Window Manager Reference

Carbon > User Experience

2007-10-31

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

Simultaneously published in the United States and Canada.

Contents

Window Manager Reference

Framework:	Carbon/Carbon.h
Declared in	HIToolboxDebugging.h HIWindowViews.h IOMacOSTypes.h MacWindows.h QuickdrawTypes.h

Overview

Your application uses the Window Manager to create and manage windows. For example, your application uses the Window Manager to create and display a new window when the user creates a new document or opens an existing document. When the user clicks or holds down the mouse button while the cursor is in a window created by your application, you use the Window Manager to determine the location of the mouse action and to alter the window display as appropriate. When the user closes a window, you use the Window Manager to remove the window from the screen.

A Macintosh application uses windows for most communication with the user, from discrete interactions such as presenting and acknowledging alert boxes to open-ended interactions such as creating and editing documents. Users generally type words and formulas, draw pictures, or otherwise enter data in a window on the screen. Your application typically lets the user save this data in a file, open saved files, and view the saved data in a window.

A window can be any size or shape, and the user can display any number of windows, within the limits of available memory, on the screen at once.

The Window Manager defines a set of standard windows and provides a set of routines for managing them. The Window Manager helps your application display windows that are consistent with the Macintosh user interface.

Note: Historically, the Window Manager has offered different successive methods for creating and manipulating windows. Many of the older functions have been deprecated and, in most cases, this reference provides a recommended replacement. For the most up-to-date information about creating windows, see the document *Handling Carbon Windows and Controls*.

Carbon supports the Window Manager. Be aware, however, that if you use custom window definition procedures (also known as WDEFs), you must move them out of resources and compile them into your application. In addition:

- Your application must use the functions defined by the Window Manager whenever it creates and disposes of Window Manager data structures. For example, instead of directly creating and disposing of window records, applications must call Window Manager functions such as `CreateNewWindow` and `DisposeWindow`.
- You must revise your application so that it accesses Window Manager data structures only through accessor functions.
- You are encouraged to adopt the standard Mac OS window definition procedures in your application. Applications that use the standard Mac OS window definition procedures inherit the Mac OS human interface appearance on Mac OS 9 and Mac OS X. Applications that use custom window definition procedures work correctly, but because custom definition procedures invoke their own drawing routines, the Mac OS can't draw these applications with the current appearance (unless you specifically use Appearance Manager drawing primitives).

Functions by Task

Accessing Information About a Window

`IsValidWindowPtr` (page 111)
Reports whether a pointer is a valid window pointer.

`GetWindowGreatestAreaDevice` (page 65)
Returns the graphics device with the greatest area of intersection with a specified window region.

`HIWindowGetGreatestAreaDisplay` (page 98)
Finds the display with the greatest area of intersection with a window region.

`HIWindowCopyShape` (page 91)
Retrieves a shape that describes a region of a window.

`HIWindowGetScaleMode` (page 100)
Obtains the window's scale mode and the application's display scale factor.

`GetWindowList` (page 73)
Obtains the first window in a window list.

`GetWindowWidgetHilite` (page 82)
Obtains the window part code of the window widget that is currently highlighted.

`IsWindowModified` (page 116)
Obtains the modification state of the specified window.

`SetWindowModified` (page 146)
Sets the modification state of the specified window.

`HIWindowGetCGWindowID` (page 98)
Returns the Quartz window ID assigned to a window.

`HIWindowFromCGWindowID` (page 95)
Returns the window in the current process with a specified Quartz window ID.

`GetWindowRegion` (page 265) Deprecated in Mac OS X v10.5
Obtains a handle to a specific window region.

Activating Window Path Pop-Up Menus

`IsWindowPathSelectEvent` (page 117)
Determines whether a Carbon event describing a click on a window's title should cause a path selection menu to be displayed.

`WindowPathSelect` (page 166)
Displays a window path pop-up menu.

`IsWindowPathSelectClick` (page 271) Deprecated in Mac OS X v10.5
Reports whether a mouse click should activate the window path pop-up menu. (Deprecated. Use `IsWindowPathSelectEvent` (page 117) instead.)

Associating Data With Windows

`GetWindowProperty` (page 75)
Obtains a piece of data that is associated with a window.

`SetWindowProperty` (page 147)
Associates an arbitrary piece of data with a window.

`GetWindowPropertySize` (page 77)
Obtains the size of a piece of data that is associated with a window.

`RemoveWindowProperty` (page 123)
Removes a piece of data that is associated with a window.

`ChangeWindowPropertyAttributes` (page 35)
Changes attributes associated with a window property.

`GetWindowPropertyAttributes` (page 76)
Obtains the attributes of a window property.

Collapsing Windows

`CollapseWindow` (page 37)
Collapses or expands a window to the dock.

`CollapseAllWindows` (page 37)
Collapses or expands all collapsable windows in an application.

`IsWindowCollapsed` (page 113)
Determines whether a window is currently collapsed.

`IsWindowCollapsable` (page 112)
Determines whether a window can be collapsed.

Creating, Storing, and Closing Windows

`CreateNewWindow` (page 41)
Creates a window from parameter data.

`CreateCustomWindow` (page 40)
Creates a custom window based on a registered toolbox object class or a custom window root view.

`HIWindowCreate` (page 92)
Creates a standard or custom window.

`DisposeWindow` (page 45)
Removes a window.

`CreateWindowFromCollection` (page 255) Deprecated in Mac OS X v10.5
Creates a window from collection data. (Deprecated. Use `HIArchiveCopyDecodedCFType` to decode a window from an archive instead.)

`CreateWindowFromResource` (page 255) Deprecated in Mac OS X v10.5
Creates a window from `'wind'` resource data. (Deprecated. Use nib files and `CreateWindowFromNib` instead.)

`StoreWindowIntoCollection` (page 283) Deprecated in Mac OS X v10.5
Stores data describing a window into a collection. (Deprecated. Use `HIArchiveEncodeCFType` to encode a window to an archive instead.)

Displaying Floating Windows and Window Animations

`AreFloatingWindowsVisible` (page 31)
Indicates whether an application's floating windows are currently visible.

`HideFloatingWindows` (page 83)
Hides an application's floating windows.

`ShowFloatingWindows` (page 153)
Shows an application's floating windows.

Displaying Windows

`ActivateWindow` (page 30)
Activates or deactivates a window.

`IsWindowActive` (page 112)
Indicates whether the specified window is active.

`HiliteWindow` (page 84)
Sets a window's highlighting status.

`SelectWindow` (page 129)
Makes a window active.

`ShowWindow` (page 156)
Makes an invisible window visible.

`HideWindow` (page 84)
Makes a window invisible.

`ShowHide` (page 154)
Sets a window's visibility.

`BringToFront` (page 33)
Brings a window to the front.

`SendBehind` (page 129)
Moves one window behind another.

`HIWindowInvalidateShadow` (page 102)
Recalculates a window's shadow.

Dragging Proxy Icons

`BeginWindowProxyDrag` (page 32)
Creates the drag reference and the drag image when the user drags a proxy icon.

`EndWindowProxyDrag` (page 47)
Disposes of the drag reference when the user completes the drag of a proxy icon.

`HiliteWindowFrameForDrag` (page 85)
Sets the highlight state of the window's structure region to reflect the window's validity as a drag-and-drop destination.

`TrackWindowProxyDrag` (page 159)
Handles all aspects of the drag process when the user drags a proxy icon.

`TrackWindowProxyFromExistingDrag` (page 160)
Allows custom handling of the drag process when the user drags a proxy icon.

`HIWindowTrackProxyDrag` (page 108)
Tracks the drag of a window proxy icon.

Establishing Proxy Icons

`GetWindowProxyAlias` (page 78)
Obtains an alias for the file that is associated with a window.

`SetWindowProxyAlias` (page 148)
Associates a file with a window.

`GetWindowProxyIcon` (page 78)
Obtains a window's proxy icon.

`SetWindowProxyIcon` (page 149)
Overrides the default proxy icon for a window.

`RemoveWindowProxy` (page 123)
Dissociates a file from a window.

`SetWindowProxyCreatorAndType` (page 148)
Sets the proxy icon for a window that lacks an associated file.

`HIWindowGetProxyFSRef` (page 100)
Obtains the `FSRef` used to determine the proxy icon for a window.

`HIWindowSetProxyFSRef` (page 106)
Sets the proxy icon for a window using an `FSRef` to a file system object.

Getting and Setting Window Structure Fields

`GetNextWindow` (page 56)
Returns the next window in a window list.

`GetWindowKind` (page 73)
Returns a window's window kind.

`SetWindowKind` (page 145)
Sets a window's window kind.

`GetWindowPort` (page 74)
Gets the window's color graphics port.

`SetPortWindowPort` (page 132)
Sets the current graphics port to the window's port.

`GetWindowPortBounds` (page 75)
Obtains the bounds of the window port.

`GetWindowStandardState` (page 79)
Obtains a window's standard zoom rectangle.

`SetWindowStandardState` (page 151)
Sets a window's standard zoom rectangle.

`GetWindowUserState` (page 81)
Returns a window's user zoom rectangle.

`SetWindowUserState` (page 152)
Sets a window's user zoom rectangle.

`IsWindowHilited` (page 114)
Indicates whether the window frame is currently highlighted.

`IsWindowLatentVisible` (page 115)
Indicates whether a window is visible onscreen or is latently visible but not currently onscreen.

`IsWindowVisible` (page 118)
Indicates whether the window frame is currently visible.

`GetWindowStructurePort` (page 80)
Obtains a graphics port that is used when drawing a window's structure.

`GetWindowStructureWidths` (page 80)
Obtains the width of the structure region on each edge of a window.

Handling Mouse Events in Windows

`DragWindow` (page 46)
Moves a window on the screen when the user drags it by its drag region.

`MoveWindow` (page 119)
Moves a window on the desktop.

`PinRect` (page 121)
Returns the point within the specified rectangle that is closest to the specified point.

`SizeWindow` (page 156)
Sets the size of a window.

`TrackBox` (page 158)
Tracks clicks in the collapse, close, size, and zoom boxes, and clicks of the toolbar button.

`TrackGoAway` (page 159)
Tracks the cursor when the user presses the mouse button while the cursor is in the close box.

`ZoomWindow` (page 167)
Zooms the window when the user has pressed and released the mouse button with the cursor in the zoom box.

Locating Windows

`ActiveNonFloatingWindow` (page 31)
Returns the currently active nonfloating window.

`FrontNonFloatingWindow` (page 50)
Returns to the application the frontmost visible window that is not a floating window.

`FindWindow` (page 48)
Maps the location of the cursor to a part of the screen or a region of a window when your application receives a mouse-down event.

`FindWindowOfClass` (page 49)
Finds a window of a specific class at the specified point onscreen.

`HIWindowFindAtLocation` (page 94)
Finds a window in the current process at a specified location.

`GetFrontWindowOfClass` (page 55)
Obtains the frontmost window of a given class.

`GetNextWindowOfClass` (page 57)
Obtains the next window in a given window group.

`GetPreviousWindow` (page 57)
Returns the window above the specified window in the window list.

`FrontWindow` (page 260) Deprecated in Mac OS X v10.5
Identifies the frontmost visible window. (Deprecated. Use `ActiveNonFloatingWindow` (page 31), `FrontNonFloatingWindow` (page 50), or `GetFrontWindowOfClass` (page 55) instead.)

Maintaining the Update Region

`BeginUpdate` (page 31)
Starts updating a window when you receive an update event for that window.

`EndUpdate` (page 47)
Finishes updating a window.

`InvalWindowRect` (page 109)
Adds a rectangle to a window's update region.

`InvalWindowRgn` (page 110)
Adds a region to a window's update region.

`IsWindowUpdatePending` (page 118)
Determines whether a window update is pending.

`ValidWindowRect` (page 165)
Removes a rectangle from a window's update region.

`ValidWindowRgn` (page 166)
Removes a region from a window's update region.

Managing Activation Scope

`GetWindowActivationScope` (page 59)
Obtains a window's activation scope.

`SetWindowActivationScope` (page 135)
Sets a window's activation scope.

Managing Dock Tiles

`HIWindowCreateCollapsedDockTileContext` (page 93)
Creates a Quartz graphics context for drawing a collapsed window's Dock tile.

`HIWindowReleaseCollapsedDockTileContext` (page 104)
Releases a Quartz graphics context for drawing a collapsed window's Dock tile.

`GetWindowDockTileMenu` (page 64)
Returns the menu to be displayed by a window's dock tile.

`SetWindowDockTileMenu` (page 140)
Associates a pop-up menu with a window.

`UpdateCollapsedWindowDockTile` (page 164)
Updates the image of a window in the dock to the current contents of the window.

`CreateQDContextForCollapsedWindowDockTile` (page 254) Deprecated in Mac OS X v10.5
Obtains a `CGrafPtr` for a collapsed window's tile in the dock. (Deprecated. Use `HIWindowCreateCollapsedDockTileContext` (page 93) instead.)

`ReleaseQDContextForCollapsedWindowDockTile` (page 279) Deprecated in Mac OS X v10.5
Releases a port and other state created by `CreateQDContextForCollapsedWindowDockTile`. (Deprecated. Use `HIWindowReleaseCollapsedDockTileContext` (page 104) instead.)

Managing Modality

`GetWindowModality` (page 74)
Obtains the modality of a window.

`SetWindowModality` (page 145)
Sets the modality of a window.

`HIWindowIsDocumentModalTarget` (page 103)
Determines if a window is currently the target window of another document modal window, such as a sheet.

Managing Themes

`SetThemeWindowBackground` (page 133)
Sets a window's background theme.

`HIWindowGetThemeBackground` (page 101)
Gets the theme background brush for a window.

`SetThemeTextColorForWindow` (page 133)
Sets a text color that contrasts with a theme brush.

Managing Toolbars

`GetWindowToolbar` (page 81)
Obtains the toolbar associated with a window.

`SetWindowToolbar` (page 152)
Associates a toolbar with a window.

`ShowHideWindowToolbar` (page 154)
Shows or hides the toolbar.

`IsWindowToolbarVisible` (page 117)
Determines whether a window's toolbar is visible.

`HIWindowSetToolbarView` (page 107)
Sets a custom toolbar view for a window.

Managing Transitions

`TransitionWindow` (page 162)
Shows, hides, moves, or resizes a window with appropriate animation and sound.

`TransitionWindowAndParent` (page 163)
Shows or hides a window, potentially also moving a second window, with animation and sound.

`TransitionWindowWithOptions` (page 163)
Transitions a window from one state to another with appropriate animation and sound.

Managing Transparency

`GetWindowAlpha` (page 59)
Returns the current alpha channel value for the window.

`SetWindowAlpha` (page 135)
Sets the window's alpha channel value.

Managing UPPs

`DisposeWindowDefUPP` (page 256) Deprecated in Mac OS X v10.5
Disposes of the UPP for your window definition. (Deprecated. The WDEF interface is deprecated; use a custom HIView to draw your custom window frame instead.)

`DisposeWindowPaintUPP` (page 257) Deprecated in Mac OS X v10.5
Disposes of the UPP to your region painting callback function. (Deprecated. The window content painting interface is deprecated; use a `kEventControlDraw` Carbon event handler on a compositing window's content view instead.)

`InvokeWindowDefUPP` (page 270) Deprecated in Mac OS X v10.5
Invokes the UPP for a window definition. (Deprecated. The WDEF interface is deprecated; use a custom HIView to draw your custom window frame instead.)

`InvokeWindowPaintUPP` (page 270) Deprecated in Mac OS X v10.5
Invokes the UPP for the specified painting region. (Deprecated. The window content painting interface is deprecated; use a `kEventControlDraw` Carbon event handler on a compositing window's content view instead.)

`NewWindowDefUPP` (page 277) Deprecated in Mac OS X v10.5
Creates a new UPP for a window definition. (Deprecated. The WDEF interface is deprecated; use a custom HIView to draw your custom window frame instead.)

`NewWindowPaintUPP` (page 277) Deprecated in Mac OS X v10.5
Creates a new UPP for a painting region. (Deprecated. The window content painting interface is deprecated; use a `kEventControlDraw` Carbon event handler on a compositing window's content view instead.)

Managing Window Attributes

`GetWindowAttributes` (page 59)
Obtains the attributes of a window.

`GetAvailableWindowAttributes` (page 51)
Returns the window attributes that are valid for a window class

`ChangeWindowAttributes` (page 34)
Changes a window's attributes.

`HIWindowTestAttribute` (page 108)
Returns a Boolean value indicating whether a window has a specified attribute.

`HIWindowChangeAttributes` (page 86)
Changes the attributes of a window.

`HIWindowIsAttributeAvailable` (page 102)
Returns a Boolean value indicating whether a window attribute is valid for a specified window class.

Managing Window Availability

`HIWindowChangeAvailability` (page 87)
Changes the availability of a window during Exposé or in Spaces.

`HIWindowGetAvailability` (page 96)
Obtains the availability of a window during Exposé or in Spaces.

Managing Window Classes

`GetWindowClass` (page 61)
Obtains the class of a window.

`HIWindowChangeClass` (page 87)
Changes the appearance and behavior of a window.

`IsValidWindowClass` (page 111)
Determines whether a given window class is valid.

`SetWindowClass` (page 281) Deprecated in Mac OS X v10.5
Sets the class of a window. (Deprecated. Use `HIWindowChangeClass` (page 87), `SetWindowGroup` (page 140), or `HIWindowChangeAttributes` (page 86) instead.)

Managing Window Features

`GetWindowFeatures` (page 64)
Obtains the features that a window supports.

`HIWindowChangeFeatures` (page 88)
Changes a window's features.

Managing Window Focus

`SetUserFocusWindow` (page 134)
Designates a window to receive user focus.

`GetUserFocusWindow` (page 58)
Returns the current user focus window.

`HIWindowShowsFocus` (page 107)
Returns a Boolean value indicating whether a window's content should show focus indicators such as focus rings.

Managing Window Groups

`ChangeWindowGroupAttributes` (page 35)
Changes the attributes of a window group.

`CopyWindowGroupName` (page 39)
Obtains a copy of the window group name.

`CountWindowGroupContents` (page 40)
Counts the number of members of a window group.

`CreateWindowGroup` (page 43)
Creates a window group.

`DebugPrintAllWindowGroups` (page 43)
Debugging utility function listing all window groups.

`DebugPrintWindowGroup` (page 44)
Debugging utility functions for use with window groups.

`GetIndexedWindow` (page 56)
Obtains the window at the given index in the window group.

`GetWindowGroup` (page 66)
Obtains the window group associated with a window.

`GetWindowGroupAttributes` (page 66)
Obtains the attributes of a window group.

`GetWindowGroupContents` (page 67)
Obtains the contents of a window group.

`GetWindowGroupLevel` (page 68)
Obtains the level of the group in the window class hierarchy.

`GetWindowGroupLevelOfType` (page 68)
Obtains the Core Graphics window level of a window group.

`GetWindowGroupOfClass` (page 69)
Obtains the window group corresponding to a given window class.

`GetWindowGroupOwner` (page 70)
Obtains the window that owns a window group. (if any)

`GetWindowGroupParent` (page 70)
Obtains the parent group of a window group.

`GetWindowGroupRetainCount` (page 70)
Determines the current reference count for a window group.

`GetWindowGroupSibling` (page 71)
Obtains the next or previous group of a window group.

`GetWindowIndex` (page 72)
Obtains the index number of a specified window in a group.

`IsWindowContainedInGroup` (page 114)
Determines if a window is a member of a window group or any of its subgroups.

`ReleaseWindowGroup` (page 122)
Decrements the reference count for a window group.

`RetainWindowGroup` (page 127)
Increments the reference count for a window group.

`SendWindowGroupBehind` (page 130)
Orders one window group behind another.

`SetWindowGroup` (page 140)
Assigns a window to a window group.

`SetWindowGroupLevel` (page 141)
Sets the level of group in the window class hierarchy.

`SetWindowGroupLevelOfType` (page 142)
Sets the window level of a window group.

`SetWindowGroupName` (page 143)
Assigns a name to a window group.

`SetWindowGroupOwner` (page 143)
Sets a window as the owner of a window group.

`SetWindowGroupParent` (page 144)
Sets a window group to be the parent of another window group.

Managing Window Titles

`CopyWindowAlternateTitle` (page 38)
Obtains a copy of the alternate window title.

`SetWindowAlternateTitle` (page 136)
Sets an alternate window title.

`CopyWindowTitleAsCFString` (page 39)
Copies the window title into a Core Foundation string.

`SetWindowTitleWithCFString` (page 151)
Sets the window title to the contents of a Core Foundation string.

Manipulating Drawers

`HIWindowCopyDrawers` (page 91)
Obtains an array of the drawers that are attached to a window.

`OpenDrawer` (page 120)
Opens a drawer.

`CloseDrawer` (page 36)
Closes a drawer.

`GetDrawerCurrentEdge` (page 53)
Obtains the current window edge from which the drawer appears.

`GetDrawerPreferredEdge` (page 54)
Obtains the preferred opening edge for a drawer.

`SetDrawerPreferredEdge` (page 132)
Set the preferred window edge from which the drawer should appear.

`GetDrawerOffsets` (page 53)
Obtains the positioning offsets of a drawer.

`SetDrawerOffsets` (page 131)
Sets the positioning offsets for the drawer with respect to its parent window.

`GetDrawerParent` (page 54)
Obtains the parent window of a drawer.

`SetDrawerParent` (page 131)
Sets the parent window for a drawer.

`GetDrawerState` (page 55)
Determines the current state of the drawer.

`ToggleDrawer` (page 157)
Toggles the drawer state.

Manipulating Sheets

`GetSheetWindowParent` (page 58)
Obtains the parent window of a sheet.

`ShowSheetWindow` (page 155)
Shows a sheet window using appropriate visual effects.

`HideSheetWindow` (page 84)
Hides a sheet window using appropriate visual effects.

`DetachSheetWindow` (page 44)
Detaches a sheet window from its parent window.

Manipulating Window Color Information

`GetWindowContentColor` (page 62)
Obtains the color to which a window's content region is redrawn.

`GetWindowContentPattern` (page 63)
Obtains the pattern to which a window's content region is redrawn.

`SetWindowContentColor` (page 138)
Sets the color to which a window's content region is redrawn.

`GetWRefCon` (page 82)
Returns the reference constant from a window.

`SetWindowContentPattern` (page 138)
Sets the pattern to which a window's content region is redrawn.

`SetWRefCon` (page 153)
Sets the `refCon` field of a window.

`GetWindowPic` (page 264) Deprecated in Mac OS X v10.5
Returns a handle to a window's picture. (Deprecated. Use an HIImageView object to draw a window's content and ask the view for its image instead.)

`SetWindowPic` (page 281) Deprecated in Mac OS X v10.5
Sets a picture for the Window Manager to draw in a window's content region. (Deprecated. Use an HIImageView object to draw a window's content instead.)

Referencing Windows

`CloneWindow` (page 253) Deprecated in Mac OS X v10.5
Increments the number of references to a window. (Deprecated. Use `CFRetain` instead.)

`GetWindowOwnerCount` (page 263) Deprecated in Mac OS X v10.5
Obtains the number of existing references to a window. (Deprecated. Use `CFGetRetainCount` instead.)

`GetWindowRetainCount` (page 266) Deprecated in Mac OS X v10.5
Returns the retain count of a window. (Deprecated. Use `CFGetRetainCount` instead.)

`ReleaseWindow` (page 279) Deprecated in Mac OS X v10.5
Decrements the retain count of a window, and destroys the window if the retain count falls to zero. (Deprecated. Use `CFRelease` instead.)

`RetainWindow` (page 280) Deprecated in Mac OS X v10.5
Increments the retain count of a window. (Deprecated. Use `CFRetain` instead.)

Scrolling

`ScrollWindowRect` (page 127)
Scroll any area of a window.

`ScrollWindowRegion` (page 128)
Scrolls a window's region.

Sizing and Positioning Windows

`GetWindowBounds` (page 60)
Obtains the size and position of the bounding rectangle of the specified window region.

`HIWindowGetBounds` (page 97)
Gets the bounds of a specified region of a window.

`SetWindowBounds` (page 136)
Sets a window's size and position from the bounding rectangle of the specified window region.

`HIWindowSetBounds` (page 105)
Sets the bounds of a window based on either the structure or content region.

`MoveWindowStructure` (page 119)
Positions a window relative to its structure region.

`RepositionWindow` (page 124)
Positions a window relative to another window or a display screen.

`ResizeWindow` (page 125)
Handles all user interaction while a window is being resized.

`GetAvailableWindowPositioningBounds` (page 51)
Obtains the available window positioning bounds.

`HIWindowGetAvailablePositioningBounds` (page 96)
Gets the available window positioning bounds on a display.

`GetAvailableWindowPositioningRegion` (page 52)
Obtains the available window positioning region.

`HIWindowCopyAvailablePositioningShape` (page 90)
Copies the available window positioning shape on a display.

`ConstrainWindowToScreen` (page 38)
Moves and resizes a window so that it's contained entirely on a single screen.

`HIWindowConstrain` (page 89)
Moves and resizes a window to be within a specified bounding rectangle.

`GetWindowResizeLimits` (page 79)
Returns the minimum and maximum content sizes for a window.

`SetWindowResizeLimits` (page 150)
Sets the maximum and minimum resize limits for windows.

Updating the Screen

`EnableScreenUpdates` (page 46)
Enables screen updates for changes to the current application's windows.

`DisableScreenUpdates` (page 45)
Disables updates for changes to the current application's windows.

Using Default and Cancel Buttons

You can use these functions to add dialog-like button controls to normal windows.

`SetWindowDefaultButton` (page 139)
: Specifies a default button for a window.

`GetWindowDefaultButton` (page 63)
: Returns the current default button for a window.

`SetWindowCancelButton` (page 137)
: Specifies a Cancel button for a window.

`GetWindowCancelButton` (page 61)
: Returns the current Cancel button for a window.

Zooming Windows

`GetWindowIdealUserState` (page 71)
: Obtains the size and position of a window in its user state.

`HIWindowGetIdealUserState` (page 99)
: Gets the bounds of a window's content region in its user state.

`IsWindowInStandardState` (page 115)
: Determines whether a window is currently zoomed in to the user state or zoomed out to the standard state.

`HIWindowIsInStandardState` (page 103)
: Returns a Boolean value indicating whether a window is zoomed out to its standard state.

`SetWindowIdealUserState` (page 144)
: Sets the size and position of a window in its user state.

`HIWindowSetIdealUserState` (page 105)
: Sets the bounds of a window's content region in its user state.

`ZoomWindowIdeal` (page 168)
: Zooms a window in accordance with human interface guidelines.

Miscellaneous

`CreateStandardWindowMenu` (page 42)
: Creates a standard window menu for your application.

`GetWindowFromPort` (page 65)
: Gets a window reference from a `CGrafPtr` data type.

`HIWindowFlush` (page 95)
: Flushes any dirty areas a window might have.

`RegisterWindowDefinition` (page 122)
: Registers a binding between a resource ID and a window definition function.

`ReshapeCustomWindow` (page 125)
: Notifies the Window Manager that a custom window's shape has changed.

`InstallWindowContentPaintProc` (page 269) Deprecated in Mac OS X v10.5
Installs a window content painting callback. (Deprecated. Use a `kEventControlDraw` Carbon event handler on a window's content view instead.)

Legacy Functions

`CalcVis` (page 251) Deprecated in Mac OS X v10.5
Calculates the visible region of a window. (Deprecated. There is no replacement function.)

`CalcVisBehind` (page 251) Deprecated in Mac OS X v10.5
Calculates the visible regions of a series of windows. (Deprecated. There is no replacement function.)

`CheckUpdate` (page 252) Deprecated in Mac OS X v10.5
Scans the window list for windows that need updating. (Deprecated. Use `FindSpecificEventInQueue` or `AcquireFirstMatchingEventInQueue` instead.)

`ClipAbove` (page 253) Deprecated in Mac OS X v10.5
Determines the clip region of the Window Manager port. (Deprecated. There is no replacement function.)

`DragGrayRgn` (page 257) Deprecated in Mac OS X v10.5
Moves a gray outline of a region on the screen, following the movements of the cursor, until the mouse button is released. (Deprecated. Use an overlay window or other custom drawing instead.)

`DragTheRgn` (page 259) Deprecated in Mac OS X v10.5
Tracks the mouse as the user drags the outline of a region. (Deprecated. Use an overlay window or other custom drawing instead.)

`DrawGrowIcon` (page 259) Deprecated in Mac OS X v10.5
Draws a grow icon in the window frame. (Deprecated. There is no replacement function.)

`GetGrayRgn` (page 260) Deprecated in Mac OS X v10.5
Returns a region that covers the desktop area of all active displays. (Deprecated. To determine the area in which a window may be positioned, use `HIWindowGetAvailablePositioningBounds` (page 96) or `HIWindowCopyAvailablePositioningShape` (page 90).)

`GetNewCWindow` (page 261) Deprecated in Mac OS X v10.5
Creates a color window from a window resource. (Deprecated. Use nib files and `CreateWindowFromNib` instead.)

`GetNewWindow` (page 262) Deprecated in Mac OS X v10.5
Creates a window from a window resource. (Deprecated. Use nib files and `CreateWindowFromNib` instead.)

`GetWindowProxyFSSpec` (page 265) Deprecated in Mac OS X v10.5
Obtains a file system specification structure for the file that is associated with a window. (Deprecated. Use `HIWindowGetProxyFSRef` (page 100) instead.)

`GetWTitle` (page 267) Deprecated in Mac OS X v10.5
Retrieves the title of a window as a Pascal string. (Deprecated. Use `CopyWindowTitleAsCFString` (page 39) instead.)

`GetWVariant` (page 267) Deprecated in Mac OS X v10.5
Returns a window's variation code. (Deprecated. Use `GetWindowAttributes` (page 59) to determine aspects of a window's appearance or behavior.)

`GrowWindow` (page 268) Deprecated in Mac OS X v10.5
Allows the user to change the size of a window. (Deprecated. Use `ResizeWindow` (page 125) instead.)

`NewCWindow` (page 272) Deprecated in Mac OS X v10.5
: Creates a window with a specified list of characteristics. (Deprecated. Use `CreateNewWindow` (page 41) instead.)

`NewWindow` (page 274) Deprecated in Mac OS X v10.5
: Creates a window from a parameter list. (Deprecated. Use `CreateNewWindow` (page 41) instead.)

`PaintBehind` (page 277) Deprecated in Mac OS X v10.5
: Redraws a series of windows in the window list. (Deprecated. Use `InvalWindowRect` (page 109), `InvalWindowRgn` (page 110), or `HIViewSetNeedsDisplay` to invalidate a portion of a window.)

`PaintOne` (page 278) Deprecated in Mac OS X v10.5
: Redraws the invalid, exposed portions of one window on the desktop. (Deprecated. Use `InvalWindowRect` (page 109), `InvalWindowRgn` (page 110), or `HIViewSetNeedsDisplay` to invalidate a portion of a window.)

`SetWindowProxyFSSpec` (page 282) Deprecated in Mac OS X v10.5
: Associates a file with a window. (Deprecated. Use `HIWindowSetProxyFSRef` (page 106) instead.)

`SetWTitle` (page 283) Deprecated in Mac OS X v10.5
: Specifies a window's title. (Deprecated. Use `SetWindowTitleWithCFString` (page 151) instead.)

Functions

ActivateWindow

Activates or deactivates a window.

```
OSStatus ActivateWindow (
   WindowRef inWindow,
   Boolean inActivate
);
```

Parameters

inWindow
: The window to activate or deactivate.

inActivate
: Pass `true` to activate the window, `false` otherwise.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

BSDLLCTest

Declared In

`MacWindows.h`

ActiveNonFloatingWindow

Returns the currently active nonfloating window.

```
WindowRef ActiveNonFloatingWindow (
   void
);
```

Return Value
A reference to the active window.

Discussion
Note that the active window is not necessarily the frontmost window, and it is not necessarily the window with user focus. Call `GetUserFocusWindow` (page 58) to get the window that has user focus.

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

Declared In
`MacWindows.h`

AreFloatingWindowsVisible

Indicates whether an application's floating windows are currently visible.

```
Boolean AreFloatingWindowsVisible (
   void
);
```

Return Value
A Boolean whose value is `true` if the application's floating windows are currently shown or `false` if the application's floating windows are currently hidden.

Discussion
This function checks the visibility state of an application's floating windows, which are hidden automatically when the application receives a suspend event and are made visible automatically when the application receives a resume event.

Special Considerations

The `AreFloatingWindowsVisible` function operates only upon windows created with the `kFloatingWindowClass` constant; see "Window Class Constants" (page 184) for more details on this constant.

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

Declared In
`MacWindows.h`

BeginUpdate

Starts updating a window when you receive an update event for that window.

```
void BeginUpdate (
   WindowRef window
);
```

Parameters

window

The window that is to be updated when an update event is received. Your application gets this information from the `message` field in the update event structure.

Discussion

The `BeginUpdate` function limits the visible region of the window's graphics port to the intersection of the visible region and the update region it then sets the window's update region to an empty region. After calling `BeginUpdate`, your application redraws either the entire content region or only the visible region. In either case, only the parts of the window that require updating are actually redrawn on the screen.

Every call to `BeginUpdate` must be matched with a subsequent call to `EndUpdate` (page 47) after your application redraws the content region. `BeginUpdate` and `EndUpdate` can't be nested. That is, you must call `EndUpdate` before the next call to `BeginUpdate`.

In Mac OS X, you only receive one update event. If you don't call `BeginUpdate`, you won't receive any further update events until the window is invalidated again.

Special Considerations

This function should not be used on composited windows. Modifying a composited window's update region does not affect the area of the window to be drawn. A composited window does not use its window update region to control drawing. Instead, a composited window determines what to draw by looking at the invalid regions of the views contained in the window.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

HideMenuBar

Simple DrawSprocket

Declared In

`MacWindows.h`

BeginWindowProxyDrag

Creates the drag reference and the drag image when the user drags a proxy icon.

```
OSStatus BeginWindowProxyDrag (
   WindowRef window,
   DragRef *outNewDrag,
   RgnHandle outDragOutlineRgn
);
```

Parameters

window

The window whose proxy icon is being dragged.

outNewDrag

On input, a pointer to a value of type `DragRef`. On return, the value refers to the current drag process.

outDragOutlineRgn

On input, a value of type `RgnHandle`. Your application can create this handle with a call to the QuickDraw function `NewRgn`. On return, this region is set to the outline of the icon being dragged.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Discussion
Typically, if the proxy icon represents a type of object (currently, file system entities such as files, folders, and volumes) supported by the Window Manager, the Window Manager can handle all aspects of the drag process itself, and your application should call the function `TrackWindowProxyDrag` (page 159). However, if the proxy icon represents a type of data that the Window Manager does not support, or if you want to implement custom dragging behavior, your application should call the function `TrackWindowProxyFromExistingDrag` (page 160).

The `TrackWindowProxyFromExistingDrag` (page 160) function accepts an existing drag reference and adds file data if the window contains a file proxy. If your application uses `TrackWindowProxyFromExistingDrag`, you then have the choice of using this function in conjunction with the functions `BeginWindowProxyDrag` and `EndWindowProxyDrag` (page 47) or simply calling `TrackWindowProxyFromExistingDrag` and handling all aspects of creating and disposing of the drag yourself.

Specifically, your application can call `BeginWindowProxyDrag` to set up the drag image and drag reference. Your application must then track the drag, using `TrackWindowProxyFromExistingDrag`, and do any required moving of data and, finally, call `EndWindowProxyDrag` (page 47) to dispose of the drag reference. `BeginWindowProxyDrag` should not be used for types handled by the Window Manager unless the application wants to implement custom dragging behavior for those types.

Your application detects a drag when the function `FindWindow` (page 48) returns the `inProxyIcon` result code.

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

Not available to 64-bit applications.

