code. See "Control Manager Result Codes" (page 308).

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 85) 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 193).

Return Value
A result code. See "Control Manager Result Codes" (page 308).

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 137) 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 308).

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 193).

Return Value
A result code. See "Control Manager Result Codes" (page 308).

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 193).

Return Value

A result code. See "Control Manager Result Codes" (page 308).

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 193).

Return Value
A result code. See "Control Manager Result Codes" (page 308).

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 193).

Return Value

A result code. See "Control Manager Result Codes" (page 308).

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 193).

Return Value
A result code. See "Control Manager Result Codes" (page 308).

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 193).

Return Value

A result code. See "Control Manager Result Codes" (page 308).

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 237).

outControl

On return, the new control. For a description of this data type, see ControlRef (page 193).

Return Value
A result code. See "Control Manager Result Codes" (page 308).

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 308).

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 193).

Return Value
A result code. See "Control Manager Result Codes" (page 308).

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 98), 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 193).

Return Value

A result code. See "Control Manager Result Codes" (page 308).

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 308).

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 101).

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 318) or `GetNewControl` (page 100). You can specify that a specific control be embedded into another by calling `EmbedControl` (page 73).

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 193).

Return Value

A result code. See "Control Manager Result Codes" (page 308).

Discussion

`CreateRoundButtonControl` is preferred over `NewControl` (page 318) 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 193).

Return Value

A result code. See "Control Manager Result Codes" (page 308).

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 308).

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 318), unless changed by calling `SetControlData` (page 137).

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 193).

Return Value

A result code. See "Control Manager Result Codes" (page 308).

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 238).

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 193).

Return Value
A result code. See "Control Manager Result Codes" (page 308).

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 318), use the `kControlSliderLiveFeedback` variant.

2. Write an appropriate ControlActionProc and associate it with your slider by calling `SetControlAction` (page 135). This allows your application to update its content appropriately when the live-tracking slider is clicked.

3. When calling`HandleControlClick` (page 103) or `TrackControl` (page 155) 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 193).

Return Value
A result code. See "Control Manager Result Codes" (page 308).

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 238) for possible values.

direction

The control tab direction. See "`Control Tab Direction Constants`" (page 238) 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 193).

Return Value

A result code. See "Control Manager Result Codes" (page 308).

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 225).

outControl

On return, the new control. For a description of this data type, see ControlRef (page 193).

Return Value

A result code. See "Control Manager Result Codes" (page 308).

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 193).

Return Value
A result code. See "Control Manager Result Codes" (page 308).

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 308).

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 193).

Return Value
A result code. See "Control Manager Result Codes" (page 308).

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 193).

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 120). If an embedding hierarchy is present, passing the root control to the `DisposeControl` function is the effectively the same as calling `KillControls` (page 120). 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`

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 193).

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 155) 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 193).

Discussion
Although you should generally use the functionUpdateControls (page 156) 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 193).

Discussion
Typically, controls are automatically drawn in their owner's graphics port with the functions `DrawControls` (page 72), `Draw1Control` (page 71), and `UpdateControls` (page 156). `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 156) 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 308).

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 193).

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 193).

Return Value

A result code. See "Control Manager Result Codes" (page 308).

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 53) function or by setting the appropriate dialog flag. The root control can be retrieved by calling `GetRootControl` (page 101).

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 318) or `GetNewControl` (page 100). 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 308).

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 193).

Return Value

The control part code of the control in which the mouse-down event occurred; see "Control Meta Part Code Constants" (page 274), "Control Part Code Constants " (page 232), and" Control State Part Code Constants" (page 235). For a description of this data type, see ControlPartCode (page 192).

Discussion
The `FindControl` function is not recommended when the Appearance Manager is available. When the Appearance Manager is available, you should call `FindControlUnderMouse` (page 76) 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 155) 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 232), "Control Part Code Constants " (page 232), and " Control State Part Code Constants" (page 235).

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 193).

Discussion

You should call the `FindControlUnderMouse` function instead of `FindControl` (page 75) 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 308).

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 308).

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 308).

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 308).

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 193).

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 133).

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 193).

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 133).

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 134).