Declared In
`MacWindows.h`

BringToFront

Brings a window to the front.

```
void BringToFront (
   WindowRef window
);
```

Parameters

window

The window that is to be brought to the front.

Discussion
The `BringToFront` function puts the specified window at the beginning of the window list and redraws the window in front of all others on the screen. It does not change the window's highlighting or make it active.

Your application does not ordinarily call `BringToFront`. The user interface guidelines specify that the frontmost window should be the active window. To bring a window to the front and make it active, call the function `SelectWindow` (page 129).

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

Related Sample Code
BSDLLCTest

Declared In
`MacWindows.h`

ChangeWindowAttributes

Changes a window's attributes.

```
OSStatus ChangeWindowAttributes (
   WindowRef window,
   WindowAttributes setTheseAttributes,
   WindowAttributes clearTheseAttributes
);
```

Parameters

window
The window whose attributes you want to change.

setTheseAttributes
The attributes you want to set. Pass `kWindowNoAttributes` if you do not want to set any attributes. See "Window Attributes" (page 194) for a list of window attributes.

clearTheseAttributes
The attributes you want to clear (if any). Pass `kWindowNoAttributes` if you do not want to clear any attributes. See "Window Attributes" (page 194) for a list of window attributes.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Discussion
If the changed attributes affect the visible window's frame, the window regions are recalculated and the window is redrawn.

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

Related Sample Code
QTCarbonShell

Declared In
MacWindows.h

ChangeWindowGroupAttributes

Changes the attributes of a window group.

```
OSStatus ChangeWindowGroupAttributes (
   WindowGroupRef inGroup,
   WindowGroupAttributes setTheseAttributes,
   WindowGroupAttributes clearTheseAttributes
);
```

Parameters

inGroup
The window group whose attributes you want to set.

setTheseAttributes
The attributes you want to set. See "Window Group Attributes" (page 227) for a list of possible attributes.

clearTheseAttributes
The attributes you want to clear (if any). See "Window Group Attributes" (page 227) for a list of possible attributes.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

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

Declared In
MacWindows.h

ChangeWindowPropertyAttributes

Changes attributes associated with a window property.

```
OSStatus ChangeWindowPropertyAttributes (
   WindowRef window,
   OSType propertyCreator,
   OSType propertyTag,
   OptionBits attributesToSet,
   OptionBits attributesToClear
);
```

Parameters

window
The window whose property attributes are to be changed.

propertyCreator
The property creator.

propertyTag
The property tag.

attributesToSet

The attributes to set. For a possible value, see "Window Property Persistent Constant" (page 220).

attributesToClear

The attributes to clear. For a possible value, see "Window Property Persistent Constant" (page 220).

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

CloseDrawer

Closes a drawer.

```
OSStatus CloseDrawer (
   WindowRef inDrawerWindow,
   Boolean inAsync
);
```

Parameters

inDrawerWindow

The drawer window that is to be closed.

inAsync

Pass `true` for asynchronous closing; otherwise, pass `false`.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Discussion

`CloseDrawer` may close the drawer either synchronously or asynchronously, depending on the value of the `inAsync` parameter. If `inAsync` is true, `CloseDrawer` installs an event loop timer that closes the drawer after `CloseDrawer` returns to the caller; therefore, the caller must be running its event loop for the drawer to close. If `inAsync` is false, `CloseDrawer` closes the drawer completely before returning to the caller. `CloseDrawer` retains the drawer window while the drawer is closing, and releases it when the drawer is fully closed. `CloseDrawer` sends the `kEventWindowDrawerClosing` event to the drawer, the drawer's parent, and the application before closing the drawer. If an event handler for this event returns `userCanceledErr`, `CloseDrawer` will return immediately without closing the drawer. `CloseDrawer` sends the `kEventWindowDrawerClosed` event to the drawer, the drawer's parent, and the application after the drawer has finished closing.

Availability

Available in Mac OS X v10.2 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

CollapseAllWindows

Collapses or expands all collapsable windows in an application.

```
OSStatus CollapseAllWindows (
   Boolean collapse
);
```

Parameters

collapse

Set to `true` to collapse all windows in the application. Set to `false` to expand all windows in the application.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Discussion

In Mac OS X, this function works with any window that has the `kWindowCollapseBoxAttribute`. If that attribute is not present, the Window Manager checks for the `kWindowCanCollapse` feature bit.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

CollapseWindow

Collapses or expands a window to the dock.

```
OSStatus CollapseWindow (
   WindowRef window,
   Boolean collapse
);
```

Parameters

window

The window that is to be collapsed or expanded.

collapse

Indicates whether the window should be collapsed or expanded.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Discussion

The `CollapseWindow` function tells the Window Manager to collapse or expand a window depending upon the value passed in the `collapse` parameter. In Mac OS X, any window that has the `kWindowCollapseBoxAttribute` can be collapsed. If that attribute is not present, the Window Manager checks for the `kWindowCanCollapse` feature bit.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
MacWindows.h

ConstrainWindowToScreen

Moves and resizes a window so that it's contained entirely on a single screen.

```
OSStatus ConstrainWindowToScreen (
    WindowRef inWindowRef,
    WindowRegionCode inRegionCode,
    WindowConstrainOptions inOptions,
    const Rect *inScreenRect,
    Rect *outStructure
);
```

Parameters

inWindowRef
The window to constrain.

inRegionCode
The window region to constrain. See "Window Region Constants" (page 217) for a list of possible constants to pass.

inOptions
Flags controlling how the window is constrained.

inScreenRect
A rectangle, in global coordinates, in which to constrain the window. May be NULL. If NULL, the window is constrained to the screen with the greatest intersection with the specified window region.

outStructure
On exit, contains the new structure bounds of the window, in global coordinates. May be NULL.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

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

See Also
HIWindowConstrain (page 89)

Declared In
MacWindows.h

CopyWindowAlternateTitle

Obtains a copy of the alternate window title.

```
OSStatus CopyWindowAlternateTitle (
   WindowRef inWindow,
   CFStringRef *outTitle
);
```

Parameters

inWindow

The window to get the alternate title from.

outTitle

Receives the alternate title for the window. If the window does not have an alternate title, `NULL` will be returned in `outTitle`.

Return Value
A result code. See "Window Manager Result Codes" (page 247). An operating system status code.

Discussion
See the discussion of `SetWindowAlternateTitle` (page 136) for more information about alternate window titles.

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

Declared In
`MacWindows.h`

CopyWindowGroupName

Obtains a copy of the window group name.

```
OSStatus CopyWindowGroupName (
   WindowGroupRef inGroup,
   CFStringRef *outName
);
```

Parameters

inGroup

The window group to query. For information on this data type,

outName

Return Value
A result code. See "Window Manager Result Codes" (page 247).

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

Declared In
`MacWindows.h`

CopyWindowTitleAsCFString

Copies the window title into a Core Foundation string.

```
OSStatus CopyWindowTitleAsCFString (
   WindowRef inWindow,
   CFStringRef *outString
);
```

Parameters

inWindow

The window whose title is to be copied.

outString

On output, the window's title.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

CarbonSketch

QTCarbonShell

Declared In

`MacWindows.h`

CountWindowGroupContents

Counts the number of members of a window group.

```
ItemCount CountWindowGroupContents (
   WindowGroupRef inGroup,
   WindowGroupContentOptions inOptions
);
```

Parameters

inGroup

The window group whose members are to be counted.

inOptions

Counting options. See "Window Group Content Options" (page 228) for possible constants to pass.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

CreateCustomWindow

Creates a custom window based on a registered toolbox object class or a custom window root view.

```
OSStatus CreateCustomWindow (
   const WindowDefSpec *def,
   WindowClass windowClass,
   WindowAttributes attributes,
   const Rect *contentBounds,
   WindowRef *outWindow
);
```

Parameters

def

For information on this data type, see `WindowDefSpec` (page 182).

windowClass

The class the custom window should belong to. This value determines the layer ordering of the custom window.

attributes

Attributes for the window. See "Window Attributes" (page 194) for a list of possible attributes.

contentBounds

Pointer to a `Rect` structure in global coordinates indicating the dimensions of the window's content region.

outWindow

On return, the newly-created window.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

See Also

`HIWindowCreate` (page 92)

Declared In

`MacWindows.h`

CreateNewWindow

Creates a window from parameter data.

```
OSStatus CreateNewWindow (
   WindowClass windowClass,
   WindowAttributes attributes,
   const Rect *contentBounds,
   WindowRef *outWindow
);
```

Parameters

windowClass

A constant that categorizes the class of window to be created. For certain classes, the window class can be altered after the window is created by calling `HIWindowChangeClass` (page 87). See "Window Class Constants" (page 184) for a description of possible values for this parameter.

attributes

Attributes for the window. See "Window Attributes" (page 194) for a list of possible attributes.

contentBounds

Pointer to a `Rect` structure in global coordinates indicating the dimensions of the window's content region.

outWindow

On input, a pointer to a value of type `WindowRef`. On return, the window pointer points to the newly created window.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Discussion

The `CreateNewWindow` function creates a window based on the attributes and class you specify in the `attributes` and `windowClass` parameters. `CreateNewWindow` sets the new window's content region to the size and location specified by the rectangle passed in the `bounds` parameter, which in turn determines the dimensions of the entire window. The Window Manager creates the window invisibly and places it at the front of the window's window group. After calling `CreateNewWindow`, you should set any desired associated data—using Window Manager or Control Manager accessor functions—then call the function `TransitionWindow` (page 162) or `ShowWindow` (page 156) to display the window.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

See Also

`HIWindowCreate` (page 92)

Related Sample Code

CarbonSketch

Declared In

`MacWindows.h`

CreateStandardWindowMenu

Creates a standard window menu for your application.

```
OSStatus CreateStandardWindowMenu (
   OptionBits inOptions,
   MenuRef *outMenu
);
```

Parameters

inOptions

Option bits. Pass 0 or `kWindowMenuIncludeRotate`. For information on the `kWindowMenuIncludeRotate` constant, see "Window Menu Item Property Constants" (page 238).

outMenu

On output, a new menu reference that contains the standard window menu items and commands.

Return Value

A result code. See "Window Manager Result Codes" (page 247). An operating system status code.

Discussion
You can call this function to create a window menu for your application. To register a window to be tracked by this menu, you either create your window with `CreateNewWindow` (page 41), passing the `kWindowInWindowMenuAttribute`, or you can use `ChangeWindowAttributes` (page 34) after the window is created. The Toolbox takes care of acting on the standard items such as zoom and minimize, as well as bringing selected windows to the front. All you need to do is insert the menu in your menu bar (typically at the end of your menu list) and register your windows, and the Toolbox does the rest.

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

Declared In
`MacWindows.h`

CreateWindowGroup

Creates a window group.

```
OSStatus CreateWindowGroup (
   WindowGroupAttributes inAttributes,
   WindowGroupRef *outGroup
);
```

Parameters

inAttributes
Attributes for the new window group. See "Window Group Attributes" (page 227) for a listing of possible attributes.

outGroup
For information on this data type, see WindowGroupRef (page 182).

Return Value
A result code. See "Window Manager Result Codes" (page 247).

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

Declared In
`MacWindows.h`

DebugPrintAllWindowGroups

Debugging utility function listing all window groups.

```
void DebugPrintAllWindowGroups (
   void
);
```

Carbon Porting Notes

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

Not available to 64-bit applications.

Declared In
HIToolboxDebugging.h

DebugPrintWindowGroup

Debugging utility functions for use with window groups.

```
void DebugPrintWindowGroup (
   WindowGroupRef inGroup
);
```

Parameters

inGroup
The window group. For information on this data type,

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

Declared In
HIToolboxDebugging.h

DetachSheetWindow

Detaches a sheet window from its parent window.

```
OSStatus DetachSheetWindow (
   WindowRef inSheet
);
```

Parameters

inSheet
The window sheet that is to be detached from its parent window.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Discussion
This function detaches a sheet window from its parent window without affecting the visibility or position of the sheet or its parent. This function is useful for hiding a sheet window without an animation effect. To do so, call DetachSheetWindow to detach the sheet from the parent, and then call HideWindow (page 84) to hide the sheet. Call DisposeWindow (page 45) to destroy the sheet.

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In
MacWindows.h

DisableScreenUpdates

Disables updates for changes to the current application's windows.

```
OSStatus DisableScreenUpdates (
   void
);
```

Return Value
A result code. See "Window Manager Result Codes" (page 247).

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

Declared In
`MacWindows.h`

DisposeWindow

Removes a window.

```
void DisposeWindow (
   WindowRef window
);
```

Parameters

window
On input, the window to be closed.

Discussion
The `DisposeWindow` function reduces a window's reference count by one. If the resulting reference count is zero, then `DisposeWindow` removes the window from the screen and deletes it from the window list, then releases the memory occupied by all structures associated with the window, including the window structure.

Note that `DisposeWindow` assumes that any picture pointed to by the window structure field `windowPic` is data, not a resource, and it calls the QuickDraw function `KillPicture` to delete it. If your application uses a picture stored as a resource, you must release the resource or release the memory it occupies with the `ReleaseResource` function and set the `windowPic` field to `NULL` before closing the window.

Any pending update events for the window are discarded. If the window being removed is the frontmost window, the window behind it, if any, becomes the active window.

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

Related Sample Code
HID Calibrator
HID Config Save
HID Explorer
QTCarbonShell
QTMetaData

Declared In
MacWindows.h

DragWindow

Moves a window on the screen when the user drags it by its drag region.

```
void DragWindow (
   WindowRef window,
   Point startPt,
   const Rect *boundsRect
);
```

Parameters

window

The window that is to be dragged.

startPt

On input, the location, in global coordinates, of the cursor at the time the user pressed the mouse button. Your application retrieves this point from the `where` field of the event structure.

boundsRect

On input, a pointer to a rectangle, given in global coordinates, that limits the region to which a window can be dragged. If the mouse button is released when the cursor is outside the limits of `boundsRect`, `DragWindow` returns without moving the window (or, if it was inactive, without making it the active window).

In CarbonLib and Mac OS X, this parameter can be `NULL` to indicate that there are no restrictions on window movement. This parameter is ignored by CarbonLib and Mac OS X v10.0 through v10.2; it is obeyed in Mac OX v10.3 and later.

Discussion
The `DragWindow` function moves the window around the screen, following the movement of the cursor until the user releases the mouse button. If the Command key was not pressed when the mouse button was pressed, `DragWindow` calls `SelectWindow` to make the window active before it drags the window. If the Command key was pressed when the mouse button was pressed, `DragWindow` moves the window without making it active.

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

Not available to 64-bit applications.

Related Sample Code
HideMenuBar

Declared In
MacWindows.h

EnableScreenUpdates

Enables screen updates for changes to the current application's windows.

```
OSStatus EnableScreenUpdates (
   void
);
```

Return Value
A result code. See "Window Manager Result Codes" (page 247).

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

Declared In
`MacWindows.h`

EndUpdate

Finishes updating a window.

```
void EndUpdate (
   WindowRef window
);
```

Parameters
window
The window for which updating is to be finished.

Discussion
The `EndUpdate` function restores the normal visible region of a window's graphics port. When you receive an update event for a window, you call `BeginUpdate` (page 31), redraw the update region, and then call `EndUpdate`. Each call to `BeginUpdate` must be balanced by a subsequent call to `EndUpdate`.

Special Considerations

This function should not be used on composited windows. Modifying a composited window's update region does not affect the area of the window to be drawn. A composited window does not use its window update region to control drawing. Instead, a composited window determines what to draw by looking at the invalid regions of the views contained in the window.

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

Related Sample Code
HideMenuBar
Simple DrawSprocket

Declared In
`MacWindows.h`

EndWindowProxyDrag

Disposes of the drag reference when the user completes the drag of a proxy icon.

```
OSStatus EndWindowProxyDrag (
   WindowRef window,
   DragRef theDrag
);
```

Parameters

window

The window whose proxy icon is being dragged.

theDrag

A value that refers to the current drag process. Pass in the value produced in the `outNewDrag` parameter of `BeginWindowProxyDrag`.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Discussion

Typically, if the proxy icon represents a type of object (currently, file system entities such as files, folders, and volumes) supported by the Window Manager, the Window Manager can handle all aspects of the drag process itself, and your application should call the function `TrackWindowProxyDrag` (page 159). However, if the proxy icon represents a type of data that the Window Manager does not support, or if you want to implement custom dragging behavior, your application should call the function `TrackWindowProxyFromExistingDrag` (page 160).

The `TrackWindowProxyFromExistingDrag` (page 160) function accepts an existing drag reference and adds file data if the window contains a file proxy. If your application uses `TrackWindowProxyFromExistingDrag`, you then have the choice of using this function in conjunction with the functions `BeginWindowProxyDrag` (page 32) and `EndWindowProxyDrag` or simply calling `TrackWindowProxyFromExistingDrag` and handling all aspects of creating and disposing of the drag yourself.

Specifically, your application can call `BeginWindowProxyDrag` to set up the drag image and drag reference. Your application must then track the drag, using `TrackWindowProxyFromExistingDrag`, and do any required moving of data and, finally, call `EndWindowProxyDrag` to dispose of the drag reference and its associated image data. The `EndWindowProxyDrag` function does not dispose of the region created for use by `BeginWindowProxyDrag`, however, so this remains the application's responsibility to dispose. The `EndWindowProxyDrag` function should not be used for types handled by the Window Manager unless you want to implement custom dragging behavior for those types.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

FindWindow

Maps the location of the cursor to a part of the screen or a region of a window when your application receives a mouse-down event.

```
WindowPartCode FindWindow (
    Point thePoint,
    WindowRef *window
);
```

Parameters

thePoint

The point, in global coordinates, where the mouse-down event occurred. Your application retrieves this information from the `where` field of the event structure.

window

A pointer to the window in which the mouse-down event occurred. `FindWindow` produces `NULL` if the mouse-down event occurred outside a window.

Return Value

The location of the cursor when the user pressed the mouse button; see "Window Part Code Constants" (page 209).

Discussion

You typically call the function `FindWindow` whenever you receive a mouse-down event. The `FindWindow` function helps you dispatch the event by reporting whether the cursor was in the menu bar or in a window when the mouse button was pressed. If the cursor was in a window, the function will produce both a pointer to the window and a constant that identifies the region of the window in which the event occurred.

If you are using the Carbon event handlers to handle events, a faster way of getting the window and part that received a mouse-down event is to get the `kEventParamWindowRef` and `kEventParamWindowPartCode` parameters from the event.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

See Also

`HIWindowFindAtLocation` (page 94)

Related Sample Code

HideMenuBar

Simple DrawSprocket

Declared In

`MacWindows.h`

FindWindowOfClass

Finds a window of a specific class at the specified point onscreen.

```
OSStatus FindWindowOfClass (
   const Point *where,
   WindowClass inWindowClass,
   WindowRef *outWindow,
   WindowPartCode *outWindowPart
);
```

Parameters

where

The point, in global coordinates, at which to search for a window.

inWindowClass

The class of window for which to search. Passing `kAllWindowClasses` returns any window found at `where`.

outWindow

On return, a pointer to the window, if it is of the specified class. If no window was found, this value is `NULL`. Note that you can pass `NULL` for this parameter.

outWindowPart

On return, the part code of the window part under the mouse. If no window was found, this value is `inDesk`. Note that you can pass `NULL` for this parameter.

Return Value

A result code. If no window of the specified class is found at the specified point, this function returns `errWindowNotFound`. For other possible return values, see "Window Manager Result Codes" (page 247).

Discussion

This function is similar to `FindWindow` (page 48), but lets you restrict the search to windows of a particular class.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

See Also

`HIWindowFindAtLocation` (page 94)

Declared In

`MacWindows.h`

FrontNonFloatingWindow

Returns to the application the frontmost visible window that is not a floating window.

```
WindowRef FrontNonFloatingWindow (
   void
);
```

Return Value

The first visible window in the window list that is of a nonfloating class. See "Window Class Constants" (page 184) for a description of window classes.

Discussion
Your application should call the `FrontNonFloatingWindow` function when you want to identify the frontmost visible window that is not a floating window. If you want to identify the frontmost visible window, whether floating or not, your application should call the function `FrontWindow`.

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

Related Sample Code
CarbonSketch
QTCarbonShell

Declared In
`MacWindows.h`

GetAvailableWindowAttributes

Returns the window attributes that are valid for a window class

```
WindowAttributes GetAvailableWindowAttributes (
   WindowClass inClass
);
```

Parameters
`inClass`
The window class to query.

Return Value
The window attributes that are valid for the window class specified by `inClass`. See "Window Attributes" (page 194) for a list of possible attributes.

Discussion
Some window classes support different attributes on different platforms. For example, floating windows can have collapse boxes in Mac OS 9, but not in Mac OS X. The Window Manager returns an error if you attempt to create a window with attributes that aren't supported for the requested window class.

You can use this API to determine those attributes that are supported by the current platform and remove those attributes that are not supported by the current platform before calling `CreateNewWindow` (page 41).

Availability
Available in Mac OS X v10.1 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

GetAvailableWindowPositioningBounds

Obtains the available window positioning bounds.

```
OSStatus GetAvailableWindowPositioningBounds (
   GDHandle inDevice,
   Rect *outAvailableRect
);
```

Parameters

inDevice

The screen for which the available window positioning bounds are to be obtained.

outAvailableRect

On return, a pointer to the available bounds for the device specified by `inDevice`.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Discussion

The available window positioning bounds is that area on the screen inside which a window may be positioned without intersecting or overlapping the menu bar, Dock, or other UI provided by the operating system.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

GetAvailableWindowPositioningRegion

Obtains the available window positioning region.

```
OSStatus GetAvailableWindowPositioningRegion (
   GDHandle inDevice,
   RgnHandle ioRgn
);
```

Parameters

inDevice

The screen for which the available window positioning region is to be obtained.

ioRgn

On input, contains a preallocated `RgnHandle`. On return, the `RgnHandle` has been modified to contain the available region for the given screen.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Discussion

The available window positioning region is that area on the screen inside which a window may be positioned without intersecting or overlapping the menu bar, Dock, or other UI provided by the operating system. This function differs from `GetAvailableWindowPositioningBounds` (page 51) in that the bounds version removes the entire area that may theoretically be covered by the Dock, even if the Dock does not currently reach from edge to edge of the device on which it is positioned. The region version includes the area at the sides of the Dock that is not covered by the Dock in the available region.

Availability

Available in Mac OS X v10.2 and later.

Not available to 64-bit applications.

Declared In
MacWindows.h

GetDrawerCurrentEdge

Obtains the current window edge from which the drawer appears.

```
OptionBits GetDrawerCurrentEdge (
   WindowRef inDrawerWindow
);
```

Parameters

inDrawerWindow
The drawer window whose window edge is to be obtained.

Return Value
The current window edge. See "Window Edge Constants" (page 237) for a list of possible return values.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
MacWindows.h

GetDrawerOffsets

Obtains the positioning offsets of a drawer.

```
OSStatus GetDrawerOffsets (
   WindowRef inDrawerWindow,
   CGFloat *outLeadingOffset,
   CGFloat *outTrailingOffset
);
```

Parameters

inDrawerWindow
The drawer window whose positioning offsets are to be obtained.

outLeadingOffset
On exit, a pointer to the drawer's leading offset. Pass NULL if you don't need this information.

outTrailingOffset
On exit, a pointer to the drawer's trailing offset. Pass NULL if you don't need this information.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
MacWindows.h

GetDrawerParent

Obtains the parent window of a drawer.

```
WindowRef GetDrawerParent (
   WindowRef inDrawerWindow
);
```

Parameters

inDrawerWindow
The drawer window whose parent window is to be obtained.

Return Value
The window that is the parent of the drawer specified by inDrawerWindow.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
MacWindows.h

GetDrawerPreferredEdge

Obtains the preferred opening edge for a drawer.

```
OptionBits GetDrawerPreferredEdge (
   WindowRef inDrawerWindow
);
```

Parameters

inDrawerWindow
The drawer window whose preferred opening edge is to be obtained.

Return Value
See "Window Edge Constants" (page 237) for a list of possible values.

Discussion
Note that the preferred edge may not be the same as the current edge, due to window positioning. For example, the right edge may be the preferred edge, but if the window is placed such that the right edge is offscreen, the drawer will appear on the left edge instead.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
MacWindows.h

GetDrawerState

Determines the current state of the drawer.

```
WindowDrawerState GetDrawerState (
   WindowRef inDrawerWindow
);
```

Parameters

inDrawerWindow

The drawer window whose state is to be determined.

Return Value

See "Drawer State Constants" (page 236) for a list of possible values.

Availability

Available in Mac OS X v10.2 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

GetFrontWindowOfClass

Obtains the frontmost window of a given class.

```
WindowRef GetFrontWindowOfClass (
   WindowClass inWindowClass,
   Boolean mustBeVisible
);
```

Parameters

inWindowClass

The class of the window you want to obtain. If you pass `kAllWindowClasses`, the function returns the frontmost window in the window list.

mustBeVisible

If set to `true`, the function returns the frontmost visible window. If set to `false`, the function returns the frontmost window of the specified class, regardless of whether the window is visible.

Return Value

A reference to the frontmost window of the class specified by `inWindowClass`.

Carbon Porting Notes

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

HID Config Save

QTCarbonShell

QTMetaData

Declared In

`MacWindows.h`

GetIndexedWindow

Obtains the window at the given index in the window group.

```
OSStatus GetIndexedWindow (
   WindowGroupRef inGroup,
   ItemCount inIndex,
   WindowGroupContentOptions inOptions,
   WindowRef *outWindow
);
```

Parameters

inGroup

The window group. For information on this data type,

inIndex

The index of the window. This parameter may range from `1` to the value returned by `CountWindowGroupContents`.

inOptions

Options for determining the number of windows. See "Window Group Content Options" (page 228) for possible values.

outWindow

The window at the index specified by `inIndex` in the group specified by `inGroup`.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

GetNextWindow

Returns the next window in a window list.

```
WindowRef GetNextWindow (
    WindowRef window
);
```

Parameters

window

The window to start from.

Return Value

The next window in a window list.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

GetNextWindowOfClass

Obtains the next window in a given window group.

```
WindowRef GetNextWindowOfClass (
   WindowRef inWindow,
   WindowClass inWindowClass,
   Boolean mustBeVisible
);
```

Parameters

inWindow

The window at which to start.

inWindowClass

The class of window to obtain. If you pass `kAllWindowClasses`, the function returns the window directly behind the input window. If no windows exist behind the front window, the function returns `NULL`.

mustBeVisible

If set to `true`, this function returns the next visible window of the specified window class. If set to `false`, this function returns the next window of the specified window class, regardless of whether it is visible.

Return Value

A reference for the next window of the specified class after the window specified by `inWindow`.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

QTCarbonShell

Declared In

`MacWindows.h`

GetPreviousWindow

Returns the window above the specified window in the window list.

```
WindowRef GetPreviousWindow (
   WindowRef inWindow
);
```

Parameters

inWindow

The window at which to start.

Return Value

A reference for the previous window of the specified class.

Availability

Available in Mac OS X v10.2 and later.

Not available to 64-bit applications.

Declared In
MacWindows.h

GetSheetWindowParent

Obtains the parent window of a sheet.

```
OSStatus GetSheetWindowParent (
   WindowRef inSheet,
   WindowRef *outParentWindow
);
```

Parameters

inSheet

The window sheet whose parent is to be obtained.

outParentWindow

A pointer to the reference for the parent of the window sheet specified by `inSheet`.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

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

Declared In
MacWindows.h

GetUserFocusWindow

Returns the current user focus window.

```
WindowRef GetUserFocusWindow (
    void
);
```

Return Value
The window receiving user focus.

Discussion
This function returns the window that receives menu commands and keyboard input as part of the standard event dispatching.

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

Declared In
MacWindows.h

GetWindowActivationScope

Obtains a window's activation scope.

```
OSStatus GetWindowActivationScope (
   WindowRef inWindow,
   WindowActivationScope *outScope
);
```

Parameters

inWindow

The window whose activation scope is to be obtained.

outScope

On return, a pointer to the window's activation scope.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

MacWindows.h

GetWindowAlpha

Returns the current alpha channel value for the window.

```
OSStatus GetWindowAlpha (
   WindowRef inWindow,
   CGFloat *outAlpha
);
```

Parameters

inWindow

The window for which the value of the alpha channel is to be obtained.

outAlpha

The current alpha value. This value can range from 0.0 (completely transparent) to 1.0 (opaque).

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Availability

Available in Mac OS X v10.2 and later.

Not available to 64-bit applications.

Declared In

MacWindows.h

GetWindowAttributes

Obtains the attributes of a window.

```
OSStatus GetWindowAttributes (
   WindowRef window,
   WindowAttributes *outAttributes
);
```

Parameters

window

The window whose attributes you want to obtain.

outAttributes

On input, a pointer to an unsigned 32-bit value of type `WindowAttributes`. On return, the bits are set to the attributes of the specified window. See "Window Attributes" (page 194) for a description of possible attributes.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Discussion

Window attributes specify a window's features (such as whether the window has a close box) and logical attributes (such as whether the window receives update and activate events).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

GetWindowBounds

Obtains the size and position of the bounding rectangle of the specified window region.

```
OSStatus GetWindowBounds (
   WindowRef window,
   WindowRegionCode regionCode,
   Rect *globalBounds
);
```

Parameters

window

The window whose bounds you want to obtain.

regionCode

A constant identifying the window region whose bounds you want to obtain. See "Window Region Constants" (page 217) for a list of possible values.

globalBounds

A pointer to a structure of type `Rect`. On return, the rectangle contains the dimensions and position, in global coordinates, of the window region specified in the `regionCode` parameter.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Discussion

Use the `GetWindowBounds` function to obtain the bounding rectangle for the specified window region for the specified window.

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

See Also
`HIWindowGetBounds` (page 97)

Related Sample Code
QTCarbonShell

Declared In
`MacWindows.h`

GetWindowCancelButton

Returns the current Cancel button for a window.

```
OSStatus GetWindowCancelButton (
    WindowRef inWindow,
    ControlRef *outControl
);
```

Parameters

inWindow
The window whose Cancel button you want to obtain.

outControl
A pointer to a control. On output, the control is the Cancel button.

Return Value
A result code.

Discussion
You can use this function to determine which button or control is the specified Cancel button for a given window. This button would be considered to have been clicked if the user instead presses Command-period or the Escape key.

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

Declared In
`MacWindows.h`

GetWindowClass

Obtains the class of a window.

```
OSStatus GetWindowClass (
   WindowRef window,
   WindowClass *outClass
);
```

Parameters

window

The window whose class you want to obtain.

outClass

On input, a pointer to a value of type `WindowClass`. On return, this value identifies the class of the specified window. See "Window Class Constants" (page 184) for a list of possible window classes. In Mac OS 8 and Mac OS 9, for windows not originally created by `CreateNewWindow` (page 41), the class pointed to by the `outClass` parameter is always identified by the constant `kDocumentWindowClass`.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

GetWindowContentColor

Obtains the color to which a window's content region is redrawn.

```
OSStatus GetWindowContentColor (
   WindowRef window,
   RGBColor *color
);
```

Parameters

window

The window whose content color is being retrieved.

color

On input, a pointer to an `RGBColor` structure. On return, the structure contains the content color for the specified window.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Discussion

The `GetWindowContentColor` function obtains the color to which the window's content region is redrawn.

See also the function `SetWindowContentColor` (page 138).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
MacWindows.h

GetWindowContentPattern

Obtains the pattern to which a window's content region is redrawn.

```
OSStatus GetWindowContentPattern (
   WindowRef window,
   PixPatHandle outPixPat
);
```

Parameters

window

The window whose content pattern is being retrieved.

outPixPat

On input, a handle to a structure of type PixPat. On return, the structure contains a copy of the content pattern data for the specified window, which your application is responsible for disposing.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Discussion
The GetWindowContentPattern function obtains the pattern to which the window's content region is redrawn.

See also the function SetWindowContentPattern (page 138).

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

Declared In
MacWindows.h

GetWindowDefaultButton

Returns the current default button for a window.

```
OSStatus GetWindowDefaultButton (
    WindowRef inWindow,
    ControlRef *outControl
);
```

Parameters

inWindow

The window whose default button you want to obtain.

outControl

A pointer to a control. On output, the control is the default button.

Return Value
A result code.

Discussion
You can use this function to determine which button or control is the default for a given window. This button would be considered to have been clicked if the user instead presses the Return or Enter keys on the keyboard.

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

Declared In
`MacWindows.h`

GetWindowDockTileMenu

Returns the menu to be displayed by a window's dock tile.

```
MenuRef GetWindowDockTileMenu (
   WindowRef inWindow
);
```

Parameters

`inWindow`
The window whose menu is to be obtained.

Return Value
The menu reference for the window specified by `inWindow`. See the Menu Manager documentation for a description of the `MenuRef` data type.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

GetWindowFeatures

Obtains the features that a window supports.

```
OSStatus GetWindowFeatures (
   WindowRef window,
   UInt32 *outFeatures
);
```

Parameters

`window`
A pointer to the window to be examined.

`outFeatures`
On input, a pointer to an unsigned 32-bit value. On return, the bits of the value specify the features the window supports; see "Window Feature Bits" (page 207).

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Discussion
The `GetWindowFeatures` function produces a window definition function's features in response to a `kWindowMsgGetFeatures` message.

Instead of calling this function, most applications should call `GetWindowAttributes` (page 59) to check for specific attributes, such as `kWindowCollapseBoxAttribute` and `kWindowResizableAttribute`.

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

Declared In
`MacWindows.h`

GetWindowFromPort

Gets a window reference from a `CGrafPtr` data type.

```
WindowRef GetWindowFromPort (
   CGrafPtr port
);
```

Parameters

port
The port to query.

Return Value
The window reference obtained from the port specified by `port`, or `NULL` if the `port` parameter is not actually attached to a window.

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

Declared In
`MacWindows.h`

GetWindowGreatestAreaDevice

Returns the graphics device with the greatest area of intersection with a specified window region.

```
OSStatus GetWindowGreatestAreaDevice (
   WindowRef inWindow,
   WindowRegionCode inRegion,
   GDHandle *outGreatestDevice,
   Rect *outGreatestDeviceRect
);
```

Parameters

inWindow
The window to compare against.

inRegion

The window region to compare against. See "Window Region Constants" (page 217) for a list of possible values.

outGreatestDevice

On return, the graphics device with the greatest intersection. May be `NULL`.

outGreatestDeviceRect

On return, the bounds of the graphics device with the greatest intersection. May be `NULL`. If the device with the greatest intersection also contains the menu bar, the device rect will exclude the menu bar area.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

GetWindowGroup

Obtains the window group associated with a window.

```
WindowGroupRef GetWindowGroup (
   WindowRef inWindow
);
```

Parameters

inWindow

The window whose window group is to be obtained.

Return Value

The window group reference for the window specified by `inWindow`. For information on this data type, see `WindowGroupRef` (page 182).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

GetWindowGroupAttributes

Obtains the attributes of a window group.

```
OSStatus GetWindowGroupAttributes (
   WindowGroupRef inGroup,
   WindowGroupAttributes *outAttributes
);
```

Parameters

inGroup

The window group whose attributes are to be changed. For information on this data type,

outAttributes

On return, the attributes of the group. See "Window Group Attributes" (page 227) for a list of possible values.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

GetWindowGroupContents

Obtains the contents of a window group.

```
OSStatus GetWindowGroupContents (
   WindowGroupRef inGroup,
   WindowGroupContentOptions inOptions,
   ItemCount inAllowedItems,
   ItemCount *outNumItems,
   void **outItems
);
```

Parameters

inGroup

The window group whose contents you want to obtain. For information on this data type, see `WindowGroupRef` (page 182).

inOptions

Options for determining how to count the group members. See "Window Group Content Options" (page 228) for a list of possible values.

inAllowedItems

The number of items that will fit in `outItems`.

outNumItems

On return, the number of items in the group.

outItems

On entry, this parameter must be a pointer to a pre-allocated buffer in which the window group contents (either window references or window group references) are to be placed. On return, the buffer pointed to by this parameter contains the requested window references or window group references.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

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

Declared In
`MacWindows.h`

GetWindowGroupLevel

Obtains the level of the group in the window class hierarchy.

```
OSStatus GetWindowGroupLevel (
   WindowGroupRef inGroup,
   SInt32 *outLevel
);
```

Parameters

inGroup
The window group. For information on this data type,

outLevel
On exit, the window level of the windows in this group.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Discussion
The window group's level is only used to set the level of its windows if the window group is a child of the root group. If there is another group in the group hierarchy between this group and the root group, this group's level is ignored.

See the Core Graphics frameworks header `CGWindowLevel.h` for a listing of window levels.

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

Declared In
`MacWindows.h`

GetWindowGroupLevelOfType

Obtains the Core Graphics window level of a window group.

```
OSStatus GetWindowGroupLevelOfType (
   WindowGroupRef inGroup,
   UInt32 inLevelType,
   CGWindowLevel *outLevel
);
```

Parameters

inGroup
The window group whose Core Graphics window level is to be obtained.

inLevelType

The level type to obtain. Specify `kWindowGroupLevelActive`, `kWindowGroupLevelInactive`, or `kWindowGroupLevelPromoted`. For details, see "Window Group Level Constants" (page 244).

outLevel

On output, the Core Graphics window level for the windows in this group.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Discussion
In Mac OS X v10.4 and later, multiple Core Graphics window levels may be associated with a window group: one level for when the application is active and another for when the application is inactive. The Window Manager automatically switches each group's Core Graphics window level as the application becomes active or inactive. Use `GetWindowGroupLevelOfType` to get each Core Graphics window level associated with a window group, including the promoted window level that is actually in use for windows in the group when the application is active.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

GetWindowGroupOfClass

Obtains the window group corresponding to a given window class.

```
WindowGroupRef GetWindowGroupOfClass (
   WindowClass windowClass
);
```

Parameters

windowClass

The window class to query.

Return Value
For information on this data type, see `WindowGroupRef` (page 182).

Discussion
Each window class has an associated pre-defined window group. This function returns the window group reference for the window group that is associated with `windowClass`. Note that all windows in a group do not have to be of the same window class.

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

Declared In
`MacWindows.h`

GetWindowGroupOwner

Obtains the window that owns a window group. (if any)

```
WindowRef GetWindowGroupOwner (
   WindowGroupRef inGroup
);
```

Parameters

inGroup

The window group to query. For information on this data type,

Return Value

The window reference for the window that owns the group specified by `inGroup`.

Discussion

You call `SetWindowGroupOwner` (page 143) to associate a window group with a particular window.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

GetWindowGroupParent

Obtains the parent group of a window group.

```
WindowGroupRef GetWindowGroupParent (
   WindowGroupRef inGroup
);
```

Parameters

inGroup

The window group whose parent is to be obtained.

Return Value

The parent of the window group specified by `inGroup`.

Discussion

You can nest window groups within each other.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

GetWindowGroupRetainCount

Determines the current reference count for a window group.

```
ItemCount GetWindowGroupRetainCount (
   WindowGroupRef inGroup
);
```

Parameters

inGroup

The window group for which the current reference count is to be obtained. For information on this data type, see `WindowGroupRef` (page 182).

Availability

Available in Mac OS X v10.1 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

GetWindowGroupSibling

Obtains the next or previous group of a window group.

```
WindowGroupRef GetWindowGroupSibling (
   WindowGroupRef inGroup,
   Boolean inNextGroup
);
```

Parameters

inGroup

The window group for which the next or previous group is to be obtained. For information on this data type, see `WindowGroupRef` (page 182).

inNextGroup

Pass `true` to obtain the next sibling; `false` to obtain the previous sibling.

Return Value

The next or previous group. For information on this data type, see `WindowGroupRef` (page 182).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

GetWindowIdealUserState

Obtains the size and position of a window in its user state.

```
OSStatus GetWindowIdealUserState (
    WindowRef inWindow,
    Rect *outUserState
);
```

Parameters

inWindow

The window for which you want to obtain the user state.

outUserState

On input, a pointer to a structure of type `Rect`. On return, this rectangle specifies the current size and position of the window's user state, in global coordinates.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Discussion

Because the window definition function relies upon the `WStateData` structure, it is unaware of the ideal standard state, and this causes the user state data that it stores in the `WStateData` structure to be unreliable. While the Window Manager is reliably aware of the window's zoom state, it cannot record the current user state in the `WStateData` structure, because the window definition function can overwrite that data. Therefore, the function `ZoomWindowIdeal` (page 168) maintains the window's user state independently of the `WStateData` structure. The `GetWindowIdealUserState` function gives your application access to the user state data maintained by `ZoomWindowIdeal`. However, your application should not typically need to use this function; it is supplied for completeness.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

See Also

`HIWindowGetIdealUserState` (page 99)

Declared In

`MacWindows.h`

GetWindowIndex

Obtains the index number of a specified window in a group.

```
OSStatus GetWindowIndex (
   WindowRef inWindow,
   WindowGroupRef inStartGroup,
   WindowGroupContentOptions inOptions,
   ItemCount *outIndex
);
```

Parameters

inWindow

The window whose window group index number is to be obtained.

inStartGroup

The window group to query.

inOptions

Specifies how to enumerate the specified window; `kWindowGroupContentsReturnWindows` is implied and does not need to be specified explicitly.

outIndex

A pointer to a variable that, on return, contains the window's z-order index. The frontmost window in a window group has an index of 1. Window indexes increase as the window gets lower in z-order (that is, visually further from the top of the window list and closer to the desktop.)

Return Value
A result code. See "Window Manager Result Codes" (page 247).

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

Declared In
`MacWindows.h`

GetWindowKind

Returns a window's window kind.

```
short GetWindowKind (
   WindowRef window
);
```

Parameters

window

The window whose window kind is to be returned.

Return Value
An integer representing the window kind; see "Window Kinds" (page 225).

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

Declared In
`MacWindows.h`

GetWindowList

Obtains the first window in a window list.

```
WindowRef GetWindowList (
   void
);
```

Return Value
A window reference for the first window in the list.

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

Declared In
MacWindows.h

GetWindowModality

Obtains the modality of a window.

```
OSStatus GetWindowModality (
   WindowRef inWindow,
   WindowModality *outModalKind,
   WindowRef *outUnavailableWindow
);
```

Parameters

inWindow
The window whose modality is to be obtained.

outModalKind
On return, contains the modality of the window.

outUnavailableWindow
On return, if the window is window-modal, contains the target window of the specified window's modality.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

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

Declared In
MacWindows.h

GetWindowPort

Gets the window's color graphics port.

```
CGrafPtr GetWindowPort (
   WindowRef window
);
```

Parameters

window
The window whose color graphics port is to be obtained.

Return Value
A pointer to the window's color graphics port. See the QuickDraw Manager documentation for a description of the `CGrafPtr` data type.

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

Related Sample Code
CarbonSketch
QTCarbonShell

Declared In
`MacWindows.h`

GetWindowPortBounds

Obtains the bounds of the window port.

```
Rect * GetWindowPortBounds (
   WindowRef window,
   Rect *bounds
);
```

Parameters

window
The window whose port bounds you want.

bounds
A pointer to a `Rect` structure. On return, the `Rect` structure contains the bounds of the window port.

Return Value
The same value (pointer to a `Rect` structure) that was passed to `GetWindowPortBounds` in the `bounds` parameter.

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

Related Sample Code
CarbonSketch
HID Explorer

Declared In
`MacWindows.h`

GetWindowProperty

Obtains a piece of data that is associated with a window.

```
OSStatus GetWindowProperty (
   WindowRef window,
   PropertyCreator propertyCreator,
   PropertyTag propertyTag,
   ByteCount bufferSize,
   ByteCount *actualSize,
   void *propertyBuffer
);
```

Parameters

window
The window to be examined for associated data.

propertyCreator
The creator code (typically, the application's signature) of the associated data to be obtained.

propertyTag
The application-defined code identifying the associated data to be obtained.

bufferSize
The size of the associated data to be obtained. If the size of the data is unknown, use the function `GetWindowPropertySize` (page 77) to get the data's size. If the size specified does not match the actual size of the property, `GetWindowProperty` only retrieves data up to the size specified or up to the actual size of the property, whichever is smaller, and an error is returned.

actualSize
On input, a pointer to a value. On return, the value specifies the actual size of the obtained data. You may pass `NULL` for the `actualSize` parameter if you are not interested in this information.

propertyBuffer
On input, a pointer to a buffer. On return, this buffer contains a copy of the data that is associated with the specified window.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Discussion
The data retrieved by the `GetWindowProperty` function must have been previously associated with the window with the function `SetWindowProperty` (page 147).

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

Related Sample Code
QTMetaData

Declared In
`MacWindows.h`

GetWindowPropertyAttributes

Obtains the attributes of a window property.

```
OSStatus GetWindowPropertyAttributes (
   WindowRef window,
   OSType propertyCreator,
   OSType propertyTag,
   OptionBits *attributes
);
```

Parameters

window
The window having a property whose attributes are to be obtained.

propertyCreator
The property creator.

propertyTag
The property tag.

attributes

On return, the property's attributes. Currently, the only valid property is `kWindowPropertyPersistent`. For a description of this property, see "Window Property Persistent Constant" (page 220).

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Carbon Porting Notes

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

Declared In
`MacWindows.h`

GetWindowPropertySize

Obtains the size of a piece of data that is associated with a window.

```
OSStatus GetWindowPropertySize (
   WindowRef window,
   PropertyCreator creator,
   PropertyTag tag,
   ByteCount *size
);
```

Parameters

window

The window to be examined for associated data.

creator

The creator code (typically, the application's signature) of the associated data whose size is to be obtained.

tag

The application-defined code identifying the associated data whose size is to be obtained.

size

A pointer to a value that, on return, specifies the size of the associated data.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Discussion
If you want to retrieve a piece of associated data with the `GetWindowProperty` (page 75) function, you typically need to use the `GetWindowPropertySize` function to determine the size of the data beforehand.

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

Declared In
`MacWindows.h`

GetWindowProxyAlias

Obtains an alias for the file that is associated with a window.

```
OSStatus GetWindowProxyAlias (
   WindowRef window,
   AliasHandle *alias
);
```

Parameters

window

The window for which you want to determine the associated file.

alias

On input, a pointer to a value of type `AliasHandle`. On return, the `AliasRecord` structure referenced by the alias handle contains a copy of the alias data for the file associated with the specified window. Your application must dispose of this handle.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Discussion

Your application can call the `GetWindowProxyAlias` function to retrieve alias data for the file associated with a window.

See also the function `SetWindowProxyAlias` (page 148).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

GetWindowProxyIcon

Obtains a window's proxy icon.

```
OSStatus GetWindowProxyIcon (
   WindowRef window,
   IconRef *outIcon
);
```

Parameters

window

The window for which you want to obtain the proxy icon.

outIcon

A pointer to a variable of type `IconRef` that, on return, identifies the window's proxy icon. Your application must not dispose of this icon.

Return Value

A result code. If no proxy icon is found, this function returns `errWindowDoesNotHaveProxy`. For other possible return values, see "Window Manager Result Codes" (page 247).

Discussion

There are several different ways to associate a proxy icon with a window:

- If you use the function `SetWindowProxyIcon` (page 149), `GetWindowProxyIcon` returns the proxy icon you set.
- If you use the function `SetWindowProxyCreatorAndType` (page 148), that function uses Icon Services to find and set the proxy icon corresponding to the creator and type. `GetWindowProxyIcon` returns that icon.
- If you use `SetWindowProxyAlias` (page 148), `SetWindowProxyFSSpec` (page 282), or `HIWindowSetProxyFSRef` (page 106), then `GetWindowProxyIcon` attempts to resolve the alias (if available) and returns the icon associated with the specified file.

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

Declared In
`MacWindows.h`

GetWindowResizeLimits

Returns the minimum and maximum content sizes for a window.

```
OSStatus GetWindowResizeLimits (
   WindowRef inWindow,
   HISize *outMinLimits,
   HISize *outMaxLimits
);
```

Parameters

`inWindow`
The window whose minimum and maximum content sizes are to be obtained.

`outMinLimits`
On return, the window's minimum content size. Pass `NULL` if you don't want this information. For information on the `HISize` data type, see HIGeometry.h.

`outMaxLimits`
On return, the window's maximum content size. Pass `NULL` if you don't want this information. For information on the `HISize` data type, see HIGeometry.h.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

GetWindowStandardState

Obtains a window's standard zoom rectangle.

```
Rect * GetWindowStandardState (
   WindowRef window,
   Rect *rect
);
```

Parameters

window

The window whose standard zoom rectangle is to be obtained.

rect

On input, a pointer to a `Rect` structure. On return, the `Rect` structure contains the window's standard zoom rectangle, in global coordinates. A window's standard zoom rectangle is the window content bounds when the window is zoomed out to its greatest extent.

Return Value

The same value (pointer to a `Rect` structure) that was passed to `GetWindowStandardState` in the `rect` parameter.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

GetWindowStructurePort

Obtains a graphics port that is used when drawing a window's structure.

```
CGrafPtr GetWindowStructurePort (
   WindowRef inWindow
);
```

Parameters

inWindow

The window to query.

Return Value

The CGrafPtr that is used when drawing the window's structure (window frame).

Availability

Available in Mac OS X v10.1 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

GetWindowStructureWidths

Obtains the width of the structure region on each edge of a window.

```
OSStatus GetWindowStructureWidths (
   WindowRef inWindow,
   Rect *outRect
);
```

Parameters

inWindow

The window to query.

outRect

On return, the `Rect` structure is filled in with the widths of the structure.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

GetWindowToolbar

Obtains the toolbar associated with a window.

```
OSStatus GetWindowToolbar (
   WindowRef inWindow,
   HIToolbarRef *outToolbar
);
```

Parameters

inWindow

The window whose toolbar is to be obtained.

outToolbar

On return, the toolbar that is attached to the window, or `NULL` if the window has no toolbar.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Availability

Available in Mac OS X v10.2 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

GetWindowUserState

Returns a window's user zoom rectangle.

```
Rect * GetWindowUserState (
   WindowRef window,
   Rect *rect
);
```

Parameters

window

The window whose user zoom rectangle is to be returned.

rect

On input, a pointer to a `Rect` structure. On return, the `Rect` structure contains the window's user zoom rectangle, in global coordinates. A window's user zoom rectangle is the window content bounds when the window is zoomed back in.

Return Value

The same value (pointer to a `Rect` structure) that was passed to `GetWindowUserState` in the `rect` parameter.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

GetWindowWidgetHilite

Obtains the window part code of the window widget that is currently highlighted.

```
OSStatus GetWindowWidgetHilite (
   WindowRef inWindow,
   WindowDefPartCode *outHilite
);
```

Parameters

inWindow

The window to query.

outHilite

The highlight.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

GetWRefCon

Returns the reference constant from a window.

```
SRefCon GetWRefCon (
   WindowRef window
);
```

Parameters

window

The window whose reference constant is to be returned.

Return Value

The long integer data stored in the `refCon` field of the window structure specified in the `window` parameter.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

CarbonSketch

QTCarbonShell

Declared In

`MacWindows.h`

HideFloatingWindows

Hides an application's floating windows.

```
OSStatus HideFloatingWindows (
   void
);
```

Return Value

A result code. For details, see "Window Manager Result Codes" (page 247).

Discussion

When an application receives a suspend event, its floating windows are hidden automatically. When the application receives a resume event, the floating windows are made visible automatically. Call this function if you want to hide your floating windows manually.

See also the function `ShowFloatingWindows` (page 153).

Special Considerations

The `HideFloatingWindows` function operates only upon windows created with the `kFloatingWindowClass` constant; see "Window Class Constants" (page 184) for more details on this constant.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

HideSheetWindow

Hides a sheet window using appropriate visual effects.

```
OSStatus HideSheetWindow (
   WindowRef inSheet
);
```

Parameters

inSheet

The window sheet that is to be hidden.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

HideWindow

Makes a window invisible.

```
void HideWindow (
   WindowRef window
);
```

Parameters

window

The window that is to be made invisible.

Discussion

The `HideWindow` function make a visible window invisible. If you hide the frontmost window, `HideWindow` removes the highlighting, brings the window behind it to the front, highlights the new frontmost window, and generates the appropriate activate events.

To reverse the actions of `HideWindow`, you must call both `ShowWindow` (page 156), to make the window visible, and `SelectWindow` (page 129), to select it.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

HiliteWindow

Sets a window's highlighting status.

```
void HiliteWindow (
   WindowRef window,
   Boolean fHilite
);
```

Parameters

window

On input, a pointer to the window structure.

fHilite

On input, a Boolean value that specifies the highlighting status: `true` highlights a window; `false` removes highlighting.

Discussion

The `HiliteWindow` function sets a window's highlighting status to the specified state. If the value of the `fHilite` parameter is `true`, `HiliteWindow` highlights the specified window; if the specified window is already highlighted, the function has no effect. If the value of `fHilite` is `false`, `HiliteWindow` removes highlighting from the specified window; if the window is not already highlighted, the function has no effect.

Your application doesn't normally need to call `HiliteWindow`. To make a window active, you can call `SelectWindow` or `ActivateWindow`, which handle highlighting for you.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

HiliteWindowFrameForDrag

Sets the highlight state of the window's structure region to reflect the window's validity as a drag-and-drop destination.

```
OSStatus HiliteWindowFrameForDrag (
   WindowRef window,
   Boolean hilited
);
```

Parameters

window

The window for which you want to set the highlight state.

hilited

Pass `true` if the window's frame should be highlighted otherwise, pass `false`.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Discussion

Applications typically call the Drag Manager functions `ShowDragHilite` and `HideDragHilite` to indicate that a window is a valid drag-and-drop destination. If your application does not do this—that is, if your application implements any type of custom drag highlighting, such as highlighting more than one area of a window at a time—it must call the `HiliteWindowFrameForDrag` function.

The `HiliteWindowFrameForDrag` function highlights a window's proxy icon when the user drags content inside the window that is a valid content type for that destination. The default behavior of system-defined windows is to highlight the proxy icon along with the window's content area when the window is a valid drag-and-drop destination. If you call the Drag Manager functions `ShowDragHilite` and `HideDragHilite`, you don't need to use `HiliteWindowFrameForDrag`.

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

Declared In
`MacWindows.h`

HIWindowChangeAttributes

Changes the attributes of a window.

```
OSStatus HIWindowChangeAttributes (
   WindowRef inWindow,
   const int *inAttrToSet,
   const int *inAttrToClear
);
```

Parameters

inWindow
The window to change.

inAttrToSet
A zero-terminated array of window attribute constants. Possible values are described in "Window Attribute Identifiers" (page 188). Each array entry specifies an attribute of the window to set. You may pass `NULL` if you do not wish to set any attributes.

inAttrToClear
A zero-terminated array of window attribute constants. Possible values are described in "Window Attribute Identifiers" (page 188). Each array entry specifies an attribute of the window to clear. You may pass `NULL` if you do not wish to clear any attributes.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Discussion
This function takes two arrays of window attribute constants, as described in "Window Attribute Identifiers" (page 188). The first array specifies the attributes to set, and the second specifies the attributes to clear. For example, you might call this function as follows:

```
int setAttr[] = { kHIWindowBitCloseBox, kHIWindowBitZoomBox, 0 };
int clearAttr[] = { kHIWindowBitNoTitleBar, 0 };
HIWindowChangeAttributes (window, setAttr, clearAttr);
```

Special Considerations

In Mac OS X v10.4 or earlier, you can use the function `ChangeWindowAttributes` (page 34) to achieve similar results.

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

Not available to 64-bit applications.

Declared In
MacWindows.h

HIWindowChangeAvailability

Changes the availability of a window during Exposé or in Spaces.

```
OSStatus HIWindowChangeAvailability (
   WindowRef inWindow,
   HIWindowAvailability inSetAvailability,
   HIWindowAvailability inClearAvailability
);
```

Parameters

inWindow
The window whose availability is to be changed.

inSetAvailability
The availability bits to set. For details, see "Window Availability Constants" (page 242).

inClearAvailability
The availability bits to clear. For details, see "Window Availability Constants" (page 242).

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Discussion
This function overrides the default behavior of the Window Manager in determining whether a window is visible during Exposé or in all Spaces workspaces. Most applications should not override the default behavior; these options should only be used in special cases. For example, accessibility assistance applications may need to create windows that are visible in all workspaces.

By default, newly created windows of class `kDocumentWindowClass` are given an availability of `0` (meaning that they are available during Exposé), and windows from all other window classes are given an availability of `kHIWindowExposeHidden`.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
MacWindows.h

HIWindowChangeClass

Changes the appearance and behavior of a window.

```
OSStatus HIWindowChangeClass (
   WindowRef inWindow,
   WindowClass inWindowClass
);
```

Parameters

window

The window whose class you want to change.

inClass

The new class that is to be applied to the window. See "Window Class Constants" (page 184) for a list of possible window classes.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Discussion

This function changes the class of a window. Unlike `SetWindowClass` (page 281), `HIWindowChangeClass` effectively changes both the appearance and behavior of the window.

This function can convert a window between `kDocumentWindowClass`, `kFloatingWindowClass`, `kUtilityWindowClass`, and `kMovableModalWindowClass` only. It cannot, for example, change a document window into a plain window.

The attributes of the window are adjusted to contain only those that are allowed for the new class. It is the caller's responsibility to adjust them further, as necessary, after `HIWindowChangeClass` returns.

Availability

Available in Mac OS X v10.3 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

HIWindowChangeFeatures

Changes a window's features.

```
OSStatus HIWindowChangeFeatures (
   WindowRef inWindow,
   UInt64 inSetThese,
   UInt64 inClearThese
);
```

Parameters

inWindow

The window whose features are to be changed.

inSetThese

The feature bits to set. For details, see "Window Feature Bits" (page 207).

inClearThese

The feature bits to clear. For details, see "Window Feature Bits" (page 207).

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Discussion
`HIWindowChangeFeatures` changes the features of a window on the fly. This function should only be used by custom window definitions or window frame views.

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

HIWindowConstrain

Moves and resizes a window to be within a specified bounding rectangle.

```
OSStatus HIWindowConstrain (
   WindowRef inWindowRef,
   WindowRegionCode inRegionCode,
   WindowConstrainOptions inOptions,
    HICoordinateSpace inSpace,
   const HIRect *inScreenBounds,
    const HISize *inMinimumSize,
   HIRect *ioBounds
);
```

Parameters

inWindowRef
The window to constrain.

inRegionCode
The window region to constrain. For a list of possible values, see "Window Region Constants" (page 217).

inOptions
Flags controlling how the window is constrained. For a list of possible options, see "Window Constrain Options" (page 224).

inSpace
The coordinate space in which the `inScreenBounds`, `inMinimumSize`, and `ioBounds` parameters are expressed. This parameter must be either `kHICoordSpaceScreenPixels` or `kHICoordSpace72DPIGlobal`.

inScreenBounds
A rectangle within which to constrain the window. You may pass `NULL` if you don't need to specify a screen bounds. If `NULL`, the window is constrained to the screen that has the greatest intersection with the specified window region.

inMinimumSize
A minimum size that should be kept within the specified screen bounds. This parameter is ignored if the `kWindowConstrainMoveMinimum` option is not set. Even if that option is set, you may still pass `NULL` if you don't need to customize the minimum dimensions.

ioBounds

If the `inOptions` parameter contains `kWindowConstrainUseSpecifiedBounds`, then this parameter should be a bounding rectangle of the specified window region. The bounding rectangle does not have to match the actual current bounds of the specified region; it may be a hypothetical bounds that you would like to constrain without actually moving the window to that location. On output, contains the new structure bounds of the window. You may pass `NULL` if you don't need the window bounds returned.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

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

Not available to 64-bit applications.

Declared In
`MacWindows.h`

HIWindowCopyAvailablePositioningShape

Copies the available window positioning shape on a display.

```
OSStatus HIWindowCopyAvailablePositioningShape (
   CGDirectDisplayID inDisplay,
   HICoordinateSpace inSpace,
   HIShapeRef *outShape
);
```

Parameters

inDisplay

The display for which to find the available shape. May be `kCGNullDirectDisplay` to request the shape of the main display.

inSpace

The coordinate space in which the positioning shape should be returned. This parameter must be either `kHICoordSpaceScreenPixel` or `kHICoordSpace72DPIGlobal`.

outShape

A pointer to a shape (an HIShape object). On output, the shape describes the available bounds for the specified display. This shape is returned in the specified coordinate space. You should release the shape when you no longer need it.

Discussion
This function finds the area on the display in which a window may be positioned without intersecting or overlapping the menu bar, Dock, or other UI provided by the operating system. This function differs from `HIWindowGetAvailablePositioningBounds` (page 96) in that the bounds version removes the entire area that may theoretically be covered by the Dock, even if the Dock does not currently reach from edge to edge of the display on which it is positioned. The shape version includes the area at the sides of the Dock that is not covered by the Dock.

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

Not available to 64-bit applications.

Declared In
`MacWindows.h`

HIWindowCopyDrawers

Obtains an array of the drawers that are attached to a window.

```
OSStatus HIWindowCopyDrawers (
    WindowRef inWindow,
    CFArrayRef *outDrawers
);
```

Parameters

inWindow

The parent window to access.

outDrawers

A pointer to a Core Foundation array. On output, each array entry is a drawer window attached to the parent window specified in the `inWindow` parameter. The array will be valid, but empty, if the parent window has no drawers. You should release the array when you no longer need it.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Availability

Available in Mac OS X v10.5 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

HIWindowCopyShape

Retrieves a shape that describes a region of a window.

```
OSStatus HIWindowCopyShape (
    WindowRef inWindow,
    WindowRegionCode inRegion,
    HICoordinateSpace inSpace,
    HIShapeRef *outShape
);
```

Parameters

inWindow

The window to access.

inRegion

The window region whose shape you want to obtain. For a list of possible values, see "Window Region Constants" (page 217).

inSpace

The coordinate space in which the shape should be returned. This parameter must be `kHICoordSpaceWindow`, `kHICoordSpaceScreenPixel`, or `kHICoordSpace72DPIGlobal`.

outShape

A pointer to a shape (an HIShape object). On output, the shape describes the specified window region. The shape is returned in the specified coordinate space. You should release the shape when you no longer need it. If the window does not support the specified window region, no shape is returned.

Return Value
A result code. If the window does not support the specified window region, the result returned is `errWindowRegionCodeInvalid`. For other possible values, see "Window Manager Result Codes" (page 247).

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

Declared In
`MacWindows.h`

HIWindowCreate

Creates a standard or custom window.

```
OSStatus HIWindowCreate (
    WindowClass inClass,
    const int *inAttributes,
    const WindowDefSpec *inDefSpec,
    HICoordinateSpace inSpace,
    const HIRect *inBounds,
    WindowRef *outWindow
);
```

Parameters

inClass
The class of window to be created. For a list of possible classes, see "Window Class Constants" (page 184).

inAttributes
A zero-terminated array of window attribute constants. Each array entry specifies an attribute of the window to set. You may pass `NULL` if you don't need to set any attributes. For a list of possible attributes, see "Window Attribute Identifiers" (page 188).

inDefSpec
A pointer to a custom window proc ID or root view for the window. You may pass `NULL` if you don't need to customize the window.

inSpace
The coordinate space in which the content bounds is expressed. This parameter must be either `kHICoordSpaceScreenPixels` or `kHICoordSpace72DPIGlobal`.

inBounds
A pointer to the bounds of the content area of the window in the coordinate space specified by the `inSpace` parameter. If you specify non-integral coordinates, they will be rounded to the nearest integral value in screen pixel space when the window is actually positioned or sized.

outWindow
A pointer to a window variable. On output, the variable contains the new window.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Discussion
This function makes it possible to create windows with content bounds expressed in different coordinate spaces. In Mac OS X v10.5 and later, you can use this function in place of `CreateNewWindow` (page 41) or `CreateCustomWindow` (page 40) to create a window from a set of parameters.

Most developers will want to work primarily in the 72 DPI coordinate space. Doing so makes your code independent of the current user interface scale factor, and eases source compatibility with earlier versions of Mac OS X that do not support resolution independence. However, there are also certain cases where your application must express your window's bounds in pixel coordinates; primarily when you need to position your windows so they exactly align with each other or with some other fixed location, such as the edge of the display. For these cases, you should use the screen pixel coordinate space.

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

Declared In
`MacWindows.h`

HIWindowCreateCollapsedDockTileContext

Creates a Quartz graphics context for drawing a collapsed window's Dock tile.

```
OSStatus HIWindowCreateCollapsedDockTileContext (
   WindowRef inWindow,
   CGContextRef *outContext,
   HISize *outContextSize
);
```

Parameters

inWindow
The collapsed window.

outContext
A pointer to a `CGContextRef` variable. On output, the variable contains the graphics context for drawing the window's Dock tile.

outContextSize
A pointer to a `HISize` structure. On output, the structure contains the width and height of the area in which to draw.

Return Value
A result code. If the window is not collapsed, the result code is `windowWrongStateErr`. For other possible values, see "Window Manager Result Codes" (page 247).

Discussion
When you are finished drawing in the graphics context, you should:

1. Call `CGContextFlush` to ensure that your drawing appears onscreen.
2. Call `HIWindowReleaseCollapsedDockTileContext` (page 104) to release the context. Do not call `CFRelease` or `CGContextRelease` to release the context, or you may leak system resources.

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

Not available to 64-bit applications.

Declared In
MacWindows.h

HIWindowFindAtLocation

Finds a window in the current process at a specified location.

```
OSStatus HIWindowFindAtLocation (
    const HIPoint *inLocation,
    HICoordinateSpace inSpace,
    WindowRef inStartWindow,
    OptionBits inOptions,
    WindowRef *outWindow,
    WindowPartCode *outWindowPart,
    HIPoint *outWindowLocation
);
```

Parameters

inLocation
The location, in global coordinates, at which to search for a window.

inSpace
The coordinate space in which the location is expressed. This parameter must be either kHICoordSpaceScreenPixel or kHICoordSpace72DPIGlobal.

inStartWindow
The window at which to start the search, inclusive. Pass kFirstWindowOfClass to start the search at the beginning of the window list. Passing NULL will cause the search to start at the end of the window list, and therefore no window will be found.

inOptions
Reserved. Pass zero.

outWindow
A pointer to a window variable. On output, the variable contains the window in the current process at the specified location, if any, or NULL if no window is found.

outWindowPart
A pointer to a window part code variable. On output, the variable contains the window part that was hit. You may pass NULL if you don't need this information.

outWindowLocation
A pointer to a point variable. On output, the variable contains the specified location transformed into window-relative coordinates, taking into account any window transform or magnification. You may pass NULL if you don't need this information.

Return Value
A result code. If no window is found that satisfies the search criteria, this function returns errWindowNotFound. For other possible return values, see "Window Manager Result Codes" (page 247).

Discussion
This function searches the window list of the current process for a window that contains the specified location. If you need to determine whether the window is of a particular class, you can use the function GetWindowClass (page 61) and compare the result to the desired class.

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

Declared In
`MacWindows.h`

HIWindowFlush

Flushes any dirty areas a window might have.

```
OSStatus HIWindowFlush (
   WindowRef inWindow
);
```

Parameters

window
The window to flush.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Discussion
This function allows you to manually flush dirty areas of a window to the screen. This is the preferred way to flush window buffers in Mac OS X v10.3 and later. If called for a composited window, this function also renders any views in the window that are invalid.

Availability
Available in Mac OS X v10.3 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

HIWindowFromCGWindowID

Returns the window in the current process with a specified Quartz window ID.

```
WindowRef HIWindowFromCGWindowID (
    CGWindowID inWindowID
);
```

Parameters

inWindowID
The window ID, as returned by `HIWindowGetCGWindowID` (page 98) or `CGWindowListCopyWindowInfo`.

Return Value
The window to which the window ID is assigned. This function returns `NULL` if the window ID is invalid or if it refers to a window in another process.

Discussion
This function returns the window in the current process to which the specified window ID is assigned by the window server when the window is created.

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

See Also
`HIWindowGetCGWindowID` (page 98)

Declared In
`MacWindows.h`

HIWindowGetAvailability

Obtains the availability of a window during Exposé or in Spaces.

```
OSStatus HIWindowGetAvailability (
   WindowRef inWindow,
   HIWindowAvailability *outAvailability
);
```

Parameters

inWindow
The window whose availability is to be obtained.

outAvailability
On exit, the current setting of the window's availability bits. For details, see "Window Availability Constants" (page 242).

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

HIWindowGetAvailablePositioningBounds

Gets the available window positioning bounds on a display.

```
OSStatus HIWindowGetAvailablePositioningBounds (
   CGDirectDisplayID inDisplay,
   HICoordinateSpace inSpace,
   HIRect *outAvailableRect
);
```

Parameters

inDisplay
The display for which to find the available bounds. May be `kCGNullDirectDisplay` to request the bounds of the main display.

inSpace

The coordinate space in which the positioning bounds should be returned. This must be either `kHICoordSpaceScreenPixel` or `kHICoordSpace72DPIGlobal`.

outAvailableRect

A pointer to a rectangle provided by the caller. On output, the rectangle contains the available bounds for the specified display. This rectangle is returned in the specified coordinate space.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Discussion
This function gets the bounds of the display not including the menu bar and Dock, if located on that display.

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

Declared In
`MacWindows.h`

HIWindowGetBounds

Gets the bounds of a specified region of a window.

```
OSStatus HIWindowGetBounds (
    WindowRef inWindow,
    WindowRegionCode inRegion,
    HICoordinateSpace inSpace,
    HIRect *outBounds
);
```

Parameters

inWindow

The window to access.

inRegion

The window region. For a list of possible values, see "Window Region Constants" (page 217).

inSpace

The coordinate space in which the bounds should be returned. This parameter must be `kHICoordSpaceWindow`, `kHICoordSpaceScreenPixel`, or `kHICoordSpace72DPIGlobal`.

outBounds

A pointer to an `HIRect` structure. On output, the structure contains the origin and size of the bounding rectangle of the specified window region. If the window does not support the region, the structure is not modified.

Return Value
A result code. If the window does not support the specified window region, the result returned is `errWindowRegionCodeInvalid`.

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

See Also
HIWindowSetBounds (page 105)

Declared In
MacWindows.h

HIWindowGetCGWindowID

Returns the Quartz window ID assigned to a window.

```
CGWindowID HIWindowGetCGWindowID (
   WindowRef inWindow
);
```

Parameters

inWindow
The window to access.

Return Value
The window ID of the specified window, or zero if the window is invalid.

Discussion
This function returns the window ID assigned by the window server when a window is created. The window ID is not generally useful with any other Carbon function, but may be used with other Mac OS X functions that require a window ID, such as functions in OpenGL.

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

See Also
HIWindowFromCGWindowID (page 95)

Declared In
MacWindows.h

HIWindowGetGreatestAreaDisplay

Finds the display with the greatest area of intersection with a window region.

```
OSStatus HIWindowGetGreatestAreaDisplay (
   WindowRef inWindow,
   WindowRegionCode inRegion,
   HICoordinateSpace inSpace,
   CGDirectDisplayID *outGreatestDisplay,
   HIRect *outGreatestDisplayRect
);
```

Parameters

inWindow
The window to compare against.

inRegion

The window region to compare against. See "Window Region Constants" (page 217) for a list of possible values.

inSpace

The coordinate space in which the display bounds should be returned. This must be either `kHICoordSpaceScreenPixel` or `kHICoordSpace72DPIGlobal`.

outGreatestDisplay

A pointer to a display ID provided by the caller, or `NULL` if you don't need this information. On output, the display ID contains the display with the greatest intersection.

outGreatestDisplayRect

A pointer to a rectangle provided by the caller, or `NULL` if you don't need this information. On output, the rectangle contains the bounds of the display with the greatest intersection. If the display with the greatest intersection also contains the menu bar, the rectangle excludes the menu bar area. This rectangle is returned in the specified coordinate space.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Availability

Available in Mac OS X v10.5 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

HIWindowGetIdealUserState

Gets the bounds of a window's content region in its user state.

```
OSStatus HIWindowGetIdealUserState (
   WindowRef inWindow,
   HICoordinateSpace inSpace,
   HIRect *outUserState
);
```

Parameters

inWindow

The window to access.

inSpace

The coordinate space in which the user state bounds should be returned. This parameter must be `kHICoordSpaceScreenPixel` or `kHICoordSpace72DPIGlobal`.

outUserState

A pointer to a structure of type `HIRect`. On return, this rectangle contains the global coordinates of the window's content region when zoomed in. If the window has not yet been zoomed, this rectangle contains the window's current content bounds.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Discussion

This function returns information about the window's user state most recently recorded by the function `ZoomWindowIdeal` (page 168).