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 159) 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 103) and `TrackControl` (page 155). You can use the function `SetControlAction` (page 135) 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 193).

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 193).

Return Value
A result code. See "Control Manager Result Codes" (page 308).

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 264).

Return Value

A result code. See "Control Manager Result Codes" (page 308).

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 308).

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 274)The part code of the control part from which data is to be obtained; see , "Control Part Code Constants " (page 232), and " Control State Part Code Constants" (page 235). 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 192).

inTagName

A constant representing the control-specific data you wish to obtain see the data tag constants in the "Control Manager Constants" (page 203) 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 86) 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 86). 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 308). 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 138).

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 193).

inPart

The part code of the control part with which the data is associated; see"Control Meta Part Code Constants" (page 274), "Control Part Code Constants " (page 232), and " Control State Part Code Constants" (page 235). 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 203) 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 137) and `GetControlData` (page 85) to allocate a sufficiently large buffer for variable-length data.

Return Value

A result code. See "Control Manager Result Codes" (page 308). 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 137) and `GetControlData` (page 85) 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 193).

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 161).

Return Value
A result code. See "Control Manager Result Codes" (page 308). 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 193).

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 193).

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 308).

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 193).

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 191).

Return Value
A result code. See "Control Manager Result Codes" (page 308).

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 80) 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 193).

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 318). You can change the maximum setting by using the function SetControlMaximum (page 141).

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 80) 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 318). You can change the minimum setting by using the function `SetControlMinimum` (page 141).

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 193).

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 94) 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 308).

Discussion
You may use the function `GetControlProperty` to obtain a copy of data previously set by your application with the function `SetControlProperty` (page 143).

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 193).

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 308).

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 193).

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 308).

Discussion
If you want to retrieve a piece of associated data with the function `GetControlProperty` (page 92), 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 318). 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 144) 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 274), "Control Part Code Constants " (page 232), and " Control State Part Code Constants" (page 235) 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 308).

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

Declared In
`Controls.h`

GetControlValue

Obtains a control's current setting. (**Deprecated.** Use `GetControl32BitValue` (page 81) 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 318). You can change the setting by calling `SetControlValue` (page 146).

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 203) section for descriptions of control variation codes. For a description of this data type, see ControlVariant (page 196).

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 308).

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 193).

inIndex

A one-based index—an integer between 1 and the value returned in the `outNumChildren` parameter of `CountSubControls` (page 31)—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 308). 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 308).

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 193).

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 153) 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 308).

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 64), `ActivateControl` (page 27),and `DeactivateControl` (page 63) 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 193).

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 193).

Return Value

A result code. See "Control Manager Result Codes" (page 308).

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 308).

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 159) 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 274), "Control Part Code Constants " (page 232), and" Control State Part Code Constants" (page 235). For a description of this data type, see ControlPartCode (page 192).

Discussion

Call the `HandleControlClick` function after a call to `FindControl` (page 75) or `FindControlUnderMouse` (page 76). The `HandleControlClick` function should be called instead of `TrackControl` (page 155) 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 82) to get a pointer to the control action function that is associated with the control, and you can call `SetControlAction` (page 135) 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 308).

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 308).

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 139) 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 105) 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 308).

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 139) 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`

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 308).

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 153) was called on its embedder control.

To make the control visible again, call ShowControl (page 153).

You can also call SetControlVisibility (page 147) 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 193).

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 232), "Control Part Code Constants " (page 232), and " Control State Part Code Constants" (page 235).

Discussion
If the Appearance Manager is available, you should call the functions ActivateControl (page 27) and DeactivateControl (page 63) 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 107), you should use HiliteControl to make all controls inactive when their windows are not frontmost. The function TrackControl (page 155) 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`

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 193).

partCode

The part code for which the control action UPP is to be invoked. For possible values, see "Control Meta Part Code Constants" (page 274), "Control Part Code Constants " (page 232), and " Control State Part Code Constants" (page 235).

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 308).

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 308).

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`

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 193).

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 193).

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 191).

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 182).

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 192).

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 192) 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 192).

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 192).

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 308).