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

Declared In
MacWindows.h

HIWindowGetProxyFSRef

Obtains the FSRef used to determine the proxy icon for a window.

```
OSStatus HIWindowGetProxyFSRef (
   WindowRef window,
   FSRef *outRef
);
```

Parameters

inWindow
The window whose proxy FSRef is to be obtained.

outRef
On exit, the FSRef for the window's proxy icon.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Discussion
If the specified window's proxy icon has been specified using HIWindowSetProxyFSRef (page 106) or SetWindowProxyAlias (page 148), HIWindowGetProxyFSRef returns noErr and a valid FSRef for the window's proxy icon. If the window has no proxy icon, or if the icon was specified by calling SetWindowProxyCreatorAndType or SetWindowProxyIcon, this function returns an error.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
MacWindows.h

HIWindowGetScaleMode

Obtains the window's scale mode and the application's display scale factor.

```
OSStatus HIWindowGetScaleMode (
   WindowRef inWindow,
   HIWindowScaleMode *outMode,
   CGFloat *outScaleFactor
);
```

Parameters

inWindow
The window whose scale mode is to be obtained.

outMode

On exit, an `HIWindowScaleMode` indicating the window's scale mode. For details, see "Window Scale Mode Constants" (page 243).

outScaleFactor

On exit, a float indicating the display scale factor for the application. Pass `NULL` if you are not interested in acquiring the scale factor.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Discussion
The window's scale mode is based on the application's display scale factor and any resolution-independent attributes specified at window creation time. Applications and the views within the window can use the scale mode and display scale factor to draw properly the content of a window for a particular scale mode.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

HIWindowGetThemeBackground

Gets the theme background brush for a window.

```
OSStatus HIWindowGetThemeBackground (
    WindowRef inWindow,
    ThemeBrush *outThemeBrush
);
```

Parameters

inWindow

The window from which to get the brush.

outThemeBrush

A pointer to a theme brush. On output, the brush is the window's theme background brush.

Return Value
A result code. If no brush is found, `themeNoAppropriateBrushErr` is returned.

Discussion
This function gets the theme background brush previously set by calling the function `SetThemeWindowBackground` (page 133).

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

Declared In
`MacWindows.h`

HIWindowInvalidateShadow

Recalculates a window's shadow.

```
OSStatus HIWindowInvalidateShadow (
   WindowRef inWindow
);
```

Parameters

inWindow

The window whose shadow is to be recalculated.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Discussion

`HIWindowInvalidateShadow` is not typically used by applications. It is useful if your application has customized window frames that change shape dynamically. After you have drawn the new window shape, you should call `HIWindowInvalidateShadow` to recalculate the shadow so that it follows the new window shape.

Availability

Available in Mac OS X v10.4 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

HIWindowIsAttributeAvailable

Returns a Boolean value indicating whether a window attribute is valid for a specified window class.

```
Boolean HIWindowIsAttributeAvailable (
   WindowClass inClass,
   int inAttr
);
```

Parameters

inClass

The window class to test.

inAttr

The window attribute to test. You must specify one of the window attributes described in "`Window Attribute Identifiers`" (page 188).

Return Value

If `true`, the window class supports the specified attribute. Otherwise, `false`.

Availability

Available in Mac OS X v10.5 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

HIWindowIsDocumentModalTarget

Determines if a window is currently the target window of another document modal window, such as a sheet.

```
Boolean HIWindowIsDocumentModalTarget (
   WindowRef inWindow,
   WindowRef *outOwner
);
```

Parameters

inWindow

The window to query.

outOwner

If this function returns `true`, `inWindow` is the target of a document modal window and `outOwner` is set to the document modal window. If this function does not return `true`, `outOwner` is undefined. Pass `NULL` if you don't want the owner's window reference.

Return Value

A Boolean whose value is `true` if the window specified by `inWindow` is currently the target of a document modal window; otherwise, `false`.

Availability

Available in Mac OS X v10.3 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

HIWindowIsInStandardState

Returns a Boolean value indicating whether a window is zoomed out to its standard state.

```
Boolean HIWindowIsInStandardState (
   WindowRef inWindow,
   const HISize *inIdealSize,
    HICoordinateSpace inSpace,
   HIRect *outIdealStandardState
);
```

Parameters

inWindow

The window whose zoom state is to be determined.

inIdealSize

The ideal width and height of the window's content region, regardless of the actual screen device dimensions. If you set `inIdealSize` to `NULL`, this function examines the dimensions stored in the `stdState` field of the `WStateData` structure.

inSpace

The coordinate space in which the ideal size is expressed and in which the standard state bounds should be returned. This parameter must be either `kHICoordSpaceScreenPixel` or `kHICoordSpace72DPIGlobal`.

outIdealStandardState

A pointer to an `HIRect` variable. On return, the variable contains the bounds of the content region of the window in its standard state, based on the data supplied in the `inIdealSize` parameter. You may pass `NULL` if you do not need this information.

Return Value

If `true`, the window is currently in its standard state. If `false`, the window is currently in its user state.

Discussion

This function compares the window's current dimensions to those in the `inIdealSize` parameter to determine if the window is currently in its standard state. You can use this function to decide whether a user's click in the zoom box is a request to zoom to the user state or the standard state, as described in the function `ZoomWindowIdeal` (page 168). You can also use this function to determine the size and position of the standard state that the Window Manager would calculate for a window, given a specified ideal size; this value is returned in the `outIdealStandardState` parameter.

Availability

Available in Mac OS X v10.5 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

HIWindowReleaseCollapsedDockTileContext

Releases a Quartz graphics context for drawing a collapsed window's Dock tile.

```
OSStatus HIWindowReleaseCollapsedDockTileContext (
   WindowRef inWindow,
   CGContextRef inContext
);
```

Parameters

inWindow

The collapsed window.

inContext

The graphics context to release. On return, the context is invalid and should no longer be used.

Return Value

A result code. If the window is not collapsed, the result code is `windowWrongStateErr`. For other possible values, see "Window Manager Result Codes" (page 247).

Discussion

To ensure that your drawing appears onscreen, you should call `CGContextFlush` before calling this function to release the context. Do not call `CFRelease` or `CGContextRelease` to release the context, or you may leak system resources.

Availability

Available in Mac OS X v10.5 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

HIWindowSetBounds

Sets the bounds of a window based on either the structure or content region.

```
OSStatus HIWindowSetBounds (
    WindowRef inWindow,
    WindowRegionCode inRegion,
    HICoordinateSpace inSpace,
    HIRect *inBounds
);
```

Parameters

inWindow

The window to access.

inRegion

The window region on which to base the window's new bounds. This parameter must be either `kWindowStructureRgn` or `kWindowContentRgn`.

inSpace

The coordinate space in which the bounds are expressed. This parameter must be `kHICoordSpaceWindow`, `kHICoordSpaceScreenPixel`, or `kHICoordSpace72DPIGlobal`.

inBounds

A pointer to an `HIRect` structure that specifies the origin and size of the bounding rectangle of the specified window region. If the coordinate space is `kHICoordSpaceWindow`, then the origin of the bounds is a window-relative value. Therefore, you can use this coordinate space to resize a window without first getting its current bounds by setting the origin to (0,0), or you can offset a window from its current position by setting the origin to the offset amount and the size to the window's current size.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Availability

Available in Mac OS X v10.5 and later.

Not available to 64-bit applications.

See Also

`HIWindowGetBounds` (page 97)

Declared In

`MacWindows.h`

HIWindowSetIdealUserState

Sets the bounds of a window's content region in its user state.

```
OSStatus HIWindowSetIdealUserState (
    WindowRef inWindow,
    HICoordinateSpace inSpace,
    const HIRect *inUserState
);
```

Parameters

inWindow

The window to access.

inSpace

The coordinate space in which the user state bounds are expressed. This parameter must be either `kHICoordSpaceScreenPixel` or `kHICoordSpace72DPIGlobal`.

inUserState

The new bounds (position and size) of the window's content region in its user state.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Discussion
A window's ideal user state is used by the function `ZoomWindowIdeal` (page 168) when zooming in.

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

Declared In
`MacWindows.h`

HIWindowSetProxyFSRef

Sets the proxy icon for a window using an `FSRef` to a file system object.

```
OSStatus HIWindowSetProxyFSRef (
   WindowRef window,
   const FSRef *inRef
);
```

Parameters

inWindow

The window whose proxy icon is to be set.

inRef

The file system object the window represents.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Discussion
This function determines the window's proxy icon by asking Icon Services for the icon for the object specified by `inRef`.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
MacWindows.h

HIWindowSetToolbarView

Sets a custom toolbar view for a window.

```
OSStatus HIWindowSetToolbarView (
    WindowRef inWindow,
    HIViewRef inView
);
```

Parameters

inWindow

The window whose toolbar view to set.

inView

The custom toolbar view for the window. You may pass NULL to remove the custom view from the window. Setting a custom view will also remove any HIToolbar that is associated with the window.

After a custom toolbar view has been set, the window owns the view and will release it automatically when the window is destroyed, or when a different custom view or standard HIToolbar is set for the window.

Return Value
A result code.

Discussion
This function is provided for use by applications that cannot use the HIToolbar API. For best compatibility with future versions of Mac OS X, you should use HIToolbar if possible. However, if HIToolbar is not sufficient for your needs, you can use this function to provide a custom toolbar view that will be placed at the standard location inside the window frame.

You are responsible for defining the appearance and behavior of the view. You cannot use this function to customize the view that is associated with an HIToolbar; a window with an HIToolbar uses a standard view that cannot be customized. When using a custom toolbar view, no function that takes an HIToolbar will work with that window. For more information about custom toolbar views, see MacWindows.h.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
MacWindows.h

HIWindowShowsFocus

Returns a Boolean value indicating whether a window's content should show focus indicators such as focus rings.

```
Boolean HIWindowShowsFocus (
    WindowRef inWindow
);
```

Parameters

inWindow

The window to access.

Return Value

If `true`, a window's content should show focus indicators; otherwise `false`.

Discussion

This function returns `true` if the window is either the modeless focus or the effective focus.

Availability

Available in Mac OS X v10.5 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

HIWindowTestAttribute

Returns a Boolean value indicating whether a window has a specified attribute.

```
Boolean HIWindowTestAttribute (
   WindowRef inWindow,
   int inAttr
);
```

Parameters

inWindow

The window to test.

inAttr

The window attribute to test. You must specify one of the window attributes described in "Window Attribute Identifiers" (page 188).

Return Value

If `true`, the window has the specified attribute. Otherwise, `false`.

Availability

Available in Mac OS X v10.5 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

HIWindowTrackProxyDrag

Tracks the drag of a window proxy icon.

```
OSStatus HIWindowTrackProxyDrag (
    WindowRef inWindow,
    EventRef inEvent,
    DragRef inDrag
);
```

Parameters

inWindow

The window whose proxy icon to drag.

inEvent

The event that resulted in starting a drag. This will most commonly be `kEventControlTrack`, but any event with `kEventParamMouseLocation` and `kEventParamKeyModifiers` parameters is all that is required.

inDrag

The proxy icon drag reference. You may pass `NULL` if you want the Window Manager to create and populate the drag reference. The Window Manager will add its own drag flavors to the drag even if you pass a pre-created drag reference.

Discussion

You can use this function to manage the dragging of the proxy icon in your application's windows. If you use the standard window event handler and you do not need to customize the proxy icon drag process, you may rely on the standard handler to call this function.

If you want to allow the Window Manager to create the drag reference and populate it with drag flavors, you should pass `NULL` in the `inDrag` parameter. If you want to create the drag reference yourself and add your own drag flavors, you should call `BeginWindowProxyDrag` (page 32) to create the drag reference, add your own flavors, call `HIWindowTrackProxyDrag` to track the proxy icon drag, and then call `EndWindowProxyDrag` (page 47) to release the drag reference.

A proxy icon may only be dragged if the window represented by the proxy icon is not modified, as indicated by the `IsWindowModified` (page 116) function. This restriction exists because a proxy icon is a representation of a physical file system object, and dragging the proxy icon may result in the Finder making a copy of the file system object. If the window is modified, then it contains user data that has not yet been saved to disk; making a copy of the file system object would result in a stale copy that did not contain the user's current data.

By default, all newly created windows are considered to be dirty. The application must pass false to `SetWindowModified` (page 146) before the proxy icon will be draggable. In Mac OS X v10.3 and later, the proxy icon is also draggable in dirty windows if the proxy icon was provided using the `SetWindowProxyIcon` (page 149) or `SetWindowProxyCreatorAndType` (page 148) functions. Dragging is allowed in this case because the window does not represent an actual file system object, and therefore there is no risk of user data loss.

Availability

Available in Mac OS X v10.5 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

InvalWindowRect

Adds a rectangle to a window's update region.

```
OSStatus InvalWindowRect (
   WindowRef window,
   const Rect *bounds
);
```

Parameters

window

The window containing the rectangle you want to be updated.

bounds

Set this structure to specify, in local coordinates, a rectangle to be added to the window's update region.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Discussion
The `InvalWindowRect` function informs the Window Manager that an area of a window should be redrawn.

See also the functions `ValidWindowRect` (page 165) and `InvalWindowRgn` (page 110).

Special Considerations

This function should not be used on composited windows. Modifying a composited window's update region does not affect the area of the window to be drawn. A composited window does not use its window update region to control drawing. Instead, a composited window determines what to draw by looking at the invalid regions of the views contained in the window.

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

Related Sample Code
CarbonSketch
HID Explorer

Declared In
`MacWindows.h`

InvalWindowRgn

Adds a region to a window's update region.

```
OSStatus InvalWindowRgn (
   WindowRef window,
   RgnHandle region
);
```

Parameters

window

The window containing the region that you want to update.

region

Set this region to specify, in local coordinates, the area to be added to the window's update region.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Discussion
The `InvalWindowRgn` function informs the Window Manager that an area of a window should be redrawn.

See also the functions `InvalWindowRect` (page 109) and `ValidWindowRgn` (page 166).

Special Considerations

This function should not be used on composited windows. Modifying a composited window's update region does not affect the area of the window to be drawn. A composited window does not use its window update region to control drawing. Instead, a composited window determines what to draw by looking at the invalid regions of the views contained in the window.

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

Declared In
`MacWindows.h`

IsValidWindowClass

Determines whether a given window class is valid.

```
Boolean IsValidWindowClass (
   WindowClass inClass
);
```

Parameters

`inClass`
The window class to query.

Return Value
A Boolean whose value is `true` if the window class is valid; otherwise, `false`.

Availability
Available in Mac OS X v10.1 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

IsValidWindowPtr

Reports whether a pointer is a valid window pointer.

```
Boolean IsValidWindowPtr (
   WindowRef possibleWindow
);
```

Parameters

`possibleWindow`
The window to query.

Return Value
A Boolean whose value is `true` if the specified pointer is a valid window pointer; otherwise, `false`.

Discussion
This function is primarily intended for use with debugging your application.

Special Considerations

The `IsValidWindowPtr` function is a processor-intensive call.

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

Declared In
`MacWindows.h`

IsWindowActive

Indicates whether the specified window is active.

```
Boolean IsWindowActive (
   WindowRef inWindow
);
```

Parameters

inWindow
The window to query.

Return Value
Returns `true` if the window is active, `false` otherwise.

Discussion
Whether a window is considered active is determined by its activation scope, highlighting, and z-order. For windows that have an activation scope of `kWindowActivationScopeAll`, a window is active if it is the window returned by the `ActiveNonFloatingWindow` (page 31) function or if it is in the same window group as the window returned by `ActiveNonFloatingWindow` and the window group has the `kWindowGroupAttrSharedActivation` attribute. For windows that have some other activation scope, the window is active if its window frame is highlighted and the window is the frontmost window in its window group.

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

Declared In
`MacWindows.h`

IsWindowCollapsable

Determines whether a window can be collapsed.

```
Boolean IsWindowCollapsable (
   WindowRef window
);
```

Parameters

window

The window to be examined.

Return Value

If `true`, the window can be collapsed; otherwise, `false`.

Discussion

You can call the `IsWindowCollapsable` function to determine if a given window can be collapsed by the `CollapseWindow` (page 37) function. In Mac OS X, the presence or absence of the `kWindowCollapseBoxAttribute` is the primary way of determining whether a window can be collapsed. If that attribute is not present, the Window Manager checks for the `kWindowCanCollapse` feature bit.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

IsWindowCollapsed

Determines whether a window is currently collapsed.

```
Boolean IsWindowCollapsed (
   WindowRef window
);
```

Parameters

window

The window to be examined.

Return Value

If `true`, the window is collapsed. If `false`, the window is expanded.

Discussion

On Mac OS 9, only window definition functions that return the feature bit `kWindowCanCollapse` in response to a `kWindowGetFeatures` message support this function; for more information, see `GetWindowFeatures` (page 64). Typically, a window's content region is empty in a collapsed state. In Mac OS X, the presence or absence of the `kWindowCollapseBoxAttribute` attribute determines whether a window can be collapsed.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

IsWindowContainedInGroup

Determines if a window is a member of a window group or any of its subgroups.

```
Boolean IsWindowContainedInGroup (
   WindowRef inWindow,
   WindowGroupRef inGroup
);
```

Parameters

inWindow

The window to query.

inGroup

The window group to query.

Return Value

A Boolean whose value is `true` if `inWindow` is a member of the window group specified by `inGroup`, or if `inWindow` is a member of a window group that is a member of the window group specified by `inGroup`. Otherwise, this function returns `false`.

Discussion

This function returns `true` if group A contains window A. It also returns `true` if group A contains group B and group B contains window A.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

IsWindowHilited

Indicates whether the window frame is currently highlighted.

```
Boolean IsWindowHilited (
   WindowRef window
);
```

Parameters

window

The window to query.

Return Value

A Boolean value indicating whether or not the window frame is highlighted. If `true`, the window is visible. If `false`, the window frame is not highlighted.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

IsWindowInStandardState

Determines whether a window is currently zoomed in to the user state or zoomed out to the standard state.

```
Boolean IsWindowInStandardState (
    WindowRef inWindow,
    const Point *inIdealSize,
    Rect *outIdealStandardState
);
```

Parameters

inWindow

The window whose zoom state is to be determined.

inIdealSize

Set the `Point` structure to contain the ideal width and height of the window's content region, regardless of the actual screen device dimensions. If you set `inIdealSize` to `NULL`, `IsWindowInStandardState` examines the dimensions stored in the `stdState` field of the `WStateData` structure.

outIdealStandardState

On input, a pointer to a structure of type `Rect`. On return, the rectangle contains the global coordinates for the content region of the window in its standard state, based on the data supplied in the `inIdealSize` parameter. You may pass `NULL` if you do not want to receive this data.

Return Value

A Boolean whose value is `true` if the window is currently in its standard state; `false` if the window is currently in the user state.

Discussion

The `IsWindowInStandardState` function compares the window's current dimensions to those referred to by the `inIdealSize` parameter to determine if the window is currently in the standard state. Your application may use `IsWindowInStandardState` to decide whether a user's click of the zoom box is a request to zoom to the user state or the standard state, as described in the function `ZoomWindowIdeal` (page 168). Your application may also use `IsWindowInStandardState` to determine the size and position of the standard state that the Window Manager would calculate for a window, given a specified ideal size; this value is produced in the `outIdealStandardState` parameter.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

See Also

`HIWindowIsInStandardState` (page 103)

Declared In

`MacWindows.h`

IsWindowLatentVisible

Indicates whether a window is visible onscreen or is latently visible but not currently onscreen.

```
Boolean IsWindowLatentVisible (
   WindowRef inWindow,
   WindowLatentVisibility *outLatentVisible
);
```

Parameters

inWindow

The window to query.

outLatentVisible

If the window is onscreen, the latent visibility is zero. If the window is offscreen, this parameter returns the latent visibility flags of the window. If any of the flags are set, the window is latently visible.

Return Value

A Boolean whose value is `true` if the window is currently onscreen; otherwise, `false`.

Discussion

All windows are either onscreen or offscreen. A window that is offscreen may still be latently visible. This occurs, for example, when a floating window is hidden as an application is suspended. The floating window is not visible onscreen, but it is latently visible and is only hidden due to the suspended state of the application. When the application becomes active again, the floating window will be placed back onscreen.

Availability

Available in Mac OS X v10.1 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

IsWindowModified

Obtains the modification state of the specified window.

```
Boolean IsWindowModified (
   WindowRef window
);
```

Parameters

window

The window whose modification state is to be obtained.

Return Value

`true` if the content of the window has been modified; otherwise, `false`. Newly created windows start out with their modification state automatically set to `true`.

Discussion

Your application can use the functions `IsWindowModified` and `SetWindowModified` (page 146) instead of maintaining its own separate record of the modification state of the content of a window.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

QTCarbonShell

Declared In
MacWindows.h

IsWindowPathSelectEvent

Determines whether a Carbon event describing a click on a window's title should cause a path selection menu to be displayed.

```
Boolean IsWindowPathSelectEvent (
   WindowRef window,
   EventRef inEvent
);
```

Parameters

window

The window to query.

inEvent

The event. In CarbonLib and in Mac OS X v10.2 and earlier, the function only returns `true` for `kEventClassMouse`/`kEventMouseDown` events. In Mac OS X v10.3 and later, this function returns `true` for any event that has suitable `kEventParamMouseLocation` and `kEventParamModifiers` parameters.

Return Value
A Boolean whose value is `true` if the click should cause a path selection menu to be displayed; otherwise, `false`. If this function returns `true`, the application should call `WindowPathSelect` (page 166).

Discussion
Windows that have a proxy icon provided using an `FSRef` or alias can support a path selection menu, which displays the file system path to the object, one menu item per directory. Making a selection from this item automatically opens the corresponding object in the Finder.

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

Declared In
MacWindows.h

IsWindowToolbarVisible

Determines whether a window's toolbar is visible.

```
Boolean IsWindowToolbarVisible (
   WindowRef inWindow
);
```

Parameters

inWindow

The window to query.

Return Value
A Boolean whose value is `true` if the window's toolbar is visible; otherwise, `false`.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

IsWindowUpdatePending

Determines whether a window update is pending.

```
Boolean IsWindowUpdatePending (
   WindowRef window
);
```

Parameters

window
The non-composited window to query.

Return Value
A Boolean whose value is `true` if an update is pending; otherwise, `false`.

Special Considerations

Modifying a composited window's update region does not affect the area of the window to be drawn. A composited window does not use its window update region to control drawing. Instead, a composited window determines what to draw by looking at the invalid regions of the views contained in the window.

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

Declared In
`MacWindows.h`

IsWindowVisible

Indicates whether the window frame is currently visible.

```
Boolean IsWindowVisible (
    WindowRef window
);
```

Parameters

window
The window to query.

Return Value
A Boolean value indicating whether or not the window is visible. If `true`, the window is visible. If `false`, the window is invisible.

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

Declared In
MacWindows.h

MoveWindow

Moves a window on the desktop.

```
void MoveWindow (
    WindowRef window,
    short hGlobal,
    short vGlobal,
    Boolean front
);
```

Parameters

window
The window that is to be moved on the desktop.

hGlobal
On input, the new location, in global coordinates, of the left edge of the window's port rectangle.

vGlobal
On input, the new location, in global coordinates, of the top edge of the window's port rectangle.

front
On input, a Boolean value specifying whether the window is to become the frontmost, active window. If the value of the front parameter is `false`, `MoveWindow` does not change its plane or status. If the value of the front parameter is `true` and the window isn't active, `MoveWindow` makes it active by calling the `SelectWindow` (page 129) function.

Discussion
The `MoveWindow` function moves the specified window to the location specified by the `hGlobal` and `vGlobal` parameters, without changing the window's size. The upper-left corner of the window's port rectangle is placed at the point (`vGlobal`, `hGlobal`). The local coordinates of the upper-left corner are unaffected.

Your application doesn't normally call `MoveWindow`. When the user drags a window by dragging its title bar, you can call `DragWindow` (page 46) which in turn calls `MoveWindow` when the user releases the mouse button.

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

Related Sample Code
HID Calibrator

Declared In
MacWindows.h

MoveWindowStructure

Positions a window relative to its structure region.

```
OSStatus MoveWindowStructure (
   WindowRef window,
   short hGlobal,
   short vGlobal
);
```

Parameters

window

The window that is to be moved.

hGlobal

A value specifying the horizontal position, in global coordinates, to which the left edge of the window's structure region is to be moved.

vGlobal

A value specifying the vertical position, in global coordinates, to which the top edge of the window's structure region is to be moved.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Discussion

The `MoveWindowStructure` function moves the specified window, but does not change the window's size. When your application calls `MoveWindowStructure`, the positioning of the specified window is determined by the positioning of its structure region. This is in contrast to the `MoveWindow` function, where the positioning of the window's content region determines the positioning of the window. After moving the window, `MoveWindowStructure` displays the window in its new position.

Note that your application should not call the `MoveWindowStructure` function to position a window when the user drags the window by its drag region. When the user drags the window, your application should call the pre–Mac OS 8.5 Window Manager function `DragWindow`.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

QTCarbonShell

Declared In

`MacWindows.h`

OpenDrawer

Opens a drawer.

```
OSStatus OpenDrawer (
   WindowRef inDrawerWindow,
   OptionBits inEdge,
   Boolean inAsync
);
```

Parameters

inDrawerWindow

The drawer window to open.

inEdge

The parent window edge on which to open the drawer. Pass `kWindowEdgeDefault` to use the drawer's preferred edge. If there is not enough room on the preferred edge, `OpenDrawer` tries the opposite edge. If there is insufficient room on both edges, the drawer will open on the preferred edge but may extend offscreen, under the Dock, or under the menu bar.

inAsync

Indicates whether to open the drawer synchronously (the drawer is entirely opened before the function call returns) or asynchronously (the drawer opens using an event loop timer after the function call returns). Specify `true` for asynchronous and `false` for synchronous.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Availability
Available in Mac OS X v10.2 and later.

Not available to 64-bit applications.

Declared In
`MacWindows.h`

PinRect

Returns the point within the specified rectangle that is closest to the specified point.

```
long PinRect (
   const Rect *theRect,
   Point thePt
);
```

Parameters

theRect

On input, a pointer to a rectangle in which the point is to be contained.

thePt

On input, a pointer to the point to be contained.

Return Value
A long integer that specifies a point within the specified rectangle that is as close as possible to the specified point. (The high-order word of the returned long integer is the vertical coordinate; the low-order word is the horizontal coordinate.

Discussion
`DragGrayRgn` uses the `PinRect` function to contain a point within a specified rectangle. If the specified point is within the rectangle, `PinRect` returns the point itself. If not, then

- if the horizontal position is to the left of the rectangle, `PinRect` returns the left edge as the horizontal coordinate
- if the horizontal position is to the right of the rectangle, `PinRect` returns the right edge minus 1 as the horizontal coordinate
- if the vertical position is above the rectangle, `PinRect` returns the top edge as the vertical coordinate
- if the vertical position is below the rectangle, `PinRect` returns the bottom edge minus 1 as the vertical coordinate

The 1 is subtracted when the point is below or to the right of the rectangle so that a pixel drawn at that point lies within the rectangle.

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

Declared In
`MacWindows.h`

RegisterWindowDefinition

Registers a binding between a resource ID and a window definition function.

```
OSStatus RegisterWindowDefinition (
   SInt16 inResID,
   const WindowDefSpec *inDefSpec
);
```

Parameters

`inResID`
A WDEF proc ID, as used in a `'WIND'` resource.

`inDefSpec`
Specifies the `WindowDefUPP` that should be used for windows with the given WDEF proc ID. Pass `NULL` to unregister a given WDEF proc ID.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Discussion
In the Mac OS 8.x Window Manager, a `'WIND'` resource can contain an embedded WDEF procID that is used by the Window Manager as the resource ID of a `'WDEF'` resource to lay out and draw the window. The `'WDEF'` resource is loaded by the Window Manager when you load the window with `GetNewWindow`. Since WDEFs can no longer be packaged as code resources on Carbon, the procID can no longer refer directly to a WDEF resource. However, using `RegisterWindowDefinition` you can instead specify a UniversalProcPtr pointing to code in your application code fragment.

To unregister a window definition, pass `NULL` in the `inDefSpec` parameter for a given WDEF proc ID.

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

Declared In
`MacWindows.h`

ReleaseWindowGroup

Decrements the reference count for a window group.

```
OSStatus ReleaseWindowGroup (
   WindowGroupRef inGroup
);
```

Parameters

inGroup

The window group whose reference count is to be queried.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

MacWindows.h

RemoveWindowProperty

Removes a piece of data that is associated with a window.

```
OSStatus RemoveWindowProperty (
   WindowRef window,
   PropertyCreator propertyCreator,
   PropertyTag propertyTag
);
```

Parameters

window

The window whose data is to be removed.

propertyCreator

The creator code (typically, the application's signature) of the associated data to be removed.

propertyTag

The application-defined code identifying the associated data to be removed.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Discussion

The data removed by the `RemoveWindowProperty` function must have been previously associated with the window with the function `SetWindowProperty` (page 147).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

MacWindows.h

RemoveWindowProxy

Dissociates a file from a window.

```
OSStatus RemoveWindowProxy (
   WindowRef window
);
```

Parameters

window

The window for which you want to remove the associated file.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Discussion

The `RemoveWindowProxy` function redraws the window title bar after removing all data associated with a given file, including the proxy icon, path menu, and file data.

Special Considerations

On Mac OS 8.x and Mac OS 9.x, you must save and restore the current graphics port—by calling the QuickDraw functions `GetPort` and `SetPort`—around each call to the `RemoveWindowProxy` function.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

QTCarbonShell

Declared In

`MacWindows.h`

RepositionWindow

Positions a window relative to another window or a display screen.

```
OSStatus RepositionWindow (
   WindowRef window,
   WindowRef parentWindow,
   WindowPositionMethod method
);
```

Parameters

window

The window whose position you want to set.

parentWindow

A pointer to the "parent" window, as defined by your application. In cases where the window positioning method does not require a parent window, you should set the `parentWindow` parameter to `NULL`.

method

A constant specifying the window positioning method to be used; see "Window Position Constants" (page 213) for descriptions of possible values.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Discussion
Your application may call the `RepositionWindow` function to position any window, relative to another window or to a display screen. After positioning the window, `RepositionWindow` displays the window in its new position.

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

Related Sample Code
BSDLLCTest

Declared In
`MacWindows.h`

ReshapeCustomWindow

Notifies the Window Manager that a custom window's shape has changed.

```
OSStatus ReshapeCustomWindow (
   WindowRef window
);
```

Parameters

window
The window whose shape has changed.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Discussion
If the shape of a custom window needs to change dynamically, outside of the context of normal Window Manager operations, you must use `ReshapeCustomWindow` to notify the Window Manager so that it can recalculate the window regions and update the screen. The Window Manager queries your custom window definition for the new structure and content regions and updates the screen with the new window shape.

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

Declared In
`MacWindows.h`

ResizeWindow

Handles all user interaction while a window is being resized.

```
Boolean ResizeWindow (
   WindowRef inWindow,
   Point inStartPoint,
   const Rect *inSizeConstraints,
   Rect *outNewContentRect
);
```

Parameters

window

The window that is to be resized.

inStartPoint

Set the `Point` structure to contain the location, specified in global coordinates, where the mouse-down event occurred. Your application may retrieve this value from the `where` field of the `EventRecord` structure.

inSizeConstraints

Set the rectangle to specify the limits on the vertical and horizontal measurements of the content rectangle, in pixels. Although this parameter gives the address of a structure that is in the form of the `Rect` data type, the four numbers in the structure represent limits, not screen coordinates. The `top`, `left`, `bottom`, and `right` fields of the structure specify the minimum vertical measurement (`top`), the minimum horizontal measurement (`left`), the maximum vertical measurement (`bottom`), and the maximum horizontal measurement (`right`). The minimum dimensions should be large enough to allow a manageable rectangle; 64 pixels on a side is typical. The maximum dimensions can be no greater than 32,767. You can pass `NULL` to allow the user to resize the window to any size that is contained onscreen.

outNewContentRect

On input, a pointer to a structure of type `Rect`. On return, the structure contains the new dimensions of the window's content region, in global coordinates.

Return Value

`true` if the window was successfully resized; otherwise, `false`.

Discussion

The `ResizeWindow` function moves either an outline of the window's edges (Mac OS 9.x and earlier) or the actual window (Mac OS X) around the screen, following the user's cursor movements, and handles all user interaction until the mouse button is released. Unlike with the function `GrowWindow`, there is no need to follow this call with a call to the function `SizeWindow`, because once the mouse button is released, `ResizeWindow` resizes the window if the user has changed the window size. Once the resizing is complete, `ResizeWindow` draws the window in the new size.

Your application should call `ResizeWindow` instead of the earlier Window Manager functions `SizeWindow` and `GrowWindow`. The `ResizeWindow` function informs your application of the new window bounds, so that your application can respond to any changes in the window's position.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

RetainWindowGroup

Increments the reference count for a window group.

```
OSStatus RetainWindowGroup (
   WindowGroupRef inGroup
);
```

Parameters

inGroup

The window group whose reference count is to be incremented. For information on this data type, see `WindowGroupRef` (page 182).

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

ScrollWindowRect

Scroll any area of a window.

```
OSStatus ScrollWindowRect (
   WindowRef inWindow,
   const Rect *inScrollRect,
   SInt16 inHPixels,
   SInt16 inVPixels,
   ScrollWindowOptions inOptions,
   RgnHandle outExposedRgn
);
```

Parameters

inWindow

The window to scroll in.

inScrollRect

The rectangle to scroll, in local coordinates.

inHPixels

The number of pixels to scroll horizontally.

inVPixels

The number of pixels to scroll vertically.

inOptions

Options for the scroll. See "Window Scrolling Options" (page 240) for a list of possible options.

outExposedRgn

A valid region handle for the area newly revealed by the scroll (can be `NULL`). If `NULL`, the exposed region is added to the window's update region, regardless of the state of the `kScrollWindowInvalidate` option. This prevents updates from being lost in multiple monitor situations where the Window Manager can't copy the entire region due to differing color tables.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Discussion
Scrolls pixels that are inside the specified region of the input window. No other pixels or the bits they represent are affected. The pixels are shifted a distance of `inHPixels` horizontally and `inVPixels` vertically. The positive directions are to the right and down. The pixels that are shifted out of the specified window are not displayed, and the bits they represent are not saved. The exposed empty area created by the scrolling is returned in the update region parameter and optionally added to the window's update region.

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

Declared In
`MacWindows.h`

ScrollWindowRegion

Scrolls a window's region.

```
OSStatus ScrollWindowRegion (
   WindowRef inWindow,
   RgnHandle inScrollRgn,
   SInt16 inHPixels,
   SInt16 inVPixels,
   ScrollWindowOptions inOptions,
   RgnHandle outExposedRgn
);
```

Parameters

inWindow
The window to scroll in.

inScrollRgn
The region to scroll, in local coordinates.

inHPixels
The number of pixels to scroll horizontally.

inVPixels
The number of pixels to scroll vertically.

inOptions
Options for the scroll. See "Window Scrolling Options" (page 240) for a list of possible options.

outExposedRgn
A valid region handle for the area newly revealed by the scroll (can be `NULL`). If `NULL`, the exposed region is added to the window's update region, regardless of the state of the `kScrollWindowInvalidate` option. This prevents updates from being lost in multiple monitor situations where the Window Manager can't copy the entire region due to differing color tables.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Discussion
Scrolls pixels that are inside the specified region of the input window. No other pixels or the bits they represent are affected. The pixels are shifted a distance of `inHPixels` horizontally and `inVPixels` vertically. The positive directions are to the right and down. The pixels that are shifted out of the specified window are not displayed, and the bits they represent are not saved. The exposed empty area created by the scrolling is returned in the update region parameter and optionally added to the window's update region

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

Declared In
`MacWindows.h`

SelectWindow

Makes a window active.

```
void SelectWindow (
   WindowRef window
);
```

Parameters
window
The window that is to be made active.

Discussion
The `SelectWindow` function removes highlighting from the previously active window, brings the specified window to the front, highlights it, and generates the activate events to deactivate the previously active window and activate the specified window. If the specified window is already active, `SelectWindow` has no effect. Call `SelectWindow` when the user presses the mouse button while the cursor is in the content region of an inactive window.

Even if the specified window is invisible, `SelectWindow` brings the window to the front, activates the window, and deactivates the previously active window. Note that in this case, no active window is visible on the screen. If you do select an invisible window, be sure to call `ShowWindow` (page 156) immediately to make the window visible (and accessible to the user).

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

Related Sample Code
HideMenuBar
QTCarbonShell

Declared In
`MacWindows.h`

SendBehind

Moves one window behind another.

```
void SendBehind (
   WindowRef window,
   WindowRef behindWindow
);
```

Parameters

window

The window to be moved.

behindWindow

On input, a pointer to the window that is to be in front of the moved window.

Discussion

The `SendBehind` function moves the window pointed to by the parameter `window` behind the window pointed to by the parameter `behindWindow`. If the move exposes previously obscured windows or parts of windows, `SendBehind` redraws the frames as necessary and generates the appropriate update events to have any newly exposed content areas redrawn.

If the value of `behindWindow` is `NULL`, `SendBehind` sends the window to be moved behind all other windows on the desktop. If the window to be moved is the active window, `SendBehind` removes its highlighting, highlights the newly exposed frontmost window, and generates the appropriate activate events.

Do not use `SendBehind` to deactivate a window after you've made a new window active with the function `SelectWindow` (page 129). The `SelectWindow` function automatically deactivates the previously active window.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

SendWindowGroupBehind

Orders one window group behind another.

```
OSStatus SendWindowGroupBehind (
   WindowGroupRef inGroup,
   WindowGroupRef behindGroup
);
```

Parameters

inGroup

The window group.

behindGroup

The "behind" window group.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Discussion

A window group can contain multiple window groups. You can use this function to order nested groups.

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

Declared In
`MacWindows.h`

SetDrawerOffsets

Sets the positioning offsets for the drawer with respect to its parent window.

```
OSStatus SetDrawerOffsets (
   WindowRef inDrawerWindow,
   CGFloat inLeadingOffset,
   CGFloat inTrailingOffset
);
```

Parameters

inDrawerWindow
The drawer window whose positioning offsets are to be set.

inLeadingOffset
The new leading offset, in pixels. Pass `kWindowOffsetUnchanged` if you don't want to change the leading offset.

inTrailingOffset
The new trailing offset, in pixels. Pass `kWindowOffsetUnchanged` if you don't want to change the trailing offset.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

SetDrawerParent

Sets the parent window for a drawer.

```
OSStatus SetDrawerParent (
   WindowRef inDrawerWindow,
   WindowRef inParent
);
```

Parameters

inDrawerWindow
The drawer window whose parent window is to be set.

inParent
The window that is to be set as the parent of the window specified by `inDrawerWindow`.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

SetDrawerPreferredEdge

Set the preferred window edge from which the drawer should appear.

```
OSStatus SetDrawerPreferredEdge (
   WindowRef inDrawerWindow,
   OptionBits inEdge
);
```

Parameters

inDrawerWindow
The drawer window whose preferred window edge is to be set.

inEdge
The preferred edge. See "Window Edge Constants" (page 237) for a list of possible values.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

SetPortWindowPort

Sets the current graphics port to the window's port.

```
void SetPortWindowPort (
   WindowRef window
);
```

Parameters

window
The window whose graphics port is to be set.

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

Related Sample Code
HID Explorer

QTCarbonShell

Declared In
MacWindows.h

SetThemeTextColorForWindow

Sets a text color that contrasts with a theme brush.

```
OSStatus SetThemeTextColorForWindow (
   WindowRef inWindow,
   Boolean inActive,
   SInt16 inDepth,
   Boolean inColorDev
);
```

Parameters

inWindow
The window whose text color is to be set.

inActive
A Boolean whose value is `true` to indicate an active state or `false` to indicate an inactive state.

inDepth
The bit depth of the window's port. In Mac OS X, this parameter is ignored and should always be set to `32`.

inColorDev
A Boolean whose value is `true` to indicate that the window's port is color or `false` to indicate that the port is black and white. In Mac OS X, this parameter is ignored and should always be set to `true`.

Return Value
A result code. See "Window Manager Result Codes" (page 247) for a list of possible values.

Discussion
`SetThemeTextColorForWindow` sets a text color in the specified window's port that contrasts with the brush specified by `SetThemeWindowBackground` (page 133) and also matches the `inActive` parameter.

Only a subset of the theme brushes have theme text colors. As of Mac OS 9 and Mac OS X v10.4 and later, the Alert, Dialog, Modeless Dialog, and Notification brushes have corresponding text colors. For any other brush, `SetThemeTextColorForWindow` returns `themeNoAppropriateBrushErr` and does not modify the text color.

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

Declared In
MacWindows.h

SetThemeWindowBackground

Sets a window's background theme.

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

Parameters

inWindow

The window whose background theme is to be set.

inBrush

The theme brush that determines how the window background is painted. For information on theme brushes, see the Appearance Manager documentation.

inUpdate

A Boolean whose value is `true` if you want the window to be redrawn immediately using the new background brush; otherwise, `false`.

Return Value
A result code. See "Window Manager Result Codes" (page 247) for a list of possible values.

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

Not available to 64-bit applications.

Declared In
`MacWindows.h`

SetUserFocusWindow

Designates a window to receive user focus.

```
OSStatus SetUserFocusWindow (
    WindowRef inWindow
);
```

Return Value
A result code.

Discussion
You can use this function to assign user focus to a specified window. This tells the Carbon Event Manager to route events that should go to the user focus (for example, commands and keyboard events) to the specified window. This can be used, for example, to route keyboard events to a floating palette, since floating palettes do not normally receive user focus.

Setting focus automatically defocuses whatever element formerly had user focus. If the focus changes to a new window, the `kEventWindowFocusAcquired` Carbon event will be sent to the newly focused window, and the `kEventWindowFocusRelinquish` Carbon event will be sent to the previously focused window.

If you pass `kUserFocusAuto` in the `inWindow` parameter, the system picks the best candidate for user focus (typically, this will be the active window). If you temporarily change the focus to a special window, you should use this option to restore the focus rather than setting the focus to an explicit window.

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

Not available to 64-bit applications.

Declared In
MacWindows.h

SetWindowActivationScope

Sets a window's activation scope.

```
OSStatus SetWindowActivationScope (
   WindowRef inWindow,
   WindowActivationScope inScope
);
```

Parameters

inWindow
The window whose activation scope is to be set.

inScope
The new activation scope.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

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

Declared In
MacWindows.h

SetWindowAlpha

Sets the window's alpha channel value.

```
OSStatus SetWindowAlpha (
   WindowRef inWindow,
   CGFloat inAlpha
);
```

Parameters

inWindow
The window whose alpha channel value is to be set.

inAlpha
The alpha value to set. This value can range from 0.0 (completely transparent) to 1.0 (opaque).

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
MacWindows.h

SetWindowAlternateTitle

Sets an alternate window title.

```
OSStatus SetWindowAlternateTitle (
   WindowRef inWindow,
   CFStringRef inTitle
);
```

Parameters

inWindow

The window for which to set the alternate title.

inTitle

The alternate title for the window. Passing `NULL` for this parameter will remove any alternate title that might be present.

Return Value

A result code. See "Window Manager Result Codes" (page 247). An operating system status code.

Discussion

This API sets an alternate title for a window. The alternate title overrides what is displayed in the Window menu. If you do not set an alternate title, the normal window title is used. You would normally use this if the window title was not expressive enough to be used in the Window menu (or similar text-only situation).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

SetWindowBounds

Sets a window's size and position from the bounding rectangle of the specified window region.

```
OSStatus SetWindowBounds (
   WindowRef window,
   WindowRegionCode regionCode,
   const Rect *globalBounds
);
```

Parameters

window

The window whose bounds are to be set.

regionCode

A constant specifying the region to be used in determining the window's size and position. The only region codes allowed for this parameter are `kWindowStructureRgn` and `kWindowContentRgn`.

globalBounds

Set the rectangle to specify the dimensions and position, in global coordinates, of the window region specified in the `regionCode` parameter.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Discussion
The `SetWindowBounds` function sets a window's size and position to that specified by the rectangle that your application passes in the `globalBounds` parameter. After doing so, `SetWindowBounds` redraws the window, if the window is visible.

When you call the `SetWindowBounds` function, your application specifies whether the window's content region or its structure region is more important in determining the window's ultimate size and position. This distinction can be important with versions of the Mac OS running the Appearance Manager, since the total dimensions of a window—and, therefore, its spatial relationship to the rest of the screen—may vary from appearance to appearance. In general, you should specify `kWindowStructureRgn` for the `regionCode` parameter if how the window as a whole relates to a given monitor is more important than the exact positioning of its content on the screen. On the other hand, if you specify `kWindowContentRgn` for the `regionCode` parameter because the positioning of your application's content is of greatest concern, then it is important to note that with some appearances some part of the window's structure region or "frame" may extend past the edge of a monitor and not be displayed.

See also the function `GetWindowBounds` (page 60).

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

See Also
`HIWindowSetBounds` (page 105)

Declared In
`MacWindows.h`

SetWindowCancelButton

Specifies a Cancel button for a window.

```
OSStatus SetWindowCancelButton (
    WindowRef inWindow,
    ControlRef inControl
);
```

Parameters

inWindow
The window whose Cancel button you want to set.

inControl
The control to designate as the Cancel button.

Return Value
A result code.

Discussion
You can use this function to specify a control (normally a button) to be the Cancel button for a given window. This button would be considered to have been clicked if the user instead presses Command-period or the Escape key.

The standard window event handler looks for keystrokes that correspond to the cancel button and generates events of type `kEventControlHit` when it detects the correct key being pressed. This is similar to the way the Dialog Manager responds to cancel buttons, except that instead of returning an item index for which button is pressed, the Carbon Event Manager generates a control hit event.

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

Declared In
`MacWindows.h`

SetWindowContentColor

Sets the color to which a window's content region is redrawn.

```
OSStatus SetWindowContentColor (
   WindowRef window,
   const RGBColor *color
);
```

Parameters

window
The window whose content color is to be set.

color
Set this structure to specify the content color to be used.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Discussion
If your application uses the `SetWindowContentColor` function, the window's content region is redrawn to the color you specify, without affecting the value specified in the window's `CGrafPort` structure for the current background color.

See also the function `GetWindowContentColor` (page 62).

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

Declared In
`MacWindows.h`

SetWindowContentPattern

Sets the pattern to which a window's content region is redrawn.

```
OSStatus SetWindowContentPattern (
   WindowRef window,
   PixPatHandle pixPat
);
```

Parameters

window

A pointer to the window whose content pattern is being set.

pixPat

Set this structure to specify the content pattern to be used. This handle is copied by the Window Manager, and your application continues to own the original. Therefore there may be higher RAM requirements for applications with numerous identically patterned windows.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Discussion
If your application uses the `SetWindowContentPattern` function, the window's content region is redrawn to the pattern you specify, without affecting the value specified in the window's `CGrafPort` structure for the current background pattern.

See also the function `GetWindowContentPattern` (page 63).

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

Declared In
`MacWindows.h`

SetWindowDefaultButton

Specifies a default button for a window.

```
OSStatus SetWindowDefaultButton (
    WindowRef inWindow,
    ControlRef inControl
);
```

Parameters

inWindow

The window whose default button you want to set.

inControl

The control to designate as the default.

Return Value
A result code.

Discussion
You can use this function to specify a control (normally a button) to be the default for a given window. This button would be considered to have been clicked if the user instead presses the Return or Enter keys on the keyboard.

The standard window event handler looks for keystrokes that correspond to the default button and generates events of type `kEventControlHit` when it detects the correct key being pressed. This is similar to the way the Dialog Manager responds to default buttons, except that instead of returning an item index for which button is pressed, the Carbon Event Manager generates a control hit event.

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

Related Sample Code
BSDLLCTest
CarbonCocoa_PictureCursor

Declared In
`MacWindows.h`

SetWindowDockTileMenu

Associates a pop-up menu with a window.

```
OSStatus SetWindowDockTileMenu (
   WindowRef inWindow,
   MenuRef inMenu
);
```

Parameters

`inWindow`
The window with which a pop-up menu is to be associated.

`inMenu`
The pop-up menu that is to be associated with the window specified by `inWindow`

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Discussion
You specify a dock tile menu if you want to be able to present special selections when the user activates the pop-up menu associated with the window's minimized dock tile.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Related Sample Code
QTCarbonShell

Declared In
`MacWindows.h`

SetWindowGroup

Assigns a window to a window group.

```
OSStatus SetWindowGroup (
   WindowRef inWindow,
   WindowGroupRef inNewGroup
);
```

Parameters

inWindow

The window that is to be assigned to a window group.

inNewGroup

The window group. For information on this data type, see `WindowGroupRef` (page 182).

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

SetWindowGroupLevel

Sets the level of group in the window class hierarchy.

```
OSStatus SetWindowGroupLevel (
   WindowGroupRef inGroup,
   SInt32 inLevel
);
```

Parameters

inGroup

The window group.

inLevel

The new level for the windows in this group.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Discussion

The window group's level is only used to set the level of its windows if the window group is a child of the root group. If there is another group in the group hierarchy between this group and the root group, this group's level is ignored.

In Mac OS X v10.4 and later, `SetWindowGroupLevel` sets all three window levels associated with a window group: active, inactive, and promoted. It then immediately determines if the active level needs to be promoted to a larger value, and if so, sets the promoted level to that value.

See the Core Graphics frameworks header `CGWindowLevel.h` for a listing of window levels.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
MacWindows.h

SetWindowGroupLevelOfType

Sets the window level of a window group.

```
OSStatus SetWindowGroupLevelOfType (
   WindowGroupRef inGroup,
   UInt32 inLevelType,
   CGWindowLevel inLevel
);
```

Parameters

inGroup
The window group whose Core Graphics window level is to be set.

inLevelType
The level type to set. Specify kWindowGroupLevelActive or kWindowGroupLevelInactive. For details, see "Window Group Level Constants" (page 244).

inLevel
The new level that is to be set for the windows in this group.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Discussion
In Mac OS X v10.4 and later, multiple window levels may be associated with a window group: one level for when the application is active and another for when the application is inactive. The Window Manager automatically switches each group's level as the application becomes active or inactive. Call SetWindowGroupLevelOfType to set the active and inactive window level for a window group. The window group's level is only used to set the level of its windows if the window group is a child of the root group. If there is another group in the group hierarchy between this group and the root group, this group's level is ignored.

You can also use SetWindowGroupLevelOfType to set the promoted window level that is actually used for windows in the group. Doing so is not recommended, however, because the promoted window level is reset by the Window Manager whenever the window group hierarchy structure changes. Any changes that you make to the promoted level may, therefore, be overwritten. In general, you should only use SetWindowGroupLevelOfType to set the active and inactive window levels. When setting the active level of a group with the fixed-level window group attribute, this function also automatically sets the promoted level to the same value and updates the promoted level of any non-fixed-level groups above the group being modified.

Availability
Available in Mac OS X v10.4 and later.
Not available to 64-bit applications.

Declared In
MacWindows.h

SetWindowGroupName

Assigns a name to a window group.

```
OSStatus SetWindowGroupName (
   WindowGroupRef inGroup,
   CFStringRef inName
);
```

Parameters

inGroup

The window group. For information on this data type, see `WindowGroupRef` (page 182).

inName

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Carbon Porting Notes

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

SetWindowGroupOwner

Sets a window as the owner of a window group.

```
OSStatus SetWindowGroupOwner (
   WindowGroupRef inGroup,
   WindowRef inWindow
);
```

Parameters

inGroup

The window group that is to be set as the owner of the window group specified by `inWindow`.

inWindow

The window group whose owner is to be set.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Discussion

This function is rarely needed and is known to be problematic, so calling this function is not recommended.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

SetWindowGroupParent

Sets a window group to be the parent of another window group.

```
OSStatus SetWindowGroupParent (
   WindowGroupRef inGroup,
   WindowGroupRef inNewGroup
);
```

Parameters

inGroup

The window group whose parent window group is to be set. The specified window group cannot contain any windows at the time of this call.

inNewGroup

The window group that is to be the parent of `inGroup`. For information on this data type, see `WindowGroupRef` (page 182).

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Discussion

You can nest groups within each other using this function.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

SetWindowIdealUserState

Sets the size and position of a window in its user state.

```
OSStatus SetWindowIdealUserState (
    WindowRef inWindow,
    const Rect *inUserState
);
```

Parameters

inWindow

The window whose size and position in its user state is to be set.

inUserState

Set this rectangle to specify the new size and position of the window's user state, in global coordinates.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Discussion

Because the window definition function relies upon the `WStateData` structure, it is unaware of the ideal standard state, and this causes the user state data that it stores in the `WStateData` structure to be unreliable. While the Window Manager is reliably aware of the window's zoom state, it cannot record the current user state in the `WStateData` structure, because the window definition function can overwrite that data. Therefore, the function `SetWindowIdealUserState` maintains the window's user state independently of the

`WStateData` structure. The `SetWindowIdealUserState` function gives your application access to the user state data maintained by `ZoomWindowIdeal` (page 168). However, your application does not typically need to use this function; it is supplied for completeness.

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

See Also
`HIWindowSetIdealUserState` (page 105)

Declared In
`MacWindows.h`

SetWindowKind

Sets a window's window kind.

```
void SetWindowKind (
   WindowRef window,
   short kind
);
```

Parameters

window
The window whose window kind is to be set.

kind
An integer representing the window kind.

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

Declared In
`MacWindows.h`

SetWindowModality

Sets the modality of a window.

```
OSStatus SetWindowModality (
   WindowRef inWindow,
   WindowModality inModalKind,
   WindowRef inUnavailableWindow
);
```

Parameters

inWindow
The window whose modality to set.

inModalKind
The new modality for the window. See "Window Modality Options" (page 212) for a list of possible options.

inUnavailableWindow

If the window is becoming document-modal, this parameter specifies the window to which the `inWindow` parameter is modal. The window specified by this parameter will not be available while `inWindow` is in window-modal state.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Discussion
The modality of a window is used by the Carbon event manager to automatically determine appropriate event handling.

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

Declared In
`MacWindows.h`

SetWindowModified

Sets the modification state of the specified window.

```
OSStatus SetWindowModified (
   WindowRef window,
   Boolean modified
);
```

Parameters

window

The window whose modification state is to be set.

modified

Pass `true` if the content of the window has been modified; otherwise, pass `false`.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Discussion
Your application can use the functions `SetWindowModified` and `IsWindowModified` (page 116) instead of maintaining its own separate record of the modification state of the content of a window. The modification state of a window is visually represented by a dot in the window's close box. If the dot is present, the window is modified; if the dot is absent, the window is not modified.

Your application should distinguish between the modification state of the window and the modification state of the window's contents, typically a document. The modification state of the window contents are what should affect `SetWindowModified`. For example, in the case of a word processing document, you call `SetWindowModified` (passing `true` in the `modified` parameter) whenever the user types new characters into the document. However, you do not call `SetWindowModified` when the user moves the window, because that change does not affect the document contents. If you need to track whether the window position has changed, you need to do this with your own flag.

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

Related Sample Code
QTCarbonShell

Declared In
`MacWindows.h`

SetWindowProperty

Associates an arbitrary piece of data with a window.

```
OSStatus SetWindowProperty (
   WindowRef window,
   PropertyCreator propertyCreator,
   PropertyTag propertyTag,
   ByteCount propertySize,
   const void *propertyBuffer
);
```

Parameters

window
The window with which data is to be associated.

propertyCreator
The creator code (typically, the application's signature) of the data to be associated.

propertyTag
A value identifying the data to be associated. You define the tag your application uses to identify the data; this code is not to be confused with the file type for the data.

propertySize
The size of the data to be associated.

propertyBuffer
A pointer to the data to be associated.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Discussion
Data set with the `SetWindowProperty` function may be obtained with the function `GetWindowProperty` (page 75) and removed with the function `RemoveWindowProperty` (page 123).

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

Related Sample Code
HID Calibrator
QTMetaData

Declared In
`MacWindows.h`

SetWindowProxyAlias

Associates a file with a window.

```
OSStatus SetWindowProxyAlias (
   WindowRef inWindow,
   AliasHandle inAlias
);
```

Parameters

inWindow

The window with which the specified file is to be associated.

inAlias

A handle to a structure of type `AliasRecord` for the file to associate with the specified window. You can obtain an alias handle by calling the function `GetWindowProxyAlias` (page 78). The Window Manager copies the alias data, so you can dispose of the alias after `SetWindowProxyAlias` returns.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Discussion

Your application should call the `SetWindowProxyAlias` function to establish a proxy icon for a given window. The creator code and file type of the file associated with a window determine the proxy icon that is displayed for the window.

Because the `SetWindowProxyAlias` function won't work without a saved file, you must establish the initial proxy icon for a new, untitled window with the function `SetWindowProxyCreatorAndType` (page 148), which requires that you know the file type and creator code for the file, but does not require that the file have been saved.

Special Considerations

On Mac OS 8.x and Mac OS 9.x, you must save and restore the current graphics port—by calling the QuickDraw functions `GetPort` and `SetPort`—around each call to the `SetWindowProxyAlias` function.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

SetWindowProxyCreatorAndType

Sets the proxy icon for a window that lacks an associated file.

```
OSStatus SetWindowProxyCreatorAndType (
   WindowRef window,
   OSType fileCreator,
   OSType fileType,
   SInt16 vRefNum
);
```

Parameters

window

The window for which you want to set the proxy icon.

fileCreator

A code that is to be used, together with the `fileType` parameter, to determine the proxy icon. This typically is the creator code of the file that would be created, were the user to save the contents of the window.

fileType

A code that is to be used, together with the `fileCreator` parameter, to determine the proxy icon. This typically is the file type of the file that would be created, were the user to save the contents of the window.

vRefNum

A value identifying the volume containing the default desktop database to search for the icon associated with the file type and creator code specified in the `fileCreator` and `fileType` parameters. Pass `kOnSystemDisk` if the volume is unknown.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Special Considerations

On Mac OS 8.x and Mac OS 9.x, you must save and restore the current graphics port—by calling the QuickDraw functions `GetPort` and `SetPort`—around each call to the `SetWindowProxyCreatorAndType` function.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

SetWindowProxyIcon

Overrides the default proxy icon for a window.

```
OSStatus SetWindowProxyIcon (
   WindowRef window,
   IconRef icon
);
```

Parameters

window

The window for which you want to set the proxy icon.

icon

An icon reference identifying the icon to be used for the window's proxy icon. If there is already a proxy icon in use of the type desired, an `IconRef` value may be obtained for that icon by calling the function `GetWindowProxyIcon` (page 78). Otherwise, your application must call the Icon Services function `GetIconRefFromFile` to get a value of type `IconRef`. The Window Manager retains the `IconRef`, so you can release `icon` after `SetWindowProxyIcon` returns. See the Icon Services and Utilities documentation for a description of the `IconRef` data type.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Discussion
If you want to override the proxy icon that the Window Manager displays by default for a given file, your application should call the `SetWindowProxyIcon` function.

More typically, when you do not want to override a window's default proxy icon, your application would call one of the following functions: `HIWindowSetProxyFSRef` (page 106), `SetWindowProxyAlias` (page 148), or `SetWindowProxyCreatorAndType` (page 148).

On Mac OS 8.x and Mac OS 9.x, you must save and restore the current graphics port—by calling the QuickDraw functions `GetPort` and `SetPort`—around each call to the `SetWindowProxyIcon` function.

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

Declared In
`MacWindows.h`

SetWindowResizeLimits

Sets the maximum and minimum resize limits for windows.

```
OSStatus SetWindowResizeLimits (
   WindowRef inWindow,
   const HISize *inMinLimits,
   const HISize *inMaxLimits
);
```

Parameters

inWindow

The window whose maximum and minimum resize limits are to be set.

inMinLimits

The minimum limits. Pass `NULL` if you don't want to set this limit. For information on the `HISize` data type, see HIGeometry.h.

inMaxLimits

The maximum limits. Pass `NULL` if you don't want to set this limit. For information on the `HISize` data type, see HIGeometry.h.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Availability
Available in Mac OS X v10.2 and later.

Not available to 64-bit applications.

Related Sample Code
HID Calibrator

Declared In
`MacWindows.h`

SetWindowStandardState

Sets a window's standard zoom rectangle.

```
void SetWindowStandardState (
   WindowRef window,
   const Rect *rect
);
```

Parameters

window

The window whose standard zoom rectangle is to be set.

rect

On input, a rectangle (in global coordinates) representing the window's standard zoom rectangle. A window's standard zoom rectangle is the window content bounds when the window is zoomed out to its largest extent.

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

Declared In
`MacWindows.h`

SetWindowTitleWithCFString

Sets the window title to the contents of a Core Foundation string.

```
OSStatus SetWindowTitleWithCFString (
   WindowRef inWindow,
   CFStringRef inString
);
```

Parameters

inWindow

The window whose title is to be set.

inString

The title to set.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

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

Related Sample Code
HID Calibrator
QTCarbonShell
QTMetaData

Declared In
`MacWindows.h`

SetWindowToolbar

Associates a toolbar with a window.

```
OSStatus SetWindowToolbar (
   WindowRef inWindow,
   HIToolbarRef inToolbar
);
```

Parameters

`inWindow`
The window with which the toolbar specified by `inToolbar` is to be associated.

`inToolbar`
The toolbar that is to be associated with the window specified by `inWindow`.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

SetWindowUserState

Sets a window's user zoom rectangle.

```
void SetWindowUserState (
   WindowRef window,
   const Rect *rect
);
```

Parameters

`window`
The window whose user zoom rectangle is to be set.

`rect`
On input, a pointer to a rectangle (in global coordinates) representing the user zoom rectangle that is to be set. The window's user zoom rectangle is the window content bounds when the window is zoomed back in.

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

Declared In
MacWindows.h

SetWRefCon

Sets the refCon field of a window.

```
void SetWRefCon (
   WindowRef window,
   SRefCon data
);
```

Parameters

window
The window whose refCon field is to be set.

data
On input, the data to be placed in the refCon field.

Discussion
The SetWRefCon function places the specified data in the refCon field of the specified window structure. The refCon field is available to your application for any window-related data it needs to store.

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

Related Sample Code
CarbonSketch
QTCarbonShell

Declared In
MacWindows.h

ShowFloatingWindows

Shows an application's floating windows.

```
OSStatus ShowFloatingWindows (
   void
);
```

Return Value
A result code. For details, see "Window Manager Result Codes" (page 247).

Discussion
When an application receives a suspend event, its floating windows are hidden automatically. When the application receives a resume event, the floating windows are made visible automatically. Call this function if you want to make your floating windows visible manually.

See also the function HideFloatingWindows (page 83).

Special Considerations

The `ShowFloatingWindows` function operates only upon windows created with the `kFloatingWindowClass` constant; see "Window Class Constants" (page 184) for more details on this constant.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

ShowHide

Sets a window's visibility.

```
void ShowHide (
   WindowRef window,
   Boolean showFlag
);
```

Parameters

`window`

On input, a pointer to the window structure.

`showFlag`

On input, a Boolean value that specifies its visibility: `true` makes a window visible; `false` makes it invisible.

Discussion

The `ShowHide` function sets a window's visibility to the status specified by the `showFlag` parameter. If the value of `showFlag` is `true`, `ShowHide` makes the window visible if it's not already visible and has no effect if it's already visible. If the value of `showFlag` is `false`, `ShowHide` makes the window invisible if it's not already invisible and has no effect if it's already invisible.

The `ShowHide` function never changes the highlighting or front-to-back ordering of windows and generates no activate events.

Use `ShowHide` only where you need to manually control window activation. Otherwise, use `ShowWindow` or `HideWindow` instead.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

ShowHideWindowToolbar

Shows or hides the toolbar.

```
OSStatus ShowHideWindowToolbar (
   WindowRef inWindow,
   Boolean inShow,
   Boolean inAnimate
);
```

Parameters

inWindow

The window whose toolbar is to be shown or hidden.

inShow

Pass `true` to show the toolbar, `false` otherwise.

inAnimate

Pass `true` to animate the transition, pass `false` for no animation.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

ShowSheetWindow

Shows a sheet window using appropriate visual effects.

```
OSStatus ShowSheetWindow (
   WindowRef inSheet,
   WindowRef inParentWindow
);
```

Parameters

inSheet

The window sheet that is to be shown.

inParentWindow

The parent of the window specified by `inSheet`.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

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

Related Sample Code
QTCarbonShell

Declared In
`MacWindows.h`

ShowWindow

Makes an invisible window visible.

```
void ShowWindow (
    WindowRef window
);
```

Parameters

window

The window that is to be made visible.

Discussion

The `ShowWindow` function makes an invisible window visible. If the specified window is already visible, `ShowWindow` has no effect. Your application typically creates a new window in an invisible state, performs any necessary setup of the content region, and then calls `ShowWindow` to make the window visible.

When you call `ShowWindow` to display a window that is invisible, the Window Manager draws the window frame and sends an event to request the application to draw the content region before the window becomes visible. For compositing windows, the Window Manager sends a `kEventControlDraw` event to each HIView in the window. For non-compositing windows, the Window Manager sends a `kEventWindowDrawContent` event. If a non-compositing window does not handle the `kEventWindowDrawContent`, the Window Manager shows the window and generates an update event to request your application to draw the content region.

If the newly visible window is the frontmost window, `ShowWindow` highlights it if it's not already highlighted and generates an activate event to make it active. The `ShowWindow` function does not activate a window that is not frontmost on the desktop.

Because `ShowWindow` does not change the front-to-back ordering of windows, it is not the inverse of `HideWindow` (page 84). If you make the frontmost window invisible with `HideWindow`, and`HideWindow` has activated another window, you must call both `ShowWindow` and `SelectWindow` (page 129) to bring the original window back to the front.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

BSDLLCTest

CarbonSketch

HID Config Save

HID Explorer

QTCarbonShell

Declared In

`MacWindows.h`

SizeWindow

Sets the size of a window.

```
void SizeWindow (
   WindowRef window,
   short w,
   short h,
   Boolean fUpdate
);
```

Parameters

window

The window whose size is to be set.

w

On input, the new window width, in pixels.

h

On input, the new window height, in pixels.

fUpdate

On input, a Boolean value that specifies whether any newly created area of the content region is to be accumulated into the update region (`true`) or not (`false`). You ordinarily pass a value of `true` to ensure that the area is updated. If you pass `false`, you're responsible for maintaining the update region yourself. For a composited window, this parameter is ignored, and any views that intersect the newly exposed area of the window are automatically invalidated. For more information on adding rectangles to and removing rectangles from the update region, see `InvalWindowRect` (page 109) and `ValidWindowRect` (page 165).

Discussion

The `SizeWindow` function changes the size of the window's graphics port rectangle to the dimensions specified by the w and h parameters, or does nothing if the values of `w` and `h` are both `0`. The Window Manager redraws the window in the new size, recentering the title and truncating it if necessary.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

QTCarbonShell

Declared In

`MacWindows.h`

ToggleDrawer

Toggles the drawer state.

```
OSStatus ToggleDrawer (
   WindowRef inDrawerWindow
);
```

Parameters

inDrawerWindow

The drawer window whose drawer state is to be toggled.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Discussion
If the drawer is currently open or opening, this function closes the drawer. If the drawer is currently closed or closing, this function opens the drawer.

Availability
Available in Mac OS X v10.2 and later.
Not available to 64-bit applications.

Declared In
`MacWindows.h`

TrackBox

Tracks clicks in the collapse, close, size, and zoom boxes, and clicks of the toolbar button.

```
Boolean TrackBox (
   WindowRef window,
   Point thePt,
   WindowPartCode partCode
);
```

Parameters

window
The window in which the mouse button was pressed.

thePt
On input, the location of the cursor when the mouse button was pressed. Your application receives this point from the `where` field in the event structure.

partCode
On input, the part code (`inZoomIn`, `inZoomOut`, `inGoAway`, `inGrow`, `inCollapseBox`, or `inToolbarButton`) returned by `FindWindow` (page 48); see "Part Identifier Constants" (page 239).

Return Value
A Boolean whose value is `true` if the specified part was clicked; otherwise, `false`. If `TrackBox` returns `true`, it also removes highlighting from the specified part.

Discussion
The `TrackBox` function tracks the cursor when the user presses the mouse button while the cursor is in the specified part, retaining control until the mouse button is released. While the button is down, `TrackBox` highlights the part while the cursor is in the part's region.

When the mouse button is released, `TrackBox` removes the highlighting from the part and returns `true` if the cursor is within the part's region and `false` if it is not.

If `TrackBox` returns `true` after tracking the close box, your application should close the window. If `TrackBox` returns `true` after tracking the grow box, your application should call `ResizeWindow` (page 125). If `TrackBox` returns `true` after tracking the collapse box, your application should call `CollapseWindow` (page 37). When tracking the toolbar button, your application should call `ShowHideWindowToolbar` (page 154).

Your application calls the `TrackBox` function when it receives a result code of `inZoomIn` or `inZoomOut` from the function `FindWindow` (page 48). If `TrackBox` returns `true`, your application calculates the standard state, if necessary, and calls the function `ZoomWindow` (page 167) to zoom the window. If `TrackBox` returns `false`, your application does nothing.

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

Declared In
MacWindows.h

TrackGoAway

Tracks the cursor when the user presses the mouse button while the cursor is in the close box.

```
Boolean TrackGoAway (
   WindowRef window,
   Point thePt
);
```

Parameters

window
On input, the window in which the mouse-down event occurred.

thePt
On input, the location of the cursor at the time the mouse button was pressed. Your application receives this point from the `where` field of the event structure.

Return Value
When the mouse button is released, `TrackGoAway` removes the highlighting from the close box and returns `true` if the cursor is within the close region and `false` if it is not.

Discussion
The `TrackGoAway` function tracks cursor activity when the user presses the mouse button while the cursor is in the close box, retaining control until the user releases the mouse button. While the button is down, `TrackGoAway` highlights the close box as long as the cursor is in the close region.

Your application calls the `TrackGoAway` function when it receives a result code of `inGoAway` from `FindWindow` (page 48). If `TrackGoAway` returns `true`, your application calls its own function for closing a window, which can call `DisposeWindow` (page 45) to remove the window from the screen. If `TrackGoAway` returns `false`, your application does nothing.

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

Related Sample Code
HideMenuBar

Declared In
MacWindows.h

TrackWindowProxyDrag

Handles all aspects of the drag process when the user drags a proxy icon.

```
OSStatus TrackWindowProxyDrag (
   WindowRef window,
   Point startPt
);
```

Parameters

window

The window whose proxy icon is being dragged.

startPt

Set the `Point` structure to contain the point, specified in global coordinates, where the mouse-down event that began the drag occurred. Your application may retrieve this value from the `where` field of the `event` structure.