Discussion
For more information on automatic drag tracking, see `SetAutomaticControlDragTrackingEnabledForWindow` (page 129).

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 308).

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 64). 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`

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 159) 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

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 308).

Discussion
Mac OS X does not allow you to store custom control definitions in resources. However, some older functions such as `GetNewControl` (page 100) 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 252) 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 308).

Discussion
Your application may dissociate data it has previously set with the `SetControlProperty` (page 143) 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 308).

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 161) 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 193).

inMessage
A bit field representing the message(s) you wish to send; see `ControlDefProcPtr` (page 161).

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 87) 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 308).

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 105) and `HandleControlDragReceive` (page 105) 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 308).

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 210).

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 308).

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 308).

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 213).

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 308).

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 214).

Return Value

A result code. See "Control Manager Result Codes" (page 308).

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 308).

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 193).

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 80).

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 193).

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 80).

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 193).

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 81).

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 159) 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 103) and `TrackControl` (page 155) 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 193).

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 308). 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 151) to have its background color set up by any parent controls. Similarly, any Control that draws text calls `SetUpControlTextColor` (page 152) 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 308).

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 274), "Control Part Code Constants " (page 232), and " Control State Part Code Constants" (page 235). 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 203) 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 86) 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 308). 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 203) 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 308).

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 193).

inStyle

A pointer to a `ControlFontStyleRec` (page 188) 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 308).

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 308).

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 133) 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 318). To determine a control's current maximum setting, use the function `GetControlMaximum` (page 90).

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 133) 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 193).

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 318) function. To obtain a control's current minimum setting, use the function `GetControlMinimum` (page 90).

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 193).

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 308).

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 193).

`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 318). Call `GetControlReference` (page 95) 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 193).

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 193).

Return Value
A result code. See "Control Manager Result Codes" (page 308).

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 155) or `HandleControlClick` (page 103) 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`

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 308).

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 134) 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 193).

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 216) and "Radio Button Value Constants" (page 290). 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 318). 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 96).

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 193).

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 308).

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 256).

Return Value
A result code. See "Control Manager Result Codes" (page 308).

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 308).

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 308).

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 187)

Return Value

A result code. See "Control Manager Result Codes" (page 308).

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 73).

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 308).

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 193).

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 308). 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 308). 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 193).

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 318).
- Calling `HideControl` (page 107).
- Calling `SetControlVisibility` (page 147).

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 274), "Control Part Code Constants " (page 232), and" Control State Part Code Constants" (page 235). If the control is invisible or inactive, `TestControl` returns 0. For a description of this data type, see ControlPartCode (page 192).

Discussion
The `TestControl` function is called by the `FindControl` (page 75) and `TrackControl` (page 155) 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 103) 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 193).

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 159) 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 274), "Control Part Code Constants " (page 232), and" Control State Part Code Constants" (page 235). `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 192).

Discussion
When the Appearance Manager is available, you should typically call `HandleControlClick` (page 103) 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 72) 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 159)

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 161)

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 126) or create it directly using `CreateCustomControl` (page 314).

Defining Your Own Key Filter Function

`ControlKeyFilterProcPtr` (page 170)
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 169)
Ensures that the content of an editable text control is valid.

Defining Your Own User Pane Functions

`ControlUserPaneActivateProcPtr` (page 171)
Handles activate and deactivate event processing.

`ControlUserPaneBackgroundProcPtr` (page 172)
Sets the background color or pattern for user panes that support embedding.

`ControlUserPaneDrawProcPtr` (page 174)
Draws the content of your user pane control in the rectangle of user pane control.

`ControlUserPaneFocusProcPtr` (page 175)
Handles keyboard focus.

`ControlUserPaneHitTestProcPtr` (page 176)
Returns the part code of the control that the point was in when the mouse-down event occurred.

`ControlUserPaneIdleProcPtr` (page 176)
Performs idle processing.

`ControlUserPaneKeyDownProcPtr` (page 177)
Handles keyboard event processing.

`ControlUserPaneTrackingProcPtr` (page 179)
Tracks a control while the user holds down the mouse button.