Return Value

A result code. See "Window Manager Result Codes" (page 247). If you receive the error `errUserWantsToDragWindow` (–5607), your application should respond by calling the Window Manager function `DragWindow`. Errors are also returned from the Drag Manager, including `userCanceledErr` (–128).

Discussion

If your application uses proxy icons to represent a type of object (currently, file system entities such as files, folders, and volumes) supported by the Window Manager, your application should call the `TrackWindowProxyDrag` function, and the Window Manager can handle all aspects of the drag process for you. If your application calls the `TrackWindowProxyDrag` function, it does not have to call the Drag Manager function `WaitMouseMoved` before starting to track the drag, as the Window Manager handles this automatically. However, if a proxy icon represents a type of data that the Window Manager does not support, or if you want to implement custom dragging behavior, your application should call the function `TrackWindowProxyFromExistingDrag` (page 160).

Your application detects that a user is dragging one of its proxy icons when the function `FindWindow` (page 48) returns the `inProxyIcon` result code; see "Window Part Code Constants" (page 209) for more details.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

See Also

`HIWindowTrackProxyDrag` (page 108)

Declared In

`MacWindows.h`

TrackWindowProxyFromExistingDrag

Allows custom handling of the drag process when the user drags a proxy icon.

```
OSStatus TrackWindowProxyFromExistingDrag (
   WindowRef window,
   Point startPt,
   DragRef drag,
   RgnHandle inDragOutlineRgn
);
```

Parameters

window

The window whose proxy icon is being dragged.

startPt

Set the `Point` structure to contain the point, specified in global coordinates, where the mouse-down event that began the drag occurred. Your application may retrieve this value from the `where` field of the `event` structure.

drag

A value that refers to the current drag process. Pass in the value produced in the `outNewDrag` parameter of the function `BeginWindowProxyDrag` (page 32). If you are not using `BeginWindowProxyDrag` in conjunction with `TrackWindowProxyFromExistingDrag`, you must create the drag reference yourself with the Drag Manager function `NewDrag`.

inDragOutlineRgn

A region handle representing an outline of the icon being dragged. You may obtain a handle to this region from the `outDragOutlineRgn` parameter of `BeginWindowProxyDrag`. If you are not using `BeginWindowProxyDrag` in conjunction with `TrackWindowProxyFromExistingDrag`, you must create the region yourself.

Return Value

A result code. See "Window Manager Result Codes" (page 247). Errors are also returned from the Drag Manager, including `userCanceledErr` (-128).

Discussion

Typically, if the proxy icon represents a type of object (currently, file system entities such as files, folders, and volumes) supported by the Window Manager, the Window Manager can handle all aspects of the drag process itself, and your application should call the function `TrackWindowProxyDrag` (page 159). However, if the proxy icon represents a type of data that the Window Manager does not support, or if you want to implement custom dragging behavior, your application should call the `TrackWindowProxyFromExistingDrag` function.

The `TrackWindowProxyFromExistingDrag` function accepts an existing drag reference and adds file data if the window contains a file proxy. If your application uses `TrackWindowProxyFromExistingDrag`, you then have the choice of using this function in conjunction with the functions `BeginWindowProxyDrag` (page 32) and `EndWindowProxyDrag` (page 47) or simply calling `TrackWindowProxyFromExistingDrag` and handling all aspects of creating and disposing of the drag yourself.

Your application detects a drag when the function `FindWindow` (page 48) returns the `inProxyIcon` result code; see "Window Part Code Constants" (page 209) for more details.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

See Also

`HIWindowTrackProxyDrag` (page 108)

Declared In

`MacWindows.h`

TransitionWindow

Shows, hides, moves, or resizes a window with appropriate animation and sound.

```
OSStatus TransitionWindow (
   WindowRef inWindow,
   WindowTransitionEffect inEffect,
   WindowTransitionAction inAction,
   const Rect *inRect
);
```

Parameters

inWindow

The window on which to act.

inEffect

The type of visual effect to use. `TransitionWindow` supports the Zoom, Slide, Fade, and Genie transition effects. The Slide effect is supported in Mac OS X and in CarbonLib 1.5 and later. The Fade and Genie effects are supported in Mac OS X v10.3 and later. See "Window Transition Effect Constants" (page 223) for constants and descriptions of these effects.

inAction

The action to take. `TransitionWindow` supports the Show, Hide, Move, and Resize actions. The Move and Resize actions are supported in Mac OS X and in CarbonLib 1.5 and later. See "Window Transition Action Constants" (page 222) for possible values.

inRect

A screen rect in global coordinates, or `NULL` for some transition actions. The interpretation of the rect is dependent on the transition action. For details, see the documentation for each action.

If you pass `kWindowShowTransitionAction` in the `action` parameter then, before calling `TransitionWindow`, set the rectangle to specify the dimensions and position, in global coordinates, of the area from which the zoom is to start. If you pass `NULL`, `TransitionWindow` uses the center of the display screen as the source rectangle.

If you pass `kWindowHideTransitionAction` in the `action` parameter then, before calling `TransitionWindow`, set the rectangle to specify the dimensions and position, in global coordinates, of the area at which the zoom is to end.

If you pass `NULL`, `TransitionWindow` uses the center of the display screen as the destination rectangle.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Discussion

The `TransitionWindow` function displays an animation of a window's transition between the open and closed states, such as that displayed by the Finder. `TransitionWindow` uses the rectangle specified in the `rect` parameter for one end of the animation (the source or the destination of the zoom, depending upon whether the window is being shown or hidden, respectively) and the window's current size and position for the other end of the animation. `TransitionWindow` also plays sounds appropriate to the current theme for the opening and closing actions.

Your application may use `TransitionWindow` instead of the functions `ShowWindow` and `HideWindow`. Like these pre–Mac OS 8.5 Window Manager functions, `TransitionWindow` generates the appropriate update and active events when it shows and hides windows.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code
QTCarbonShell

Declared In
`MacWindows.h`

TransitionWindowAndParent

Shows or hides a window, potentially also moving a second window, with animation and sound.

```
OSStatus TransitionWindowAndParent (
   WindowRef inWindow,
   WindowRef inParentWindow,
   WindowTransitionEffect inEffect,
   WindowTransitionAction inAction,
   const Rect *inRect
);
```

Parameters

inWindow
The window that is to be shown or hidden.

inParentWindow
The window to which inWindow is related. For the Sheet effect, this parameter must be a valid window reference; for other effects, this parameter should be `NULL`.

inEffect
The type of visual effect to use. This function is most commonly used to perform the Sheet transition effect, but it also supports the Zoom, Slide, Fade, and Genie effects. See "Window Transition Effect Constants" (page 223) for constants and descriptions of these effects.

inAction
The action to take on the window. The Show, Hide, Move, and Resize actions are supported. See "Window Transition Action Constants" (page 222) for the appropriate constants.

inRect
A screen rect in global coordinates. The interpretation of the rect is dependent on the transition action; see the documentation for each action for details. May be `NULL` for some transition actions.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

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

Declared In
`MacWindows.h`

TransitionWindowWithOptions

Transitions a window from one state to another with appropriate animation and sound.

```
OSStatus TransitionWindowWithOptions (
   WindowRef inWindow,
   WindowTransitionEffect inEffect,
   WindowTransitionAction inAction,
   const HIRect *inBounds,
   Boolean inAsync,
   TransitionWindowOptions *inOptions
);
```

Parameters

inWindow

The window to transition.

inEffect

The type of visual effect to use. For possible values, see "Window Transition Effect Constants" (page 223) for a description of this value.

inAction

The action to take. For possible values, see "Window Transition Action Constants" (page 222).

inBounds

A screen rect in global coordinates. The interpretation of the rect is dependent on the transition action; see "Window Transition Action Constants" (page 222) for the details of each action. This parameter may be `NULL` for the Show and Hide actions for the Zoom and Sheet effects. This parameter is ignored and must be `NULL` for the Show and Hide actions for the Fade effect.

inAsync

A Boolean whose value indicates whether the transition should run synchronously or asynchronously. If `inAsync` is `true`, this function returns immediately, and the transition runs using an event loop timer. You must run your event loop for the transition to occur. If `inAsync` is `false`, this function blocks until the transition completes. In either case, the `kEventWindowTransitionStarted` and `kEventWindowTransitionCompleted` Carbon events are sent to the transitioning window at the start and end of the transition.

inOptions

Extra information that are required for some transitions. This parameter may be `NULL` if the specified transition effect does not require extra information.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Availability

Available in Mac OS X v10.3 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

UpdateCollapsedWindowDockTile

Updates the image of a window in the dock to the current contents of the window.

```
OSStatus UpdateCollapsedWindowDockTile (
   WindowRef inWindow
);
```

Parameters

inWindow

The window whose image is to be updated.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Discussion
Use this function for periodic updates, not for animation purposes. If you want animation, use `CreateQDContextForCollapsedWindowDockTile` (page 254).

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

Declared In
`MacWindows.h`

ValidWindowRect

Removes a rectangle from a window's update region.

```
OSStatus ValidWindowRect (
   WindowRef window,
   const Rect *bounds
);
```

Parameters

window

The window containing the rectangle you want to remove from the update region.

bounds

Set this structure to specify, in local coordinates, a rectangle to be removed from the window's update region.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Discussion
The `ValidWindowRect` function informs the Window Manager that an area of a window no longer needs to be redrawn. The `ValidWindowRect` function is similar to the `ValidRect` function, but `ValidWindowRect` allows the window that it operates upon to be explicitly specified, instead of operating on the current graphics port, so `ValidWindowRect` does not require the graphics port to be set before its use.

See also the functions `InvalWindowRect` (page 109) and `ValidWindowRgn` (page 166).

Special Considerations

This function should not be used on composited windows. Modifying a composited window's update region does not affect the area of the window to be drawn. A composited window does not use its window update region to control drawing. Instead, a composited window determines what to draw by looking at the invalid regions of the views contained in the window.

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

Declared In
`MacWindows.h`

ValidWindowRgn

Removes a region from a window's update region.

```
OSStatus ValidWindowRgn (
   WindowRef window,
   RgnHandle region
);
```

Parameters

window
The window containing the region you want to remove from the update region.

region
Set this region to specify, in local coordinates, the area to be removed from the window's update region.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Discussion
The `ValidWindowRgn` function informs the Window Manager that an area of a window no longer needs to be redrawn. The `ValidWindowRgn` function is similar to the `ValidRgn` function, but `ValidWindowRgn` allows the window that it operates upon to be explicitly specified, instead of operating on the current graphics port, so `ValidWindowRgn` does not require the graphics port to be set before its use.

See also the functions `InvalWindowRgn` (page 110) and `ValidWindowRect` (page 165).

Special Considerations

This function should not be used on composited windows. Modifying a composited window's update region does not affect the area of the window to be drawn. A composited window does not use its window update region to control drawing. Instead, a composited window determines what to draw by looking at the invalid regions of the views contained in the window.

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

Declared In
`MacWindows.h`

WindowPathSelect

Displays a window path pop-up menu.

```
OSStatus WindowPathSelect (
   WindowRef window,
   MenuRef menu,
   SInt32 *outMenuResult
);
```

Parameters

window

The window for which a window path pop-up menu is to be displayed.

menu

The menu to be displayed for the specified window or `NULL`. If you pass `NULL` in this parameter, the Window Manager provides a default menu and sends a Reveal Object Apple event to the Finder if a menu item is selected. Note that in order to pass `NULL`, a file must currently be associated with the window [call `HIWindowSetProxyFSRef` (page 106) to associate a file with the window]. If you pass a menu, this menu supersedes the default window path pop-up menu. There does not have to be a file currently associated with the window if you pass in your own menu.

outMenuResult

A pointer to a value that, on return, contains the menu and menu item the user chose. The high-order word of the value produced contains the menu ID, and the low-order word contains the item number of the menu item. If the user does not select a menu item, 0 is produced in the high-order word, and the low-order word is undefined. For file menus that have not been overridden, 0 is always produced in this parameter. Pass `NULL` in this parameter if you do not want this information.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Discussion

Your application should call the `WindowPathSelect` function when it detects a Command-click in the title of a window, that is, when the `IsWindowPathSelectClick` (page 271) or `IsWindowPathSelectEvent` (page 117) function returns a value of `true`. Calling `WindowPathSelect` causes the Window Manager to display a window path pop-up menu for your window.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

ZoomWindow

Zooms the window when the user has pressed and released the mouse button with the cursor in the zoom box.

```
void ZoomWindow (
   WindowRef window,
   WindowPartCode partCode,
   Boolean front
);
```

Parameters

window

The window that is to be zoomed.

partCode

On input, the part code (either `inZoomIn` or `inZoomOut`) returned by the `FindWindow` function; see "Part Identifier Constants" (page 239).

front

On return, a Boolean value that determines whether the window is to be brought to the front. If the value of `front` is `true`, the window necessarily becomes the frontmost, active window. If the value of `front` is `false`, the window's position in the window list does not change. Note that if a window was active before it was zoomed, it remains active even if the value of `front` is `false`.

Discussion

The `ZoomWindow` function zooms a window in or out, depending on the value of the `partCode` parameter. Your application calls `ZoomWindow`, passing it the part code returned by `FindWindow` (page 48), when it receives a result of `true` from `TrackBox`. The `ZoomWindow` function then changes the window's port rectangle to either the user state (if the part code is `inZoomIn`) or the standard state (if the part code is `inZoomOut`), as stored in the window state structure, described in the section `WStateData` (page 183).

If the part code is `inZoomOut`, your application ordinarily calculates and sets the standard state before calling `ZoomWindow`.

For best results, call the QuickDraw function `EraseRect`, passing the window's graphics port as the port rectangle, before calling `ZoomWindow`.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

ZoomWindowIdeal

Zooms a window in accordance with human interface guidelines.

```
OSStatus ZoomWindowIdeal (
   WindowRef inWindow,
   WindowPartCode inPartCode,
   Point *ioIdealSize
);
```

Parameters

inWindow

The window that is to be zoomed.

inPartCode

A value specifying the direction of the zoom being requested. Your application passes in the relevant value (either the `inZoomIn` or the `inZoomOut` constant).

ioIdealSize

When you specify `inZoomIn` in the `partCode` parameter, you pass a pointer to the `Point` structure, but do not fill the structure with data. On return, the `Point` structure contains the new height and width of the window's content region, and `ZoomWindowIdeal` restores the previous user state.

When you specify `inZoomOut` in the `partCode` parameter, you pass the ideal height and width of the window's content region in the `Point` structure. On return, the `Point` structure contains the new height and width of the window's content region. `ZoomWindowIdeal` saves the user state of the window and zooms the window to its ideal size for the standard state.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Discussion
Applications should use the `ZoomWindowIdeal` function instead of the older function `ZoomWindow`. When your application calls `ZoomWindowIdeal`, it automatically conforms to the human interface guidelines for determining a window's standard state.

The `ZoomWindowIdeal` function calculates a window's ideal standard state and updates a window's ideal user state independently of the `WStateData` structure. Previously, the window definition function was responsible for updating the user state, but because it relies upon the `WStateData` structure, the window definition function is unaware of the ideal standard state and can no longer track the window's zoom state reliably.

While the Window Manager is reliably aware of the window's zoom state, it cannot record the current user state in the `WStateData` structure, because the window definition function can overwrite that data. Therefore, if your application uses `ZoomWindowIdeal`, the `WStateData` structure is superseded, and the result of the `FindWindow` (page 48) function should be ignored when determining whether a particular user click of the zoom box is a request to zoom in or out. When you adopt `ZoomWindowIdeal` and your application receives a result of either `inZoomIn` or `inZoomOut` from `FindWindow`, your application must use the function `IsWindowInStandardState` (page 115) to determine the appropriate part code to pass in the `partCode` parameter.

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

Related Sample Code
HID Calibrator

Declared In
`MacWindows.h`

Callbacks

WindowDefProcPtr

Defines a pointer to a window definition callback function. Your window definition callback function determines how a window looks and behaves.

```
typedef long (*WindowDefProcPtr) (
    short varCode,
    WindowRef window,
    short message,
    long param
);
```

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

```
long MyWindowDefProc (
    short varCode,
    WindowRef window,
    short message,
    long param
);
```

Parameters

varCode

The window's variation code.

window

A pointer to the window's window structure.

message

A value indicating the task to be performed. The `message` parameter contains one of the values defined in "Window Definition Message Constants" (page 232). Other messages are reserved for internal use by the system. The list in the discussion section that follows explains each of these tasks in detail.

param

Data associated with the task specified by the `message` parameter. If the task requires no data, this parameter is ignored.

Return Value

Your window definition function should perform whatever task is specified by the `message` parameter and return a function result, if appropriate. If the task performed requires no result code, return 0.

Discussion

Various Window Manager functions call a window definition function whenever they need to perform a window-dependent action, such as drawing the window on the screen. If you want to define new, nonstandard windows for your application, you must write a window definition function, compile it in your application, and either use `RegisterWindowDefinition` (page 122) to register it with the system or call `CreateCustomWindow` (page 40) to create the custom window directly.

Note that Carbon does not allow you to store custom window definitions in a `'WDEF'` resource file as you could in pre-Carbon systems.

If you use `RegisterWindowDefinition` (page 122), the Window Manager calls the Resource Manager to access your window definition function with the given resource ID; see "Pre-Appearance Window Definition IDs" in *Window Manager Legacy Reference* for a description of how window definition IDs are derived from resource IDs and variation codes.

The Resource Manager reads your window definition function into memory and returns a handle to it. The Window Manager stores this handle in the `windowDefProc` field of the window structure. Later, when it needs to perform an action on the window, the Window Manager calls the window definition function and passes it the variation code as a parameter.

Your window definition function is responsible for

- drawing the window frame
- reporting the region where mouse-down events occur
- calculating the window's structure region and content region
- drawing the size box
- resizing the window frame when the user drags the size box
- reporting the window's features or the location of a specific window region
- performing any customized initialization or disposal tasks

The Window Manager defines the data type `WindowDefUPP` to identify the universal procedure pointer for this application-defined function:

```
typedef UniversalProcPtr WindowDefUPP;
```

You typically use the `NewWindowDefProc` macro like this:

```
WindowDefUPP myWindowDefUPP;
myWindowDefUPP = NewWindowDefProc(MyWindow);
```

You typically use the `CallWindowDefProc` macro like this:

```
CallWindowDefProc (myWindowDefUPP, varCode, theWindow, message,  param);
```

The `message` parameter contains a value specifying the task to be performed by your window definition function. These tasks are:

- *Drawing the Window Frame*

  When the Window Manager passes `wDraw` in the `message` parameter, your window definition function should respond by drawing the window frame in the current graphics port (which is the Window Manager port). The window part code to be drawn will be passed in the `param` parameter of your window definition function. Your window definition function should perform the following steps:

  - Change the current port from the `WMgrPort` to the `WMgrCPort` to allow the system to draw in the full range of RGB colors.
  - Update the pen attributes, text attributes, and `bkPat` fields in the `WMgrCPort` to the values of the corresponding fields in the `WMgrPort`. The Window Manager automatically transfers the `vis` and `clip` regions.

  The parallelism of the `WMgrPort` and the `WMgrCPort` is maintained only by the window definition functions. All window definition functions that draw in the `WMgrPort` should follow the steps listed above even if the changed fields do not affect their operation.

  You must make certain checks to determine exactly how to draw the frame. If the value of the `visible` field in the window structure is `false`, you should do nothing; otherwise, you should examine the `param` parameter and the status flags in the window structure:

  - If the value of `param` is 0, draw the entire window frame (including the size box, if your window definition function incorporates the size box into the frame).
  - If the value of `param` is 0 and the `hilited` field in the window structure is `true`, highlight the frame to show that the window is active. If the value of the `goAwayFlag` field in the window structure is also `true`, draw a close box in the window frame. If the value of the `spareFlag` field in the window structure is also `true`, draw a zoom box in the window frame.

- If the value of the `param` parameter is `wInGoAway`, redraw the window's close box, with or without highlighting as appropriate.
- If the value of the `param` parameter is `wInZoom`, redraw the window's zoom box, with or without highlighting as appropriate.
- If the value of the `param` parameter is `wInCollapseBox`, redraw the window's collapse box, with or without highlighting as appropriate.

You can call `GetWindowWidgetHilite` (page 82) to determine whether the close, zoom, or collapse box is currently highlighted. This function returns the part code of the currently highlighted part, or zero if no part is highlighted. You should draw the indicated part with highlighting, and draw other parts with no highlighting.

The window frame typically, but not necessarily, includes the window's title, which should be displayed in the system font and system font size. The Window Manager port is already set to use the system font and system font size.

Nothing drawn outside the window's structure region will be visible.

Your window definition function should return 0 as the function result for this message.

- *Reporting the Region of a Mouse-Down Event*

  When the Window Manager passes `wHit` in the `message` parameter, your window definition function should respond by reporting the region of the specified mouse-down event. The mouse location (in global coordinates) of the window frame will be passed into the `param` parameter of your window definition function. The vertical coordinate is in the high-order word of the parameter, and the horizontal coordinate is in the low-order word.

  In response to the `wHit` message, your window definition function should return one of the constants defined in "Window Definition Hit Test Result Code Constants" (page 230).

  In Mac OS 9, return the constants `wInGrow`, `wInGoAway`, `wInZoomIn`, `wInZoomOut`, and `wInCollapseBox` only if the window is active—by convention, the size box, close box, zoom box, and collapse box aren't drawn if the window is inactive. In an inactive document window, for example, a mouse-down event in the part of the title bar that would contain the close box if the window were active is reported as `wInDrag`. In Mac OS X, your WDEF can return these part codes for inactive windows because these boxes are drawn even if the window is inactive.

- *Calculating Regions*

  When the Window Manager passes `wCalcRgns` in the `message` parameter, your window definition function should respond by calculating the window's structure and content regions based on the current graphics port's port rectangle. These regions, whose handles are in the `strucRgn` and `contRgn` fields of the window structure, are in global coordinates. The Window Manager requests this operation only if the window is visible. The mouse location (in global coordinates) of the window frame will be passed into the `param` parameter of your window definition function.

  Your window definition function should call `IsWindowCollapsed` (page 113) to determine its collapse state. Then your window definition function can modify its structure and content regions as appropriate. Typically, a window's content region is empty in a collapsed state.

  When you calculate regions for your own type of window, do not alter the clip region or the visible region of the Window Manager port. The Window Manager and QuickDraw take care of this for you. Altering the Window Manager port's clip region or visible region may damage other windows.

  Your window definition function should return 0 as the function result for this message.

- *Performing Additional Window Initialization*

When the Window Manager passes `wNew` in the `message` parameter, your window definition function should respond by performing any initialization that it may require. If the content region has an unusual shape, for example, you might allocate memory for the region and store the region handle in the `dataHandle` field of the window structure. The initialization function for a standard document window creates the `wStateData` structure for storing zooming data.

Your window definition function should ignore the `param` parameter and return 0 as the function result for this message.

- *Performing Additional Window Disposal Actions*

  When the Window Manager passes `wDispose` in the `message` parameter, your window definition function should respond by performing any additional tasks necessary for disposing of a window. You might, for example, release memory that was allocated by the initialization function. The dispose function for a standard document window disposes of the `wStateData` structure.

  Your window definition function should ignore the `param` parameter and return 0 as the function result for this message.

- *Drawing the Window's Grow Image*

  When the Window Manager passes `wGrow` in the `message` parameter, your window definition function should respond to being resized by drawing a dotted outline of the window in the current graphics port in the pen pattern and mode. (The pen pattern and mode are set up—as `gray` and `notPatXor`—to conform to Appearance-compliant human interface guidelines.)

  A rectangle (in global coordinates) whose upper-left corner is aligned with the port rectangle of the window's graphics port is passed into the `param` parameter of your window definition function. Your grow image should be sized appropriately for the specified rectangle. As the user drags the mouse, the Window Manager sends repeated `wGrow` messages, so that you can change your grow image to match the changing mouse location.

  Your window definition function should return 0 as the function result for this message.

- *Drawing the Size Box*

  When the Window Manager passes `wDrawGIcon` in the `message` parameter, your window definition function should respond by drawing the size box in the content region if the window is active. If the window is inactive, your window definition function should draw whatever is appropriate to show that the window cannot currently be sized. Your window definition function may also draw scroll bar delimiter lines. Your window definition function should ignore the `param` parameter.

  If the size box is located in the window frame, draw the size box in response to a `wDraw` message, not a `wDrawGIcon` message.

  Your window definition function should return 0 as the function result for this message.

- *Reporting Window Features*

  When the Window Manager passes `kWindowMsgGetFeatures` in the `message` parameter, your window definition function should respond by setting the `param` parameter to reflect the features that your window supports. The value passed back in the `param` parameter should be comprised of one or more of the values defined in "Window Feature Bits" (page 207).

  Your window definition function should return 1 as the function result for this message.

- *Returning the Location of Window Regions*

  When the Window Manager passes `kWindowMsgGetRegion` in the `message` parameter, your window definition function should respond by returning the location (in global coordinates) of the specified window region. A pointer to a window region structure will be passed in the `param` parameter.

The window region structure is a structure of type `GetWindowRegionRec` (page 177). Your window definition function should return an operating system status (`OSStatus`) message as the function result for this message. The result code `errWindowRegionCodeInvalid` indicates that the window region passed in was not valid.

Application-defined window definition functions are changed with Appearance Manager 1.0 to support collapse boxes and feature reporting.

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

Declared In
`MacWindows.h`

WindowPaintProcPtr

Defines a pointer to a custom content region painting function.

```
typedef OSStatus (*WindowPaintProcPtr) (
    GDHandle device,
    GrafPtr qdContext,
    WindowRef window,
    RgnHandle inClientPaintRgn,
    RgnHandle outSystemPaintRgn,
    void * refCon
);
```

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

```
OSStatus MyWindowPaintProc (
    GDHandle device,
    GrafPtr qdContext,
    WindowRef window,
    RgnHandle inClientPaintRgn,
    RgnHandle outSystemPaintRgn,
    void * refCon
);
```

Parameters

device
The current graphics device (`GDevice`).

qdContext
The graphics port to draw into. Note that you should draw into this port, not the one associated with the window; the painting region `inClientPaintRgn` is defined relative to this port. The port may be an offscreen graphics world.

window
The window to paint in.

inClientPaintRgn
The region to be painted. Treat as a const. This region is clipped to the intersection of the current graphics device and the `clobberedRgn` parameter passed to `PaintBehind`.

outSystemPaintRgn

The region for the system to paint. Initially empty. If your paint procedure sets this region before returning, the Window Manager will erase this region using the system's window content paint function.

refCon

Application-defined data that you passed to `InstallWindowContentPaintProc` (page 269).

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Discussion
Each window in the system contains a reference to a content paint proc. This proc is called to erase the window's content region during `PaintBehind` or `PaintOne` operations. The client application can override the system paint proc by calling `InstallWindowContentPaintProc` (page 269). A window may only have one paint proc installed at any one time, and the paint proc cannot be retrieved by the client application.

If your content region painting callback returns any value other than noErr, `outSystemPaintRgn` is ignored and the entire area of `inClientPaintRgn` is painted using the system paint proc.

When a previously obscured portion of a window is exposed, the window manager will iterate over active displays and call the window's content paint proc once for each device intersecting the region.

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

Declared In
`MacWindows.h`

Data Types

BasicWindowDescription

Describes basic window characteristics for use in a collection item.

```
struct BasicWindowDescription {
    UInt32 descriptionSize
    Rect windowContentRect
    Rect windowZoomRect
    UInt32 windowRefCon
    UInt32 windowStateFlags
    WindowPositionMethod windowPositionMethod
    UInt32 windowDefinitionVersion
    union {
        struct {
            SInt16 windowDefProc;
            Boolean windowHasCloseBox;
        } versionOne;
        struct {
            WindowClass windowClass;
            WindowAttributes windowAttributes;
        } versionTwo;
    } windowDefinition;
};
typedef struct BasicWindowDescription BasicWindowDescription;
```

Fields

`descriptionSize`

A value specifying the size of the entire `BasicWindowDescription` structure.

`windowContentRect`

A structure of type `Rect`, specifying the initial size and screen location of the window's content area.

`windowZoomRect`

Reserved.

`windowRefCon`

The window's reference value field, which is simply storage space available to your application for any purpose. The value contained in this field persists when the `'WIND'` resource is stored, so you should avoid saving pointers in this field, as they may become stale.

`windowStateFlags`

A 32-bit value whose bits you set to indicate the status of transient window states. See "Basic Window Description State Constant" (page 206) for possible values.

`windowPositionMethod`

The specification last used in the function `RepositionWindow` (page 124) to position this window, if any. See "Window Position Constants" (page 213) for a description of possible values for this field.

`windowDefinitionVersion`

The version of the window definition used for the window. Set this field to a value of 1 if your application is creating a pre–Mac OS 8.5 window, that is, a window lacking class and attribute information. Set this field to a value of 2 if your application is creating a window using class and attribute information. See "Basic Window Description Version Constants" (page 219) for descriptions of these values.

`windowDefinition`

A union of the `versionOne` and `versionTwo` structures. Your application must either specify the window's class and attributes, or it must supply a window definition ID and specify whether or not the window has a close box. See "Window Class Constants" (page 184) and "Window Attributes" (page 194) for descriptions of class and attribute values.

Discussion

The `BasicWindowDescription` structure is a default collection item for a resource of type `'wind'`. You use the `BasicWindowDescription` structure to describe the statically-sized base characteristics of a window.

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

Declared In
`MacWindows.h`

GetGrowImageRegionRec

Defines a region to be XOR'd during a window grow or resize operation.

```
struct GetGrowImageRegionRec {
    Rect growRect;
    RgnHandle growImageRegion;
};
typedef struct GetGrowImageRegionRec GetGrowImageRegionRec;
```

Fields

`growRect`
The window's new bounds in global coordinates.

`growImageRegion`
The grow image region.

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

Declared In
`MacWindows.h`

GetWindowRegionRec

Passed to window definitions in the `kWindowMsgGetRegion` message.

```
struct GetWindowRegionRec {
    RgnHandle winRgn;
    WindowRegionCode regionCode;
};
typedef struct GetWindowRegionRec GetWindowRegionRec;
typedef GetWindowRegionRec * GetWindowRegionPtr;
```

Fields

`winRgn`
A handle to a window region based on the value specified in the `regionCode` field. Modify this region.

`regionCode`
A value representing a given window region; see "Window Region Constants" (page 217).

Special Considerations

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

Declared In
`MacWindows.h`

HIWindowRef

Represents a window.

```
typedef WindowRef HIWindowRef;
```

Availability
Available in Mac OS X v10.3 and later.

Declared In
`MacWindows.h`

MeasureWindowTitleRec

Defines specifications of the window title.

```
struct MeasureWindowTitleRec {
    SInt16 fullTitleWidth;
    SInt16 titleTextWidth;
    Boolean isUnicodeTitle;
    Boolean unused;
};
typedef struct MeasureWindowTitleRec MeasureWindowTitleRec;
typedef MeasureWindowTitleRec * MeasureWindowTitleRecPtr;
```

Fields

`fullTitleWidth`
: Your window definition function sets this field to a value specifying the total width in pixels of the window title text and any proxy icon that may be present, ignoring any compression or truncation that might be required when the title is actually drawn. That is, the specified width should be the ideal width that would be used if the window were sufficiently wide to draw the entire title along with a proxy icon. You should measure the title width using the current system font. If no proxy icon is present, this field should have the same value as the `titleTextWidth` field.

`titleTextWidth`
: Your window definition function sets this field to a value specifying the width in pixels of the window title text, ignoring any compression or truncation that might be required when the title is actually drawn. That is, the specified width should be the ideal width that would be used if the window were sufficiently wide to draw the entire title. You should measure the title width using the current system font.

`isUnicodeTitle`
: Your window definition function may ignore this field; it is reserved for future use.

`unused`
: Your window definition function may ignore this field; it is reserved for future use.

Discussion
If you implement a custom window definition function, when the Window Manager passes the message `kWindowMsgMeasureTitle` in your window definition function's `message` parameter it also passes a pointer to a structure of type `MeasureWindowTitleRec` in the `param` parameter. Your window definition function is responsible for setting the contents of the `MeasureWindowTitleRec` structure to contain data describing the ideal title width.

See "Window Definition Message Constants" (page 232) and "Window Feature Bits" (page 207) for more details on the `kWindowMsgMeasureTitle` message and the corresponding `kWindowCanMeasureTitle` feature flag.

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

Declared In
`MacWindows.h`

PropertyCreator

Defines the creator of a window property.

```
typedef OSType PropertyCreator;
```

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

Declared In
`MacWindows.h`

PropertyTag

Defines a window property tag.

```
typedef OSType PropertyTag;
```

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

Declared In
`MacWindows.h`

PicHandle

Defines a picture handle.

```
typedef PicPtr * PicPatHandle;
```

PixPatHandle

Pixel pattern handle.

```
typedef PixPatPtr * PixPatHandle;
```

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

Declared In
`QuickdrawTypes.h`

RGBColor

RGB color.

```
struct RGBColor {
    unsigned short red;
    unsigned short green;
    unsigned short blue;
};
typedef struct RGBColor;
typedef RGBColor * RGBColorPtr;
```

Fields

`red`
: An unsigned short integer specifying the red value of the color.

`green`
: An unsigned short integer specifying the green value of the color.

`blue`
: An unsigned short integer specifying the red value of the color.

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

Declared In
`IOMacOSTypes.h`

RgnHandle

Region handle.

```
typedef struct OpaqueRgnHandle * RgnHandle;
```

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

Declared In
`QuickdrawTypes.h`

SetupWindowProxyDragImageRec

Defines a window proxy drag image.

```
struct SetupWindowProxyDragImageRec {
    GWorldPtr imageGWorld;
    RgnHandle imageRgn;
    RgnHandle outlineRgn;
};
typedef struct SetupWindowProxyDragImageRec SetupWindowProxyDragImageRec;
```

Fields

`imageGWorld`
: A pointer to the offscreen graphics world containing the drag image. The window definition function must allocate the offscreen graphics world, since the Window Manager has no way of knowing the appropriate size for the drag image. The Window Manager disposes of the offscreen graphics world.

`imageRgn`
: A handle to a region containing the drag image. Only this portion of the offscreen graphics world referred to by the `imageGWorld` field is actually drawn. The Window Manager allocates and disposes of this region.

`outlineRgn`
: A handle to a region containing an outline of the drag image, for use on monitors incapable of displaying the drag image itself. The Window Manager allocates and disposes of this region.

Discussion
If you implement a custom window definition function, when the function `TrackWindowProxyDrag` (page 159) is called, the Window Manager passes the message `kWindowMsgSetupProxyDragImage` in your window definition function's `message` parameter and passes a pointer to a structure of type `SetupWindowProxyDragImageRec` in the `param` parameter. Your window definition function is responsible for setting the contents of the `SetupWindowProxyDragImageRec` structure to contain data describing the proxy icon's drag image.

See "Window Definition Message Constants" (page 232)and "Window Feature Bits" (page 207) for more details on the `kWindowMsgSetupProxyDragImage` message and the corresponding `kWindowCanSetupProxyDragImage` feature flag.

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

Declared In
`MacWindows.h`

TransitionWindowOptions

Defines transition options used when calling `TransitionWindowWithOptions`.

```
struct TransitionWindowOptions {
    UInt32 version;
    EventTime duration;
    WindowRef window;
    void * userData;
};
typedef struct TransitionWindowOptions TransitionWindowOptions;
```

Fields
`version`
: The structure version. You must put `0` in this field.

`duration`
: The duration of the fade, in seconds. For use with the Sheet, Slide, Fade, and Genie transition effects; ignored for other effects. You may pass 0 to use the default duration. The effect is not guaranteed to last precisely this long, but should be a close approximation.