Miscellaneous

`ControlCNTLToCollectionProcPtr` (page 160)

`ControlColorProcPtr` (page 161)

`EditUnicodePostUpdateProcPtr` (page 180)

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 193).

partCode

The control part in which the mouse-down event occurred; see "Control Meta Part Code Constants" (page 274), "Control Part Code Constants " (page 232), and " Control State Part Code Constants" (page 235). 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 122). 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 103) and `TrackControl` (page 155) 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 135) 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 308).

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 308).

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 126) or create it directly using `CreateCustomControl` (page 314).

```
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 219) 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 186).

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 319). 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 127) 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 30), `SetKeyboardFocus` (page 150), and "Control Meta Part Code Constants" (page 274), 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 64) 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 235) or `ControlKeyDownRec` (page 190) 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 274) "Control Part Code Constants " (page 232) and " Control State Part Code Constants" (page 235).

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 274).

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 193).

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 240), 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 123). 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 266). For a description of this data type, see ControlKeyFilterResult (page 191).

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 123). 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 123). 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 137) 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 137).

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 182).

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 124). 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 137) 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 137).

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 193).

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 124). 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 137) 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 137).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 161).

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 192).

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 124). 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 137) 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 137).

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 193).

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 192).

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 124). 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 137) 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 193).

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 125). 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 137) 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 137).

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 193).

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 192).

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 125). 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 137) 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 137).

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 193).

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 159).

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 192).

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 125). 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 137) 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 137).

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 219) 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 254) 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 185).

`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 137) and obtained from `GetControlData` (page 85), in conjunction with the `kControlBevelButtonContentTag` and `kControlImageWellContentTag` constants; see "Bevel Button Control Data Tag Constants" (page 207) and "Image Well Control Data Tag Constants" (page 249).

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 293) 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 85) and `SetControlData` (page 137) 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 271) and "Control Font Style Flag Constants" (page 229). 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 275).

`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 275).

`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 140) 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 232), "Control Part Code Constants " (page 232), and " Control State Part Code Constants" (page 235). For a description of this data type, see ControlPartCode (page 192).

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 219) 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 137) and `GetControlData` (page 85) 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 159) 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 276).

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 85) 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 161).

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 318) 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 318) 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 211).

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 211).

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 211).

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 211).

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 211).

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 211).

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 137) and `GetControlData` (page 85) 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 86) 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 161).

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 137) and returned by `GetControlData` (page 85).

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 318) 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 137) and `GetControlData` (page 85) 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 86) 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 137) and returned by `GetControlData` (page 85).

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 137) and returned by `GetControlData` (page 85). 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 81) 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 81) 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 161).

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 146) and are returned by `GetControlValue` (page 96) .

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 137) and `GetControlData` (page 85) 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 86) 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 161).

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 318). 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 189) 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 189) 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 161).

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 219).

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 137) and `GetControlData` (page 85) 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 86) 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 275).

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 275).

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 275).

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 188). 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 240).

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 108)to disable the control.

Available in Mac OS X v10.0 and later.

Declared in `Controls.h`.

`kControlInactivePart`

Used with `HiliteControl` (page 108) 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 169) 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 169), 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 137) and `GetControlData` (page 85) 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 86) 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 240).

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 161).

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 137) and `GetControlData` (page 85) 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 86) 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 161).

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 137) and `GetControlData` (page 85) 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 86) 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 183) data type and "Control Content Type Constants" (page 254) 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 183) data type and "Control Content Type Constants" (page 254) 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 161).

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 183).

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 137) and `GetControlData` (page 85) 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 86) 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 161).

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 314). 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 207). 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 85) and `SetControlData` (page 137) 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 84).

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 137) and `GetControlData` (page 85) 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 86) 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 161).

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 161).

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 229) and the ControlFontStyleRec (page 188) 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 161).

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 183).

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 95) 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 188) 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 161).

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 161).

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 161).

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 273)) 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 137) and `GetControlData` (page 85) 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 86) 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 161).

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 137) and `GetControlData` (page 85) 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 86) 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 288). 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 161).

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 146) and are returned by `GetControlValue` (page 96).

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 161).

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 161).

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 161).

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 161).

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 135) 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 161).

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 137) and `GetControlData` (page 85) 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 86) 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 161).

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 137) and `GetControlData` (page 85) 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 86) 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 161).

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 194).

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 137) and `GetControlData` (page 85) 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 86) 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 137) and `GetControlData` (page 85) 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 86) 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 161).

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 172). 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 172).

  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 172).

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 172).

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 172).

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 172).

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 172).

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 172).

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 137) and `GetControlData` (page 85) 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 86) 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 161).

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 161).

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.

Deprecated Control Manager Functions

A function identified as deprecated has been superseded and may become unsupported in the future.

Deprecated in Mac OS X v10.4

CreateEditTextControl

Creates a new edit text control. (Deprecated in Mac OS X v10.4. Use `CreateEditUnicodeTextControl` (page 38) 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 308).

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`

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 161) 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`

Deprecated in Mac OS X v10.5

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 252).

outControl

On return, `outControl` points to the new control.

Return Value

A result code. See "Control Manager Result Codes" (page 308).

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`

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`

GetControlTitle

Obtains the title of a control. (Deprecated in Mac OS X v10.5. Use `HIViewCopyText` or `CopyControlTitleAsCFString` (page 30) 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 318). You can change the title by using `SetControlTitle` (page 320).

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`

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 274), "Control Part Code Constants " (page 232), and" Control State Part Code Constants" (page 235). For a description of this data type, see ControlPartCode (page 192).

Discussion

If you have determined that a keyboard event has occurred in a given window, before calling the `HandleControlKey` function, call `GetKeyboardFocus` (page 99) 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`

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 193).

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`

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 33)).)

```
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 153) 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 216) and "Radio Button Value Constants" (page 290). 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 216) or "Radio Button Value Constants" (page 290)). 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 216) or "Radio Button Value Constants" (page 290)). 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 193).

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 100) 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 126). 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`

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`

SetControlTitle

Changes the title of a control and redraws the control accordingly. (Deprecated in Mac OS X v10.5. Use `HIViewSetText` or `SetControlTitleWithCFString` (page 145) 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 318). To determine a control's current title, use the function `GetControlTitle` (page 316).

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`

Document Revision History

This table describes the changes to *Control Manager Reference*.

Date	Notes
2007-03-26	Made minor formatting changes.
2006-07-24	Added deprecation information.
2006-01-10	Updated for Mac OS X v10.4.
	Added descriptions of the `CreatePopupArrowControl`, `CreatePlacardControl`, `CreateStaticTextControl`, `CreatePictureControl`, `CreateIconControl`, `CreatePushButtonControl`, `CreatePushButtonWithIconControl`, `CreateRadioButtonControl`, `CreateCheckBoxControl`, `CreateScrollBarControl`, `CreatePopupButtonControl`, `CreateEditUnicodeTextControl` functions.
	Also added descriptions of the following constants: `kControlBevelButtonIsMultiValueMenuTag`, `kControlLittleArrowsIncrementValueTag`, `kControlGroupBoxFrameRectTag`, and `kControlWindowHeaderListHeaderTag`.
	Added documentation for numerous older functions as well, and fixed miscellaneous errors.
2005-07-07	Made bug fixes and did some consolidation. Filled out result code table.
2003-10-15	Removed Data Browser APIs. These APIs are now in the *Data Browser Reference*.
	Parameter type for `outNumChildren` in `CountSubControls` (page 31) corrected to be `UInt16`.
	Moved Miscellaneous functions into other categories. Rearranged function categories and changed the location of some functions.
	Removed functions that are unsupported in Carbon. Also removed the `CtlCTab` structure and the `PreFilterEventProc` callback type.
2003-02-01	Updated formatting and linking.
	Changed mention of the `GetControlTitleAsCFString` to the correct API name: `CopyControlTitleAsCFString`.
	Updated text for `EmbedControl` (page 73), indicating that it is legal in Mac OS X for the controlto be embedded and its desired host to be in different windows; `EmbedControl` will move the control from one window to the other.

Date	Notes
	Added caveat to the `kControlClockLongDateTag` constant in "Clock Control Data Tag Constants" (page 216) indicating that some fields in the date structure may not be valid for all variants.

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

W