`window`
: The parent window of the sheet; for use with `kWindowSheetTransitionEffect`.

`userData`
: A value that is sent as the `kEventParamUserData` parameter for the `kEventWindowTransitionStarted` and `kEventWindowTransitionCompleted` events.

Availability
Available in Mac OS X v10.3 and later.

Declared In
MacWindows.h

WindowDefSpec

Defines a window definition.

```
struct WindowDefSpec {
    WindowDefType defType
    union {
        WindowDefUPP defProc;
        void * classRef;
        short procID;
        void * rootView;
    } u;
};
typedef struct WindowDefSpec WindowDefSpec;
typedef WindowDefSpec * WindowDefSpecPtr;
```

Fields

`defType`

The window definition type. See "Window Definition Type Constants" (page 229) for a list of possible values.

`u`

A pointer to the window definition, depending on the constant passed into the `defType` field.

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

Declared In
MacWindows.h

WindowDefUPP

Defines a UPP to a specified window definition.

```
typedef WindowDefProcPtr WindowDefUPP;
```

Discussion
For more information, see `WindowDefProcPtr` (page 169).

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

Declared In
MacWindows.h

WindowGroupRef

Represents a window group.

```
typedef struct OpaqueWindowGroupRef * WindowGroupRef;
```

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

Declared In
`MacWindows.h`

WindowPaintUPP

Defines a UPP to the specified region painting callback.

```
typedef WindowPaintProcPtr WindowPaintUPP;
```

Discussion
For more information, see `WindowPaintProcPtr` (page 174).

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

Declared In
`MacWindows.h`

WindowRef

An opaque type that represents a window.

```
typedef WindowPtr WindowRef;
```

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

Declared In
`QuickdrawTypes.h`

WStateData

Stores the user state and the standard state of a window.

```
struct WStateData {
    Rect userState;
    Rect stdState;
};
typedef struct WStateData WStateData;
typedef WStateData * WStateDataPtr;
```

Fields

`userState`

A rectangle that describes the window size and location established by the user.

The Window Manager initializes the user state to the size and location of the window when it is first displayed, and then updates the `userState` field whenever the user resizes a window. Although the user state specifies both the size and location of the window, the Window Manager updates the window state data structure only when the user resizes a window—not when the user merely moves a window.

`stdState`

The rectangle describing the window size and location that your application considers the most convenient, considering the function and contents of the document, the screen space available, and the position of the window in its user state. If your application does not define a standard state, the Window Manager automatically sets the standard state to the entire gray region on the main screen, minus a three-pixel border on all sides. The user cannot change a window's standard state.

Discussion

When the Appearance Manager is available, you should not extend the window state data structure. Instead use the `refCon` field of the color window structure or extend the window record structure.

The zoom box allows the user to alternate quickly between two window positions and sizes: the user state and the standard state. The Window Manager stores the user state and your application stores the standard state in the window state data structure of type `WStateData`. The handle to this structure appears in the `dataHandle` field of the window structure.

The `ZoomWindow` (page 167) function changes the size of a window according to the values in the window state data structure.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`MacWindows.h`

Constants

Window Class Constants

Constants that specify the standard window classes.

```
typedef UInt32 WindowClass;
enum {
    kAlertWindowClass = 1,
    kMovableAlertWindowClass = 2,
    kModalWindowClass = 3,
    kMovableModalWindowClass = 4,
    kFloatingWindowClass = 5,
    kDocumentWindowClass = 6,
    kUtilityWindowClass = 8,
    kHelpWindowClass = 10,
    kSheetWindowClass = 11,
    kToolbarWindowClass = 12,
    kPlainWindowClass = 13,
    kOverlayWindowClass = 14,
    kSheetAlertWindowClass = 15,
    kAltPlainWindowClass = 16,
    kDrawerWindowClass = 20,
    kAllWindowClasses = 0xFFFFFFFF
};
```

Constants

`kAlertWindowClass`

Identifies an alert box window. An alert window is used when the application needs the user's attention immediately. On Mac OS 9 and earlier, a visible alert window will prevent the user from switching to any other application. Use `kThemeBrushAlertBackgroundActive` to draw the background of alert windows. Alert windows are initially placed in the modal window group, given a modality of `kWindowModalityAppModal`, and given an activation scope of `kWindowActivationScopeAll`.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kMovableAlertWindowClass`

Identifies a movable alert box window. Generally, you should use this window class rather than `kAlertWindowClass`. Use `kThemeBrushAlertBackgroundActive` to draw the background of alert windows. Alert windows are initially placed in the modal window group, given a modality of `kWindowModalityAppModal`, and given an activation scope of `kWindowActivationScopeAll`.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kModalWindowClass`

Identifies a modal dialog box window. Use `kThemeBrushDialogBackgroundActive` to draw the background of modal dialog windows. Modal dialog windows are initially placed in the modal window group, given a modality of `kWindowModalityAppModal`, and given an activation scope of `kWindowActivationScopeAll`.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kMovableModalWindowClass`

Identifies a movable modal dialog box window. In Mac OS X and CarbonLib 1.3 and later, use `kThemeBrushMovableModalBackground` to draw the background of alert windows. Alert windows are initially placed in the modal window group, given a modality of `kWindowModalityAppModal`, and given an activation scope of `kWindowActivationScopeAll`.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kFloatingWindowClass`

Identifies a window that floats above all document windows. If your application assigns this constant to a window, the Window Manager ensures that the window has the proper floating behavior. Use `kThemeBrushUtilityWindowBackgroundActive` or `kThemeBrushDocumentWindowBackground` to draw the background of floating windows. Floating windows are initially placed in the floating window group, given a modality of `kWindowModalityNone`, and given an activation scope of `kWindowActivationScopeIndependent`.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kDocumentWindowClass`

Identifies a document window or modeless dialog box window. Use `kThemeBrushDocumentWindowBackground` or your own custom drawing to draw the background of a document window. Document windows are initially placed in the document window group, given a modality of `kWindowModalityNone`, and given an activation scope of `kWindowActivationScopeAll`.The Window Manager assigns this class to pre–Mac OS 8.5 Window Manager windows.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kUtilityWindowClass`

Identifies a utility window. A utility window is similar to a floating window, but it floats above the windows of all applications rather than just above the windows of the application that creates it. Use `kThemeBrushUtilityWindowBackgroundActive` or `kThemeBrushDocumentWindowBackground` to draw the background of utility windows. Utility windows are initially placed in the utility window group, given a modality of `kWindowModalityNone`, and given an activation scope of `kWindowActivationScopeIndependent`.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kHelpWindowClass`

Identifies a window used for help tags. It has no window frame. Typically you should use the Help Manager to display help tags, rather than creating a help tag window yourself. Help windows are initially placed in the help window group, given a modality of `kWindowModalityNone`, and given an activation scope of `kWindowActivationScopeNone`.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kSheetWindowClass`

Identifies a sheet. (Mac OS X only.) Use `kThemeBrushSheetBackgroundOpaque` to draw an opaque background for sheet windows, or `kThemeBrushSheetBackgroundTransparent` to draw a transparent background. Sheet windows are initially placed in the document window group, given a modality of `kWindowModalityNone`, and given an activation scope of `kWindowActivationScopeAll`.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kToolbarWindowClass`

Identifies a toolbar window, which is used to display a palette of controls. A toolbar window is similar to a floating window, and like a floating window, is layered above all application windows except for alert and modal windows, but is layered beneath floating windows. Toolbar windows are initially placed in the toolbar window group, given a modality of `kWindowModalityNone`, and given an activation scope of `kWindowActivationScopeNone`.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kPlainWindowClass`

Identifies a plain window, which has a single-pixel window frame. Plain windows are initially placed in the document window group, given a modality of `kWindowModalityNone`, and given an activation scope of `kWindowActivationScopeAll`.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kOverlayWindowClass`

Identifies an overlay window, which is a completely transparent window. Overlay windows are positioned by default above all other windows, but you can group an overlay window with any other window, at any z-order. Overlay windows are intended as a replacement for the pre-Carbon practice of drawing directly into the Window Manager port. By creating a full-screen overlay window and drawing into it, you can draw over any window in any application without disturbing the contents of the windows beneath your drawing. Overlay windows have a default handler for `kEventWindowPaint` that uses `CGContextClearRect` to clear the overlay window's alpha channel to zero. This ensures the initial transparency of the window. You can install your own `kEventWindowPaint` handler to do your own drawing; typically, you would call through to the default handler with `CallNextEventHandler` first, and then use `QDBeginCGContext` to create your own context for drawing. You can use either QuickDraw or Core Graphics to draw into an overlay window, but you must use Core Graphics to draw if you need any of your drawing to be non-opaque, since QuickDraw always sets the alpha channel of any pixels that it touches to 1.0. (QuickDraw is also deprecated in Mac OS X v10.4 and later.) You can also use the standard window event handler together with regular controls in an overlay window. When using the standard window event handler, you will probably want your `kEventWindowPaint` handler to return `eventNotHandledErr` (after calling the default handler with `CallNextEventHandler` first) so that after the Paint handler returns, the Window Manager will send a `kEventWindowDrawContent` event which the standard window event handler can respond to by drawing the controls in the window. Overlay windows are initially placed in the overlay window group, given a modality of `kWindowModalityNone`, and given an activation scope of `kWindowActivationScopeNone`.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kSheetAlertWindowClass`

Identifies an alert sheet. Use `kThemeBrushSheetBackgroundOpaque` to draw an opaque background for sheet alert windows, or `kThemeBrushSheetBackgroundTransparent` to draw a transparent background. Sheet alert windows are initially placed in the document window group, given a modality of `kWindowModalityNone`, and given an activation scope of `kWindowActivationScopeAll`.

Available in Mac OS X v10.1 and later.

Declared in `MacWindows.h`.

`kAltPlainWindowClass`
: Identifies an alternate plain window, which is similar to a plain window but has a solid black shadow on its right and bottom sides. It is rarely used in modern Mac OS applications. Alternate plain windows are initially placed in the document window group, given a modality of `kWindowModalityNone`, and given an activation scope of `kWindowActivationScopeAll`.

  Available in Mac OS X v10.1 and later.

  Declared in `MacWindows.h`.

`kDrawerWindowClass`
: Identifies a drawer. Use `kThemeBrushDrawerBackground` or `kThemeBrushDocumentWindowBackground` to draw the background of drawer windows. Drawer windows are initially placed in the document window group, given a modality of `kWindowModalityNone`, and given an activation scope of `kWindowActivationScopeAll`. Drawer windows should always be created using the compositing window attribute.

  Available in Mac OS X v10.2 and later.

  Declared in `MacWindows.h`.

`kAllWindowClasses`
: Specifier used to designate all window classes. Used with `GetFrontWindowOfClass`, `FindWindowOfClass`, and `GetNextWindowOfClass` to indicate that there should be no restriction on the class of the returned window. Also used with `GetWindowGroupOfClass` to get the root window group.

  Available in Mac OS X v10.0 and later.

  Declared in `MacWindows.h`.

Discussion
The `WindowClass` constants categorize windows into groups of like types. The grouping of windows facilitates the appropriate display (that is, both the look and the front-to-back ordering) and tracking of windows.

You can define a window's class using the function `CreateNewWindow` (page 41) and obtain a window's class using the function `GetWindowClass` (page 61). You can change the class of certain windows by calling `HIWindowChangeClass` (page 87).

Window Attribute Identifiers

Constants that specify standard window attributes.

```
enum {
    kHIWindowBitCloseBox = 1,
    kHIWindowBitZoomBox = 2,
    kHIWindowBitCollapseBox = 4,
    kHIWindowBitResizable = 5,
    kHIWindowBitSideTitlebar = 6,
    kHIWindowBitToolbarButton = 7,
    kHIWindowBitUnifiedTitleAndToolbar = 8,
    kHIWindowBitTextured = 9,
    kHIWindowBitNoTitleBar = 10,
    kHIWindowBitTexturedSquareCorners = 11,
    kHIWindowBitNoTexturedContentSeparator = 12,
    kHIWindowBitDoesNotCycle = 16,
    kHIWindowBitNoUpdates = 17,
    kHIWindowBitNoActivates = 18,
    kHIWindowBitOpaqueForEvents = 19,
    kHIWindowBitCompositing = 20,
    kHIWindowBitFrameworkScaled = 21,
    kHIWindowBitNoShadow = 22,
    kHIWindowBitCanBeVisibleWithoutLogin = 23,
    kHIWindowBitAsyncDrag = 24,
    kHIWindowBitHideOnSuspend = 25,
    kHIWindowBitStandardHandler = 26,
    kHIWindowBitHideOnFullScreen = 27,
    kHIWindowBitInWindowMenu = 28,
    kHIWindowBitLiveResize = 29,
    kHIWindowBitIgnoreClicks = 30,
    kHIWindowBitNoConstrain = 32,
    kHIWindowBitDoesNotHide = 33,
    kHIWindowBitAutoViewDragTracking = 34,
    kHIWindowBitDoesNotShowBadgeInDock = 35
};
```

Constants

`kHIWindowBitCloseBox`

The window has a close box. This attribute is available for windows of class `kDocumentWindowClass` (page 186), `kFloatingWindowClass` (page 186), and `kUtilityWindowClass` (page 186).

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitZoomBox`

The window has a zoom box. This attribute is available for windows of class `kDocumentWindowClass` (page 186), `kFloatingWindowClass` (page 186), and `kUtilityWindowClass` (page 186). When this attribute is set on a window, both the `kWindowHorizontalZoomAttribute` and `kWindowVerticalZoomAttribute` bits are set automatically.

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitCollapseBox`
The window has a collapse box. This attribute is available for windows of class `kDocumentWindowClass` (page 186), `kFloatingWindowClass` (page 186), and `kUtilityWindowClass` (page 186). For floating and utility window classes, this attribute must be added to the window after the window is created; it may not be added to the window at creation time.

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitResizable`
The window has a resize tab or box and is resizable. This attribute is available for windows of class `kDocumentWindowClass` (page 186), `kMovableModalWindowClass` (page 185), `kFloatingWindowClass` (page 186), `kUtilityWindowClass` (page 186), and `kSheetWindowClass` (page 186).

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitSideTitlebar`
The window has a vertical title bar on the side of the window. This attribute is available for windows of the `kFloatingWindowClass` (page 186) and `kUtilityWindowClass` (page 186) class.

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitToolbarButton`
The window has a toolbar button. This oblong clear button shows and hides the toolbar. This attribute is available for windows of class `kDocumentWindowClass` (page 186).

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitUnifiedTitleAndToolbar`
The window draws its window title and toolbar using a unified appearance that has no separator between the two areas. A window may not have both this attribute and the `kHIWindowBitTextured` attribute.

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitTextured`
The window uses the textured or brushed-metal appearance. Drawers can also be textured, but dynamically adjust their appearance based on their parent window's appearance; it is not necessary to specify this attribute for a textured drawer. This attribute is available for windows of class `kDocumentWindowClass` (page 186) and `kFloatingWindowClass` (page 186).

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitNoTitleBar`
The window's title bar can be hidden. This attribute is available for windows of class `kDocumentWindowClass` (page 186), `kFloatingWindowClass` (page 186), and `kUtilityWindowClass` (page 186).

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitTexturedSquareCorners`

Indicates that a textured window should have square corners. By default, a textured window has round corners. When this attribute is set on a window, the window frame view automatically makes the grow box view opaque, and when this attribute is cleared, the window frame view automatically makes the grow box view transparent. You can change the grow box view transparency after modifying this attribute with the function `HIGrowBoxViewSetTransparent`. Relevant only for textured windows; ignored in non-textured windows.

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitNoTexturedContentSeparator`

Indicates that no border is drawn between the toolbar and window content. This attribute is relevant only in textured windows; it is ignored in non-textured windows.

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitDoesNotCycle`

The window does not participate in window cycling invoked by Command-~ or keyboard shortcuts defined in the Keyboard & Mouse preference pane.

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitNoUpdates`

The window does not receive update events. This attribute is available for all windows.

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitNoActivates`

The window does not receive activate events. This attribute is available for all windows.

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitOpaqueForEvents`

The window receives mouse events even for areas of the window that are transparent (that is, have an alpha channel component of zero).

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitCompositing`

The window uses HIView-based compositing, which means that the entire window is comprised of HIViews, and can be treated thusly. This attribute must be specified at window creation; you may not add this attribute after the window has been created.

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitFrameworkScaled`
The window's content is scaled to match the display scale factor. This attribute can only be used when `kHIWindowBitCompositing` is also enabled. When this attribute is enabled, you may not draw with QuickDraw in the window. If this attribute is enabled and if the scale factor is something other than 1.0, the window's scale mode is `kHIWindowScaleModeFrameworkScaled`. If you specify this attribute and `kHIWindowBitApplicationScaled`, the `kHIWindowBitApplicationScaled` attribute is ignored. You may only specify this attribute at window creation time.

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitNoShadow`
The window has no shadow. This attribute is available for all windows, and is given automatically to windows of class `kOverlayWindowClass` (page 187).

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitCanBeVisibleWithoutLogin`
The window can be made visible prior to user login. By default, in Mac OS X 10.5 and later no windows can be visible before a user logs into the system; this protects the user against certain types of malicious use of insecure applications. However, some software, such as input methods or other accessibility software, may need to deliberately make windows available prior to user login. Such software should add this window attribute to its windows.

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitAsyncDrag`
The window server drags the window automatically. Your application should not call `DragWindow` (page 46) for this window because this function would fight with the Window Server for control. This attribute is ignored if the window is grouped with other windows in a window group that has the `kWindowGroupAttrMoveTogether` attribute.

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitHideOnSuspend`
The window is hidden automatically on suspend and shown on resume. This attribute available for all windows and is given automatically to windows of class `kFloatingWindowClass` (page 186), `kHelpWindowClass` (page 186), and `kToolbarWindowClass` (page 187).

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitStandardHandler`
The window supports the standard window event handler. The standard event handler provides standard actions for common window events. See *Carbon Event Manager Programming Guide* for details. This attribute is available for all windows.

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitHideOnFullScreen`
The window is automatically hidden during full-screen mode (when the menubar is invisible) and shown afterwards. Available for all windows. This attribute is automatically given to windows of class `kUtilityWindowClass` (page 186).

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitInWindowMenu`
The window title appears in the system-generated Window menu. This attribute is only available for windows of class `kDocumentWindowClass` (page 186) and is automatically given to windows of that class.

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitLiveResize`
The window supports live resizing. This attribute is available for all windows.

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitIgnoreClicks`
The window never receives mouse events, even in areas that are opaque. Instead, clicks on the window are passed through to windows beneath it.

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitNoConstrain`
The window is not repositioned by the default `kEventWindowConstrain` handler in response to changes in monitor size, Dock position, and so on.

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitDoesNotHide`
The window does not hide when the application is hidden.

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitAutoViewDragTracking`
The window automatically installs Drag Manager callbacks to detect drag actions, and automatically sends HIView drag Carbon events. Setting this attribute is equivalent to calling the function `SetAutomaticControlDragTrackingEnabledForWindow` (and calling that function will set this attribute).

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowBitDoesNotShowBadgeInDock`
Indicates that the Dock should not add a badge to this window's icon when the window is minimized to the Dock.

Discussion
In Mac OS X version 10.5 and later, you may use these constants to set or test the attributes of a window. For example, you may use them with the function `HIWindowCreate` (page 92) to define the attributes of a new window, the function `HIWindowChangeAttributes` (page 86) to change a window's attributes, and the function `HIWindowTestAttribute` (page 108) to test whether a window has a specific attribute.

Window Attributes

Bit masks that specify standard window attributes. In Mac OS X v10.5 and later, you may use "Window Attribute Identifiers" (page 188) instead.

```
typedef UInt32 WindowAttributes;
enum {
    kWindowNoAttributes = 0,
    kWindowCloseBoxAttribute = (1L << 0),
    kWindowHorizontalZoomAttribute = (1L << 1),
    kWindowVerticalZoomAttribute = (1L << 2),
    kWindowFullZoomAttribute = (kWindowVerticalZoomAttribute |
kWindowHorizontalZoomAttribute),
    kWindowCollapseBoxAttribute = (1L << 3),
    kWindowResizableAttribute = (1L << 4),
    kWindowSideTitlebarAttribute = (1L << 5),
    kWindowToolbarButtonAttribute = (1L << 6),
    kWindowUnifiedTitleAndToolbarAttribute = (1L << 7),
    kWindowMetalAttribute = (1L << 8),
    kWindowNoTitleBarAttribute = (1L << 9),
    kWindowTexturedSquareCornersAttribute = (1L << 10),
    kWindowMetalNoContentSeparatorAttribute = (1L << 11),
    kWindowDoesNotCycleAttribute = (1L << 15),
    kWindowNoUpdatesAttribute = (1L << 16),
    kWindowNoActivatesAttribute = (1L << 17),
    kWindowOpaqueForEventsAttribute = (1L << 18),
    kWindowCompositingAttribute = (1L << 19),
    kWindowFrameworkScaledAttribute = (1L << 20),
    kWindowNoShadowAttribute = (1L << 21),
    kWindowCanBeVisibleWithoutLoginAttribute = (1L << 22),
    kWindowAsyncDragAttribute = (1L << 23),
    kWindowHideOnSuspendAttribute = (1L << 24),
    kWindowStandardHandlerAttribute = (1L << 25),
    kWindowHideOnFullScreenAttribute = (1L << 26),
    kWindowInWindowMenuAttribute = (1L << 27),
    kWindowLiveResizeAttribute = (1L << 28),
    kWindowIgnoreClicksAttribute = (1L << 29),
    kWindowNoConstrainAttribute = (1L << 31),
    kWindowStandardDocumentAttributes = (kWindowCloseBoxAttribute |
kWindowFullZoomAttribute | kWindowCollapseBoxAttribute | kWindowResizableAttribute),
    kWindowStandardFloatingAttributes = (kWindowCloseBoxAttribute |
kWindowCollapseBoxAttribute)
};
```

Constants

`kWindowNoAttributes`

If no bits are set, the window has none of the standard attributes.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowCloseBoxAttribute`

If the bit specified by this mask is set, the window has a close box. See `kHIWindowBitCloseBox` (page 189).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowHorizontalZoomAttribute`
If the bit specified by this mask is set, the window changes width when zooming. See `kHIWindowBitZoomBox` (page 189).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowVerticalZoomAttribute`
If the bit specified by this mask is set, the window changes height when zooming. See `kHIWindowBitZoomBox` (page 189).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowFullZoomAttribute`
If the bits specified by this mask are set, the window changes both width and height when zooming. See `kHIWindowBitZoomBox` (page 189).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowCollapseBoxAttribute`
If the bit specified by this mask is set, the window has a collapse box. See `kHIWindowBitCollapseBox` (page 190).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowResizableAttribute`
If the bit specified by this mask is set, the window has a resize tab or box and is resizable. See `kHIWindowBitResizable` (page 190).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowSideTitlebarAttribute`
If the bit specified by this mask is set, the window has a side title bar. See `kHIWindowBitSideTitlebar` (page 190).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowToolbarButtonAttribute`
If the bit specified by this mask is set, the window has a toolbar button. See `kHIWindowBitToolbarButton` (page 190).

Available in Mac OS X v10.1 and later.

Declared in `MacWindows.h`.

`kWindowUnifiedTitleAndToolbarAttribute`
If the bit specified by this mask is set, the window draws its window title and toolbar using a unified appearance that has no separator between the two areas. A window may not have both this attribute and the `kWindowMetalAttribute` attribute.

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kWindowMetalAttribute`
If the bit specified by this mask is set, the window has a textured or brushed-metal appearance. See `kHIWindowBitTextured` (page 190).

Available in Mac OS X v10.2 and later.

Declared in `MacWindows.h`.

`kWindowNoTitleBarAttribute`
If the bit specified by this mask is set, the window's title bar can be hidden. See `kHIWindowBitNoTitleBar` (page 190).

Available in Mac OS X v10.4 and later.

Declared in `MacWindows.h`.

`kWindowTexturedSquareCornersAttribute`
See `kHIWindowBitTexturedSquareCorners` (page 191).

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kWindowMetalNoContentSeparatorAttribute`
If the bit specified by this mask is set, no border is drawn between the toolbar and window content. See `kHIWindowBitNoTexturedContentSeparator` (page 191).

Available in Mac OS X v10.4 and later.

Declared in `MacWindows.h`.

`kWindowDoesNotCycleAttribute`
If the bit specified by this mask is set, the window does not participate in window cycling. See `kHIWindowBitDoesNotCycle` (page 191).

Available in Mac OS X v10.3 and later.

Declared in `MacWindows.h`.

`kWindowNoUpdatesAttribute`
If the bit specified by this mask is set, the window does not receive update events. See `kHIWindowBitNoUpdates` (page 191).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowNoActivatesAttribute`
If the bit specified by this mask is set, the window does not receive activate events. See `kHIWindowBitNoActivates` (page 191).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowOpaqueForEventsAttribute`
If the bit specified by this mask is set, the window receives mouse events even for areas of the window that are transparent. See `kHIWindowBitOpaqueForEvents` (page 191).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowCompositingAttribute`
If the bit specified by this mask is set, the window uses HIView-based compositing. See `kHIWindowBitCompositing` (page 191).

Available in Mac OS X v10.2 and later.

Declared in `MacWindows.h`.

`kWindowFrameworkScaledAttribute`
If the bit specified by this mask is set, this window's content is scaled to match the display scale factor. See `kHIWindowBitFrameworkScaled` (page 192).

Available in Mac OS X v10.4 and later.

Declared in `MacWindows.h`.

`kWindowNoShadowAttribute`
If the bit specified by this mask is set, the window has no shadow. See `kHIWindowBitNoShadow` (page 192).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowCanBeVisibleWithoutLoginAttribute`
If the bit specified by this mask is set, the window can be made visible prior to user login. See `kHIWindowBitCanBeVisibleWithoutLogin` (page 192).

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kWindowAsyncDragAttribute`
If the bit specified by this mask is set, the window server drags the window automatically. See `kHIWindowBitAsyncDrag` (page 192).

Available in Mac OS X v10.3 and later.

Declared in `MacWindows.h`.

`kWindowHideOnSuspendAttribute`
If the bit specified by this mask is set, the window is hidden automatically on suspend and shown on resume. See `kHIWindowBitHideOnSuspend` (page 192).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowStandardHandlerAttribute`
If the bit specified by this mask is set, the window supports the standard window event handler. See `kHIWindowBitStandardHandler` (page 192).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowHideOnFullScreenAttribute`
If the bit specified by this mask is set, the window is automatically hidden during fullscreen mode (when the menubar is invisible) and shown afterwards. See `kHIWindowBitHideOnFullScreen` (page 193).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowInWindowMenuAttribute`
If the bit specified by this mask is set, the window title appears in the system-generated Window menu. See `kHIWindowBitInWindowMenu` (page 193).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowLiveResizeAttribute`
If the bit specified by this mask is set, the window supports live resizing. See `kHIWindowBitLiveResize` (page 193).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowIgnoreClicksAttribute`
If the bit specified by this mask is set, the window never receives mouse events, even in areas that are opaque. See `kHIWindowBitIgnoreClicks` (page 193).

Available in Mac OS X v10.2 and later.

Declared in `MacWindows.h`.

`kWindowNoConstrainAttribute`
If the bit specified by this mask is set, the window is not repositioned by the default `kEventWindowConstrain` handler. See `kHIWindowBitNoConstrain` (page 193).

Available in Mac OS X v10.1 and later.

Declared in `MacWindows.h`.

`kWindowStandardDocumentAttributes`
If the bits specified by this mask are set, the window has the attributes of a standard document window—that is, a close box, full zoom box, collapse box, and size box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowStandardFloatingAttributes`
If the bits specified by this mask are set, the window has the attributes of a standard floating window—that is, a close box and collapse box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

Discussion
The `WindowAttributes` enumeration defines masks your application can use to set or test the bits in a window attributes parameter. You can use these masks with the function `CreateNewWindow` (page 41) to define a window's attributes, and with the function `ChangeWindowAttributes` (page 34) to change a window's attributes. You can also use these masks to test the attributes parameter produced by the function `GetWindowAttributes` (page 59), thereby obtaining a window's attributes.

User Focus Auto-Select Constant

Defines a constant that tells the system to pick the best user focus window.

```
#define kUserFocusAuto  ((WindowRef)(-1))
```

Constants
`kUserFocusAuto`
Pass this constant to the function `SetUserFocusWindow` (page 134) to have the system choose the most appropriate window for user focus.

Available in Mac OS X v10.2 and later.

Declared in `MacWindows.h`.

Appearance-Compliant Window Resource IDs

Define window resources for Appearance-compliant applications.

```
enum {
    kWindowDocumentDefProcResID = 64,
    kWindowDialogDefProcResID = 65,
    kWindowUtilityDefProcResID = 66,
    kWindowUtilitySideTitleDefProcResID = 67,
    kWindowSheetDefProcResID = 68,
    kWindowSimpleDefProcResID = 69,
    kWindowSheetAlertDefProcResID = 70
};
```

Constants

`kWindowDocumentDefProcResID`
: Defines Appearance-compliant standard document windows with a size box. Standard document windows created with this resource ID can use variation codes to create windows with vertical and horizontal zoom boxes.

  Available with Appearance 1.0 and later.

  Declared in `MacWindows.h`.

`kWindowDialogDefProcResID`
: Defines Appearance-compliant dialog and alert boxes. Modal and movable modal dialog boxes created with this resource ID are displayed with no space between their content and structure region. Alert boxes created with this resource ID are displayed with a red-tinged border.

  Declared in `MacWindows.h`.

  Available with Appearance 1.0 and later.

`kWindowUtilityDefProcResID`
: Defines Appearance-compliant utility (floating) windows with a top title bar and a size box.

  Available with Appearance 1.0 and later.

  Declared in `MacWindows.h`.

`kWindowUtilitySideTitleDefProcResID`
: Defines Appearance-compliant utility (floating) windows with a side title bar and a size box.

  Available with Appearance 1.0 and later.

  Declared in `MacWindows.h`.

`kWindowSheetDefProcResID`
: Defines a window sheet.

  Available in Mac OS X v10.0 and later.

  Declared in `MacWindows.h`.

`kWindowSimpleDefProcResID`
: Defines a simple window with no window frame.

  Available in Mac OS X v10.0 and later.

  Declared in `MacWindows.h`.

`kWindowSheetAlertDefProcResID`
: Defines a sheet window that is displayed as an alert (rather than a dialog) on Mac OS 9.

  Available in Mac OS X v10.1 and later.

  Declared in `MacWindows.h`.

Discussion
Window resource IDs are changed with Appearance Manager 1.0. The Window Manager now provides many new standard, Appearance-compliant window resource IDs for your application.

You can use a window resource ID constant to create a window definition ID; see "Pre-Appearance Window Definition IDs" in*Window Manager Legacy Reference* for more details.

Note that the standard Appearance-compliant resource ID constants `kWindowDocumentDefProcResID`, `kWindowUtilityDefProcResID`, and `kWindowUtilitySideTitleDefProcResID` specify windows with collapse boxes.

Resource IDs 0 through 127 are reserved for use by the system.

Appearance-Compliant Window Definition ID Constants

Define different window kinds.

```
enum {
    kWindowDocumentProc = 1024,
    kWindowGrowDocumentProc = 1025,
    kWindowVertZoomDocumentProc = 1026,
    kWindowVertZoomGrowDocumentProc = 1027,
    kWindowHorizZoomDocumentProc = 1028,
    kWindowHorizZoomGrowDocumentProc = 1029,
    kWindowFullZoomDocumentProc = 1030,
    kWindowFullZoomGrowDocumentProc = 1031
};
enum {
    kWindowPlainDialogProc = 1040,
    kWindowShadowDialogProc = 1041,
    kWindowModalDialogProc = 1042,
    kWindowMovableModalDialogProc = 1043,
    kWindowAlertProc = 1044,
    kWindowMovableAlertProc = 1045
};
enum {
    kWindowMovableModalGrowProc = 1046
};
enum {
    kWindowFloatProc = 1057,
    kWindowFloatGrowProc = 1059,
    kWindowFloatVertZoomProc = 1061,
    kWindowFloatVertZoomGrowProc = 1063,
    kWindowFloatHorizZoomProc = 1065,
    kWindowFloatHorizZoomGrowProc = 1067,
    kWindowFloatFullZoomProc = 1069,
    kWindowFloatFullZoomGrowProc = 1071
};
enum {
    kWindowFloatSideProc = 1073,
    kWindowFloatSideGrowProc = 1075,
    kWindowFloatSideVertZoomProc = 1077,
    kWindowFloatSideVertZoomGrowProc = 1079,
    kWindowFloatSideHorizZoomProc = 1081,
    kWindowFloatSideHorizZoomGrowProc = 1083,
    kWindowFloatSideFullZoomProc = 1085,
    kWindowFloatSideFullZoomGrowProc = 1087
};
enum {
    kWindowSheetProc = 1088,
    kWindowSheetAlertProc = 1120
};
enum {
    kWindowSimpleProc = 1104,
    kWindowSimpleFrameProc = 1105
};
```

Constants

`kWindowDocumentProc`

Appearance-compliant movable window with no size box or zoom box. Available with Appearance 1.0 and later. The corresponding pre-Appearance window definition ID is `noGrowDocProc`.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowGrowDocumentProc`
: Appearance-compliant standard document window (movable window with size box). Available with Appearance 1.0 and later. The corresponding pre-Appearance window definition ID is `documentProc`.

  Available in Mac OS X v10.0 and later.

  Declared in `MacWindows.h`.

`kWindowVertZoomDocumentProc`
: Appearance-compliant window with vertical zoom box and no size box. Available with Appearance 1.0 and later. There is no corresponding pre-Appearance window definition ID.

  Available in Mac OS X v10.0 and later.

  Declared in `MacWindows.h`.

`kWindowVertZoomGrowDocumentProc`
: Appearance-compliant window with vertical zoom box and size box. Available with Appearance 1.0 and later. There is no corresponding pre-Appearance window definition ID.

  Available in Mac OS X v10.0 and later.

  Declared in `MacWindows.h`.

`kWindowHorizZoomDocumentProc`
: Appearance-compliant window with horizontal zoom box and no size box. Available with Appearance 1.0 and later. There is no corresponding pre-Appearance window definition ID.

  Available in Mac OS X v10.0 and later.

  Declared in `MacWindows.h`.

`kWindowHorizZoomGrowDocumentProc`
: Appearance-compliant window with horizontal zoom box and size box. Available with Appearance 1.0 and later. There is no corresponding pre-Appearance window definition ID.

  Available in Mac OS X v10.0 and later.

  Declared in `MacWindows.h`.

`kWindowFullZoomDocumentProc`
: Appearance-compliant window with full zoom box and no size box. Available with Appearance 1.0 and later. The corresponding pre-Appearance window definition ID is `zoomNoGrow`.

  Available in Mac OS X v10.0 and later.

  Declared in `MacWindows.h`.

`kWindowFullZoomGrowDocumentProc`
: Appearance-compliant window with full zoom box and size box. Available with Appearance 1.0 and later. The corresponding pre-Appearance window definition ID is `zoomDocProc`.

  Available in Mac OS X v10.0 and later.

  Declared in `MacWindows.h`.

`kWindowPlainDialogProc`
: Appearance-compliant modeless dialog box. Available with Appearance 1.0 and later. The corresponding pre-Appearance window definition ID is `plainDBox`.

  Available in Mac OS X v10.0 and later.

  Declared in `MacWindows.h`.

`kWindowShadowDialogProc`
Appearance-compliant modeless dialog box with shadow. Available with Appearance 1.0 and later. The corresponding pre-Appearance window definition ID is `altDBoxProc`.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowModalDialogProc`
Appearance-compliant modal dialog box. Available with Appearance 1.0 and later. The corresponding pre-Appearance window definition ID is `dBoxProc`.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowMovableModalDialogProc`
Appearance-compliant movable modal dialog box. Available with Appearance 1.0 and later. The corresponding pre-Appearance window definition ID is `movableDBoxProc`.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowAlertProc`
Appearance-compliant alert box. Available with Appearance 1.0 and later. There is no corresponding pre-Appearance window definition ID.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowMovableAlertProc`
Appearance-compliant movable alert box. Available with Appearance 1.0 and later. There is no corresponding pre-Appearance window definition ID.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowMovableModalGrowProc`
Appearance-compliant movable modal dialog box with size box. Available with Appearance 1.0.1 and later. There is no corresponding pre-Appearance window definition ID.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowFloatProc`
Appearance-compliant utility (floating) window with no size box or zoom box. Available with Appearance 1.0 and later. The corresponding pre-Appearance window definition ID is `floatProc`.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowFloatGrowProc`
Appearance-compliant utility (floating) window with a size box. Available with Appearance 1.0 and later. The corresponding pre-Appearance window definition ID is `floatGrowProc`.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowFloatVertZoomProc`
Appearance-compliant utility (floating) window with a vertical zoom box. Available with Appearance 1.0 and later. There is no corresponding pre-Appearance window definition ID.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowFloatVertZoomGrowProc`
Appearance-compliant utility (floating) window with a vertical zoom box and size box. Available with Appearance 1.0 and later. There is no corresponding pre-Appearance window definition ID.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowFloatHorizZoomProc`
Appearance-compliant utility (floating) window with a horizontal zoom box. Available with Appearance 1.0 and later. There is no corresponding pre-Appearance window definition ID.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowFloatHorizZoomGrowProc`
Appearance-compliant utility (floating) window with a horizontal zoom box and size box. Available with Appearance 1.0 and later. There is no corresponding pre-Appearance window definition ID.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowFloatFullZoomProc`
Appearance-compliant utility (floating) window with full zoom box. Available with Appearance 1.0 and later. The corresponding pre-Appearance window definition ID is `floatZoomProc`.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowFloatFullZoomGrowProc`
Appearance-compliant utility (floating) window with full zoom box and size box. Available with Appearance 1.0 and later. The corresponding pre-Appearance window definition ID is `floatZoomGrowProc`.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowFloatSideProc`
Appearance-compliant utility (floating) window with side title bar. Available with Appearance 1.0 and later. The corresponding pre-Appearance window definition ID is `floatSideProc`.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowFloatSideGrowProc`
Appearance-compliant utility (floating) window with side title bar and size box. Available with Appearance 1.0 and later. The corresponding pre-Appearance window definition ID is `floatSideGrowProc`.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowFloatSideVertZoomProc`
Appearance-compliant utility (floating) window with side title bar and vertical zoom box. Available with Appearance 1.0 and later. There is no corresponding pre-Appearance window definition ID.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowFloatSideVertZoomGrowProc`
Appearance-compliant utility (floating) window with side title bar, vertical zoom box, and size box. Available with Appearance 1.0 and later. There is no corresponding pre-Appearance window definition ID.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowFloatSideHorizZoomProc`
Appearance-compliant utility (floating) window with side title bar and horizontal zoom box. Available with Appearance 1.0 and later. There is no corresponding pre-Appearance window definition ID.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowFloatSideHorizZoomGrowProc`
Appearance-compliant utility (floating) window with side title bar, horizontal zoom box, and size box. Available with Appearance 1.0 and later. There is no corresponding pre-Appearance window definition ID.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowFloatSideFullZoomProc`
Appearance-compliant utility (floating) window with side title bar and full zoom box. Available with Appearance 1.0 and later. The corresponding pre-Appearance window definition ID is `floatSideZoomProc`.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowFloatSideFullZoomGrowProc`
Appearance-compliant utility (floating) window with side title bar, full zoom box, and size box. Available with Appearance 1.0 and later. The corresponding pre-Appearance window definition ID is `floatSideZoomGrowProc`.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowSheetProc`
A standard document sheet.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowSheetAlertProc`
An alert sheet.

Available in Mac OS X v10.1 and later.

Declared in `MacWindows.h`.

`kWindowSimpleProc`
A window that has no structure region; the content covers the entire window.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowSimpleFrameProc`
A window that has a 1-pixel black frame as its structure.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

Basic Window Description State Constant

Define the window description state constant.

```
enum {
    kWindowIsCollapsedState = (1 << 0L)
};
```

Constants

`kWindowIsCollapsedState`
If the bit specified by this mask is set, the window is currently collapsed.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

Discussion

You can use this mask to set a bit in the `windowStateFlags` field of a structure of type `BasicWindowDescription` (page 175), thereby specifying a transient window state.

Window Frame View Part Codes

HIView part codes used by window frame views.

```
enum {
    kHIWindowTitleBarPart = 2,
    kHIWindowDragPart = 3,
    kHIWindowTitleProxyIconPart = 2
};
```

Constants

`kHIWindowTitleBarPart`
Identifies the title bar part of a window frame view. This part code is used by the functions `GetWindowBounds` (page 60) and `GetWindowRegion` (page 265) when called with `kWindowTitleBarRgn`.

Available in Mac OS X v10.5 and later.

Declared in `HIWindowViews.h`.

`kHIWindowDragPart`
Identifies the draggable part of a window frame view. This part code is used by `GetWindowBounds` (page 60) and `GetWindowRegion` (page 265) when called with `kWindowDragRgn`.

Available in Mac OS X v10.5 and later.

Declared in `HIWindowViews.h`.

`kHIWindowTitleProxyIconPart`
Identifies the proxy icon part of a window frame title view. The title view is a subview of the window frame view and is identified by an HIViewID of `kHIViewWindowTitleID`. This part code is not used by the window frame view itself, but only by the title view. This part code is used by `GetWindowBounds` (page 60) and `GetWindowRegion` (page 265) when called with `kWindowTitleProxyIconRgn`.

Available in Mac OS X v10.5 and later.

Declared in `HIWindowViews.h`.

Discussion
These part codes are used by an HIView that implements the frame of a window. They may be used with the standard document windows provided by the Window Manager. A custom window frame view may optionally (but is not required to) implement these part codes in its event handlers for `kEventControlGetPartRegion` and `kEventControlGetPartBounds`.

Window Feature Bits

Specify features available in a window.

```
enum {
    kWindowCanGrow = (1 << 0),
    kWindowCanZoom = (1 << 1),
    kWindowCanCollapse = (1 << 2),
    kWindowIsModal = (1 << 3),
    kWindowCanGetWindowRegion = (1 << 4),
    kWindowIsAlert = (1 << 5),
    kWindowHasTitleBar = (1 << 6),
    kWindowSupportsDragHilite = (1 << 7),
    kWindowSupportsModifiedBit = (1 << 8),
    kWindowCanDrawInCurrentPort = (1 << 9),
    kWindowCanSetupProxyDragImage = (1 << 10),
    kWindowCanMeasureTitle = (1 << 11),
    kWindowWantsDisposeAtProcessDeath = (1 << 12),
    kWindowSupportsGetGrowImageRegion = (1 << 13),
    kWindowIsOpaque = (1 << 14),
    kWindowDefSupportsColorGrafPort = 0x40000002
};
```

Constants
`kWindowCanGrow`
If this bit (bit 0) is set, the window has a grow box (may not be visible).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowCanZoom`
If this bit (bit 1) is set, the window has a zoom box (may not be visible).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowCanCollapse`
If this bit (bit 2) is set, the window has a collapse box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowIsModal`
If this bit (bit 3) is set, the window should behave as modal.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowCanGetWindowRegion`
If this bit (bit 4) is set, the window supports a call to `GetWindowRegion` (page 265).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowIsAlert`
If this bit (bit 5) is set, the window is an alert box (may be movable or not). When this constant is added to `kWindowIsModal`, the user should be able to switch out of the application and move the alert box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowHasTitleBar`
If this bit (bit 6) is set, the window has a title bar.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowSupportsDragHilite`
If the bit specified by this mask is set, the window supports the `kWindowMsgDragHilite` message. (Mac OS 8.5 and later.)

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowSupportsModifiedBit`
If the bit specified by this mask is set, the window supports the `kWindowMsgModified` message. (Mac OS 8.5 and later.)

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowCanDrawInCurrentPort`
If the bit specified by this mask is set, the window supports the `kWindowMsgDrawInCurrentPort` message. (Mac OS 8.5 and later.)

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowCanSetupProxyDragImage`
If the bit specified by this mask is set, the window supports the `kWindowMsgSetupProxyDragImage` message. (Mac OS 8.5 and later.)

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowCanMeasureTitle`
If the bit specified by this mask is set, the window supports the `kWindowMsgMeasureTitle` message. (Mac OS 8.5 and later.)

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowWantsDisposeAtProcessDeath`

If the bit specified by this mask is set, the window definition function wants to receive a `wDispose` message for the window if it still exists when the application quits.

Previously, the Window Manager would send a `wDispose` message only if the application explicitly closed the window with calls to the `CloseWindow` or `DisposeWindow` functions. The Window Manager would delete a window that still existed when the application called `ExitToShell` without notifying the window definition function, as part of the destruction of the process.

Note that if a window has the `kWindowWantsDisposeAtProcessDeath` feature bit set, the Window Manager sends your window definition function a `wDispose` message for the window when the application exits for any cause, including if your application crashes.

A window might want to set this feature flag if it allocates data when it is initialized that lives outside of the application heap and that is not automatically disposed when the application quits. The `wDispose` message is sent very early in the termination process, so it is still safe for the window definition function to call the system back (for example, you may want to do this in order to dispose of any auxiliary data). However, to ensure compatibility and to create the minimum performance impact, the window definition function should try to do as little as possible after receiving a `wDispose` message sent during the termination process. (Mac OS 8.5 and later.)

This feature is only available in Mac OS 8 and 9. It is not supported in Mac OS X.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowSupportsGetGrowImageRegion`

(Mac OS 8.5 and later.)

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowIsOpaque`

Indicates that the window is entirely opaque. If this feature bit is set, the window will use less memory because no alpha channel information will be stored for the window's pixels.

Available in Mac OS X v10.1 and later.

Declared in `MacWindows.h`.

`kWindowDefSupportsColorGrafPort`

Indicates that the window definition does not require that the current port be the classic Window Manager port. Not supported in Mac OS X.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

Discussion

For descriptions of the messages that correspond to these feature flags, see "Window Definition Message Constants" (page 232).

Window Part Code Constants

Indicate which part of the window was hit.

```
typedef SInt16 WindowPartCode;
enum {
    inDesk = 0,
    inNoWindow = 0,
    inMenuBar = 1,
    inSysWindow = 2,
    inContent = 3,
    inDrag = 4,
    inGrow = 5,
    inGoAway = 6,
    inZoomIn = 7,
    inZoomOut = 8,
    inCollapseBox = 11,
    inProxyIcon = 12,
    inToolbarButton = 13,
    inStructure = 15
};
```

Constants

`inDesk`

The cursor is in the desktop region, not in the menu bar, a driver window, or any window that belongs to your application. When `FindWindow` (page 48) returns `inDesk`, your application doesn't need to do anything.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`inNoWindow`

The cursor is not in a window.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`inMenuBar`

The user has pressed the mouse button while the cursor is in the menu bar. When `FindWindow` returns `inMenuBar`, your application typically adjusts its menus and then calls the Menu Manager function `MenuSelect` to let the user choose menu items.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`inSysWindow`

Not supported by Carbon.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`inContent`

The user has pressed the mouse button while the cursor is in the content area (excluding the size box in an active window) of one of your application's windows. When `FindWindow` returns `inContent`, your application determines how to handle clicks in the content region.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`inDrag`

The user has pressed the mouse button while the cursor is in the drag region of a window. When `FindWindow` returns `inDrag`, your application typically calls `DragWindow` to let the user drag the window to a new location.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`inGrow`

The user has pressed the mouse button while the cursor is in an active window's size box. When `FindWindow` returns `inGrow`, your application typically calls `ResizeWindow` (page 125).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`inGoAway`

The user has pressed the mouse button while the cursor is in an active window's close box. When `FindWindow` returns `inGoAway`, your application typically calls`TrackGoAway` (page 159) to track mouse activity while the button is down and then calls its own function for closing a window if the user releases the button while the cursor is in the close box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`inZoomIn`

The user has pressed the mouse button while the cursor is in the zoom box of an active window that is currently in the standard state. When `FindWindow` returns `inZoomIn`, your application typically calls `TrackBox` to track mouse activity while the button is down and then calls its own function for zooming a window if the user releases the button while the cursor is in the zoom box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`inZoomOut`

The user has pressed the mouse button while the cursor is in the zoom box of an active window that is currently in the user state. When `FindWindow` returns `inZoomOut`, your application typically calls the function `TrackBox` to track mouse activity while the button is down. Your application then calls its own function for zooming a window if the user releases the button while the cursor is in the zoom box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`inCollapseBox`

The user has pressed the mouse button while the cursor is in an active window's collapse box. When `FindWindow` returns `inCollapseBox`, your application typically does nothing, because the system will collapse your window for you.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`inProxyIcon`

The user has pressed the mouse button while the cursor is in the proxy icon of a window. When `FindWindow` returns `inProxyIcon`, your application typically calls the function `TrackWindowProxyDrag` (page 159).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`inToolbarButton`
The user has pressed the mouse button while the cursor is in the toolbar button. (Mac OS X only.)

Available in Mac OS X v10.1 and later.

Declared in `MacWindows.h`.

`inStructure`
The user has pressed the mouse button while the cursor is in the window's structure region.

Available in Mac OS X v10.1 and later.

Declared in `MacWindows.h`.

Discussion
When your application receives a mouse-down event, you typically call `FindWindow` (page 48), which returns an integer that specifies the location, in global coordinates, of the cursor at the time the user pressed the mouse button.

Window Modality Options

Specify the modality of a window.

```
typedef UInt32 WindowModality;
enum {
    kWindowModalityNone = 0,
    kWindowModalitySystemModal = 1,
    kWindowModalityAppModal = 2,
    kWindowModalityWindowModal = 3
};
```

Constants
`kWindowModalityNone`
A window does not prevent interaction with any other window in the system.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowModalitySystemModal`
The window is system-modal. In Mac OS 9 and earlier, the user cannot perform any other action until the window is dismissed. In Mac OS X, this constant produces the same behavior as `kWindowModalityAppModal`, so there is no way to prevent the user from interacting with windows from other applications.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowModalityAppModal`
The window is application-modal; that is the user cannot perform any other action within the application until the window is dismissed. The user can switch to other applications, however.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowModalityWindowModal`
The window is document-modal; the user cannot perform any other action within the current document window until the modal window associated with it is dismissed. The user can switch to other windows or applications, however. Sheets are document-modal.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

Window Position Constants

Define where to place a window.

```
typedef UInt32 WindowPositionMethod;
enum {
    kWindowCenterOnMainScreen = 1,
    kWindowCenterOnParentWindow = 2,
    kWindowCenterOnParentWindowScreen = 3,
    kWindowCascadeOnMainScreen = 4,
    kWindowCascadeOnParentWindow = 5,
    kWindowCascadeOnParentWindowScreen = 6,
    kWindowCascadeStartAtParentWindowScreen = 10,
    kWindowAlertPositionOnMainScreen = 7,
    kWindowAlertPositionOnParentWindow = 8,
    kWindowAlertPositionOnParentWindowScreen = 9
};
```

Constants

`kWindowCenterOnMainScreen`
Center the window, both horizontally and vertically, on the screen that contains the menu bar.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowCenterOnParentWindow`
Center the window, both horizontally and vertically, on the parent window. If the window to be centered is wider than the parent window, its left edge is aligned with the parent window's left edge.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowCenterOnParentWindowScreen`
Center the window, both horizontally and vertically, on the screen containing the parent window. In Mac OS X v10.3 and later, the parent window may be the same as the positioned window. In CarbonLib and earlier versions of Mac OS X, the parent window must be different from the positioned window.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowCascadeOnMainScreen`
Place the window just below the menu bar at the left edge of the main screen. Subsequent windows are placed on the screen relative to the first window, such that the frame of the preceding window remains visible behind the current window. The exact amount by which windows are offset depends upon the dimensions of the window frame under a given appearance.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowCascadeOnParentWindow`
: Place the window a distance below and to the right of the upper-left corner of the parent window such that the frame of the parent window remains visible behind the current window. The exact amount by which windows are offset depends upon the dimensions of the window frame under a given appearance.

  Available in Mac OS X v10.0 and later.

  Declared in `MacWindows.h`.

`kWindowCascadeOnParentWindowScreen`
: Place the window just below the menu bar at the left edge of the screen containing the parent window. Subsequent windows are placed on the screen relative to the first window, such that the frame of the preceding window remains visible behind the current window. The exact amount by which windows are offset depends upon the dimensions of the window frame under a given appearance. In Mac OS X v10.3 and later, the parent window may be the same as the positioned window. In CarbonLib and earlier versions of Mac OS X, the parent window must be different from the positioned window.

  Available in Mac OS X v10.0 and later.

  Declared in `MacWindows.h`.

`kWindowCascadeStartAtParentWindowScreen`
: Cascade the window on the screen containing the largest portion of its parent window, starting below and to the right of its parent window. The parent window must be different from the positioned window. (Available in Mac OS X v10.2 and CarbonLib 1.6 and later.)

  Available in Mac OS X v10.2 and later.

  Declared in `MacWindows.h`.

`kWindowAlertPositionOnMainScreen`
: Center the window horizontally and position it vertically on the screen that contains the menu bar, such that about one-fifth of the screen is above it. In Mac OS X v10.3 and later, the parent window may be the same as the positioned window. In CarbonLib and earlier versions of Mac OS X, the parent window must be different from the positioned window.

  Available in Mac OS X v10.0 and later.

  Declared in `MacWindows.h`.

`kWindowAlertPositionOnParentWindow`
: Center the window horizontally and position it vertically such that about one-fifth of the parent window is above it.

  Available in Mac OS X v10.0 and later.

  Declared in `MacWindows.h`.

`kWindowAlertPositionOnParentWindowScreen`
: Center the window horizontally and position it vertically such that about one-fifth of the screen containing the parent window is above it.

  Available in Mac OS X v10.0 and later.

  Declared in `MacWindows.h`.

Discussion

To specify the factors that determine how a window should be positioned, you supply one of these `WindowPositionMethod` constants to the function `RepositionWindow` (page 124) or in the `BasicWindowDescription` structure of a resource of type `'wind'`. Do not confuse the `WindowPositionMethod` constants with the pre–Mac OS 8.5 Window Manager window positioning constants or use the `WindowPositionMethod` constants where the older constants are required (such as in the `StandardAlert` function or in `'WIND'`, `'DLOG'`, or `'ALRT'` resources).

System 7 Window Positioning Constants

Define window positioning constants used in `'WIND'`, `'DLOG'`, or `'ALRT'` resources, as well as the `StandardAlert` function.

```
enum {
    kWindowNoPosition = 0x0000,
    kWindowDefaultPosition = 0x0000,
    kWindowCenterMainScreen = 0x280A,
    kWindowAlertPositionMainScreen = 0x300A,
    kWindowStaggerMainScreen = 0x380A,
    kWindowCenterParentWindow = 0xA80A,
    kWindowAlertPositionParentWindow = 0xB00A,
    kWindowStaggerParentWindow = 0xB80A,
    kWindowCenterParentWindowScreen = 0x680A,
    kWindowAlertPositionParentWindowScreen = 0x700A,
    kWindowStaggerParentWindowScreen = 0x780A
};
```

Constants

`kWindowNoPosition`
No position.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowDefaultPosition`
Use the initial location.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowCenterMainScreen`
Center the window on the main screen.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowAlertPositionMainScreen`
Place the window in the alert position on the main screen.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowStaggerMainScreen`
Stagger the window on the main screen.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowCenterParentWindow`
Center the window on the parent window.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowAlertPositionParentWindow`
Place the window in the alert position on the parent window.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowStaggerParentWindow`
Stagger the window relative to the parent window.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowCenterParentWindowScreen`
Center the window on the parent window screen.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowAlertPositionParentWindowScreen`
Place the window in the alert position on the parent window screen.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowStaggerParentWindowScreen`
Stagger the window on the parent window screen.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

Discussion
You can use these constants in the optional positioning specification field of the window resource and in the dialog resource to override the window position established by the rectangle specified for the window or dialog. These positioning constants are convenient when the user is creating new documents or when you are handling your own dialog boxes and alert boxes.

These constants are passed into the `StandardAlert` function and are used in' `WIND` ', 'DLOG', and 'ALRT' templates. `StandardAlert` uses zero to specify the default position. Other calls use zero to specify “no position”.

Do not pass these constants to the`RepositionWindow` function or store these constants in the `BasicWindowDescription` structure of a 'WIND' resource.

The meaning of the terms used in the window positioning constant descriptions are as follows:

- center

  Centered both horizontally and vertically, relative either to a screen or to another window (if a window to be centered relative to another window is wider than the window that preceded it, it is pinned to the left edge; a narrower window is centered)

- stagger

  Located 20 pixels to the right and 20 pixels below the upper-left corner of the last window (in the case of staggering relative to a screen, the first window is placed just below the menu bar at the left edge of the screen, and subsequent windows are placed on that screen relative to the first window)

- alert position

  Centered horizontally and placed in the “alert position” vertically, that is, with about one-fifth of the window or screen above the new window and the rest below

- parent window

  Place in the position of the window in which the user was last working based on the frontmost window before the new window comes up.

Window Region Constants

Define various window regions.

```
typedef UInt16 WindowRegionCode;
enum {
    kWindowTitleBarRgn = 0,
    kWindowTitleTextRgn = 1,
    kWindowCloseBoxRgn = 2,
    kWindowZoomBoxRgn = 3,
    kWindowDragRgn = 5,
    kWindowGrowRgn = 6,
    kWindowCollapseBoxRgn = 7,
    kWindowTitleProxyIconRgn = 8,
    kWindowStructureRgn = 32,
    kWindowContentRgn = 33,
    kWindowUpdateRgn = 34,
    kWindowOpaqueRgn = 35,
    kWindowGlobalPortRgn = 40,
    kWindowToolbarButtonRgn = 41
};
```

Constants

`kWindowTitleBarRgn`

The entire area occupied by a window's title bar, including the title text region.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowTitleTextRgn`

That portion of a window's title bar that is occupied by the name of the window.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowCloseBoxRgn`

The area occupied by a window's close box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowZoomBoxRgn`

The area occupied by a window's zoom box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowDragRgn`

The draggable area of the window frame; this area includes the title bar and window outline and excludes the size box, close box, zoom box, and collapse box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowGrowRgn`

The area occupied by a window's size box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowCollapseBoxRgn`
The area occupied by a window's collapse box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowTitleProxyIconRgn`
Specifies the region in the window's title area that contains the proxy icon. The proxy icon region is always located within the window's title text region.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowStructureRgn`
The entire area occupied by a window, including the frame and content region; the window may be partially off-screen but its structure region does not change.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowContentRgn`
The window's content region—the part of a window in which your application displays the contents of the window or dialog, including the size box and any controls.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowUpdateRgn`
The window's update region––the part of the window that needs to be redrawn.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowOpaqueRgn`
Area of window considered to be opaque. Only valid for windows with alpha channels. (Mac OS X only)

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowGlobalPortRgn`
Bounds of the window's port in global coordinates; not affected by `CollapseWindow` (page 37).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowToolbarButtonRgn`
Bounds of the toolbar button area.

Available in Mac OS X v10.4 and later.

Declared in `MacWindows.h`.

Discussion
You can pass constants of type `WindowRegionCode` in the `inRegionCode` parameter of `GetWindowRegion` (page 265) to obtain a handle to a specific window region. The `WindowRegionCode` constants are available with Appearance Manager 1.0 and later.

Version Notes
With the Window Manager in Mac OS 8.5 and later, you may pass the `kWindowTitleProxyIconRgn`, `kWindowStructureRgn`, and `kWindowContentRgn` constants to the function `GetWindowRegion` (page 265).

Window Latent Visibility Constants

Defines window latent visibility constants.

```
typedef UInt32 WindowLatentVisibility;
enum {
    kWindowLatentVisibleFloater = 1 << 0,
    kWindowLatentVisibleSuspend = 1 << 1,
    kWindowLatentVisibleFullScreen = 1 << 2,
    kWindowLatentVisibleAppHidden = 1 << 3,
    kWindowLatentVisibleCollapsedOwner = 1 << 4,
    kWindowLatentVisibleCollapsedGroup = 1 << 5
};
```

Constants

`kWindowLatentVisibleFloater`
The window is a floating window, and floating windows are hidden.
Available in Mac OS X v10.1 and later.
Declared in `MacWindows.h`.

`kWindowLatentVisibleSuspend`
The window has `kWindowHideOnSuspendAttribute` set and the application is suspended.
Available in Mac OS X v10.1 and later.
Declared in `MacWindows.h`.

`kWindowLatentVisibleFullScreen`
The window has `kWindowHideOnFullScreenAttribute` set and the mode is full-screen.
Available in Mac OS X v10.1 and later.
Declared in `MacWindows.h`.

`kWindowLatentVisibleAppHidden`
The window's process is hidden.
Available in Mac OS X v10.1 and later.
Declared in `MacWindows.h`.

`kWindowLatentVisibleCollapsedOwner`
The window is in an owned group, and the owner was collapsed.
Available in Mac OS X v10.1 and later.
Declared in `MacWindows.h`.

`kWindowLatentVisibleCollapsedGroup`
The window is in a group for which `kWindowGroupAttrHideOnCollapse` is set, and another window in the group was collapsed.
Available in Mac OS X v10.1 and later.
Declared in `MacWindows.h`.

Basic Window Description Version Constants

Describe different Mac OS window versions.

```
enum {
    kWindowDefinitionVersionOne = 1,
    kWindowDefinitionVersionTwo = 2
};
```

Constants

`kWindowDefinitionVersionOne`

Specifies a pre–Mac OS 8.5 Window Manager window. Windows of this version are created using a window definition ID and a Boolean value indicating whether or not the window has a close box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowDefinitionVersionTwo`

Specifies a Mac OS 8.5 Window Manager window. Windows of this version are created using class and attribute information. For details on classes and attributes, see "Window Class Constants" (page 184) and "Window Attributes" (page 194) respectively.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

Discussion

You may supply one of these values in the `windowDefinitionVersion` field of a structure of type `BasicWindowDescription` (page 175) to specify the version of the window definition used for a window.

Window Property Persistent Constant

Define the window property persistent constant.

```
enum {
    kWindowPropertyPersistent = 0x00000001
};
```

Constants

`kWindowPropertyPersistent`

Indicates this property gets saved when the window is archived. Note, however, that window properties are not archived at all in Mac OS X v10.4.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

Window Variant Constants

Specify window variants.

```
enum {
    kDocumentWindowVariantCode = 0,
    kModalDialogVariantCode = 1,
    kPlainDialogVariantCode = 2,
    kShadowDialogVariantCode = 3,
    kMovableModalDialogVariantCode = 5,
    kAlertVariantCode = 7,
    kMovableAlertVariantCode = 9,
    kSideFloaterVariantCode = 8
};
```

Constants

`kDocumentWindowVariantCode`
Variation code for a document window.
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

`kModalDialogVariantCode`
Variation code for modal dialog boxes. The code can be added to 16 x the resource ID constant `kStandardWindowDefinition` to create a standard, pre-Appearance modal dialog box window.
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

`kPlainDialogVariantCode`
Variation code for a plain dialog box.
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

`kShadowDialogVariantCode`
Variation code for a shadow dialog box.
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

`kMovableModalDialogVariantCode`
Variation code for movable modal dialog boxes. The code can be added to 16 x the resource ID constant `kStandardWindowDefinition` to create a standard, pre-Appearance movable modal dialog box.
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

`kAlertVariantCode`
Variation code for a standard alert box.
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

`kMovableAlertVariantCode`
Variation code for a movable alert box.
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

`kSideFloaterVariantCode`
Variation code for utility (floating) windows with a side title bar. The code can be added to 16 x the resource ID constant `kFloatingWindowDefinition` to create a standard, pre-Appearance utility (floating) window with a side title bar.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

Window Transition Action Constants

Specify the type of window action taking place.

```
typedef UInt32 WindowTransitionAction;
enum {
    kWindowShowTransitionAction = 1,
    kWindowHideTransitionAction = 2,
    kWindowMoveTransitionAction = 3,
    kWindowResizeTransitionAction = 4
};
```

Constants

`kWindowShowTransitionAction`
Specifies that the animation display the window opening, that is, transitioning from a closed to an open state.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowHideTransitionAction`
Specifies that the animation display the window closing, that is, transitioning from an open to a closed state.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowMoveTransitionAction`
Moves the window. Use with the Slide transition effect. The `inRect` parameter contains the global coordinates of the window's new structure bounds and cannot be `NULL`. (Available in Mac OS X, and in CarbonLib 1.5 and later.)

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowResizeTransitionAction`
Resizes the window. Use with the Slide transition effect. The `inRect` parameter contains the global coordinates of the window's new structure bounds and cannot be `NULL`. (Available in Mac OS X and in CarbonLib 1.5 and later.)

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

Discussion

You may pass these `WindowTransitionAction` constants to the function `TransitionWindow` (page 162) to specify the direction of the animation effect that is to be performed for a window.

Window Transition Effect Constants

Designate the type of transition effect to use to show or hide the window.

```
typedef UInt32 WindowTransitionEffect;
enum {
    kWindowZoomTransitionEffect = 1,
    kWindowSheetTransitionEffect = 2,
    kWindowSlideTransitionEffect = 3,
    kWindowFadeTransitionEffect = 4,
    kWindowGenieTransitionEffect = 5
};
```

Constants

`kWindowZoomTransitionEffect`

Specifies an animation that displays the window zooming between the open and closed states. The direction of the animation, whether from open to closed, or closed to open, depends upon the `WindowTransitionAction` constant specified in conjunction with the `WindowTransitionEffect` constant; see"Window Transition Action Constants" (page 222) for descriptions of possible values.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowSheetTransitionEffect`

Zoom in or out from the parent window. Use with `TransitionWindowAndParent` (page 163) and Show or Hide transition actions. (Available in Mac OS X, and in CarbonLib 1.5 and later.)

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowSlideTransitionEffect`

Slide the window into its new position. Use with `TransitionWindow` (page 162) and Move or Resize transition actions. (Available in Mac OS X and in CarbonLib 1.5 and later.)

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowFadeTransitionEffect`

Fade the window into or out of visibility. Use with the Show or Hide transition action. (Available in Mac OS X v10.3 and later.)

Available in Mac OS X v10.3 and later.

Declared in `MacWindows.h`.

`kWindowGenieTransitionEffect`

Use the Genie effect that the Dock uses to minimize or maximize a window to show or hide the window. Use with the Show or Hide transition action. (Available in Mac OS X v10.3 and later.)

Available in Mac OS X v10.3 and later.

Declared in `MacWindows.h`.

Discussion

You may pass this `WindowTransitionEffect` constant to the function `TransitionWindow` (page 162) to specify the type of animation effect that is to be performed for a window.

Window Activation Scope Constants

Defines window activation scope constants.

```
typedef UInt32 WindowActivationScope;
enum {
    kWindowActivationScopeNone = 0,
    kWindowActivationScopeIndependent = 1,
    kWindowActivationScopeAll = 2
};
```

Constants

`kWindowActivationScopeNone`

Windows with this scope are never activated by the Window Manager. Use `kWindowActivationScopeNone` when the window's visual state does not change based on activation (for example, tooltip windows), or when the client wants to manually control all activation. The window owner is free to explicitly activate or deactivate a window by calling `ActivateWindow` (page 30).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowActivationScopeIndependent`

Windows with this scope are always active if visible and are unaffected by the activation state of other windows. This activation scope is automatically used by floating windows.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowActivationScopeAll`

Windows with this scope are activated relative to other windows with the same scope in the current process. Only one window with this scope can be active in the entire process. This activation scope is automatically used by document and dialog windows.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

Window Constrain Options

Constrain options for window resize, growing, and so on.

```
typedef UInt32 WindowConstrainOptions;
enum {
    kWindowConstrainMayResize = (1L << 0),
    kWindowConstrainMoveRegardlessOfFit = (1L << 1),
    kWindowConstrainAllowPartial = (1L << 2),
    kWindowConstrainCalcOnly = (1L << 3),
    kWindowConstrainUseTransitionWindow = (1L << 4),
    kWindowConstrainMoveMinimum = 1 << 6,
    kWindowConstrainUseSpecifiedBounds = 1 << 8,
    kWindowConstrainStandardOptions = kWindowConstrainMoveRegardlessOfFit
};
```

Constants

`kWindowConstrainMayResize`

The window may be resized if necessary to make it fit onscreen.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowConstrainMoveRegardlessOfFit`
The window may be moved even if it doesn't fit entirely onscreen.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowConstrainAllowPartial`
Allow partial intersection of the specified window region with the screen instead of requiring total intersection.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowConstrainCalcOnly`
Calculate the new window bounds but don't actually move the window.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowConstrainUseTransitionWindow`
Use `TransitionWindow` (page 162) with `kWindowSlideTransitionEffect` to move windows onscreen.

Available in Mac OS X v10.2 and later.

Declared in `MacWindows.h`.

`kWindowConstrainMoveMinimum`
Move the window the minimum amount necessary to be onscreen. This option is only supported by the function `HIWindowConstrain` (page 89). This option applies if a partial fit is not allowed (`kWindowConstrainAllowPartial` is not specified) or a partial fit is allowed, but the window is not even partially visible. In either case, the window will be moved just enough to be slightly onscreen. You may customize the minimum amount that is required to be visible by passing the desired dimensions in the `inMinimumSize` parameter to `HIWindowConstrain`.

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kWindowConstrainUseSpecifiedBounds`
Use the specified bounds of the window region to be constrained. This option is only supported by the function `HIWindowConstrain` (page 89). The bounds are specified using the `ioBounds` parameter, allowing you to constrain a window to a hypothetical location. For example, if you plan to move your window such that its content region is at a certain location, and you want to know in advance before moving the window whether the window would be offscreen at that location, you can use this option.

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kWindowConstrainStandardOptions`
Use the most common options: don't resize the window, move the window regardless of fit to the screen, require total intersection of the specified window region with the screen, and move the window.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

Window Kinds

Identify how a window was created.

```
enum {
    dialogKind = 2,
    userKind = 8,
    kDialogWindowKind = 2,
    kApplicationWindowKind = 8
};
```

Constants

`dialogKind`
Obsolete equivalent to `kDialogWindowKind`.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`userKind`
Obsolete equivalent to `kApplicationWindowKind`.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kDialogWindowKind`
Identifies all dialog or alert windows, whether created by system software or, indirectly through the Dialog Manager, by your application. The Dialog Manager uses this field to track dialog and alert windows.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kApplicationWindowKind`
Identifies a window created directly by your application.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

Discussion

The Window Manager uses these constants in the `windowKind` field of a color window structure or window structure. Your application can use any value greater than 7.

Window Group Selection Constants

Indicate which window group to select.

```
enum {
    kNextWindowGroup = true,
    kPreviousWindowGroup = false
};
```

Constants

`kNextWindowGroup`
Move to the next window group (in the z-order).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kPreviousWindowGroup`
Move to the previous window group (in the z-order).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

Window Group Attributes

Define attributes for window groups.

```
typedef UInt32 WindowGroupAttributes;
enum {
    kWindowGroupAttrSelectAsLayer = 1 << 0,
    kWindowGroupAttrMoveTogether = 1 << 1,
    kWindowGroupAttrLayerTogether = 1 << 2,
    kWindowGroupAttrSharedActivation = 1 << 3,
    kWindowGroupAttrHideOnCollapse = 1 << 4,
    kWindowGroupAttrFixedLevel = 1 << 5
};
```

Constants

`kWindowGroupAttrSelectAsLayer`

Makes the group behave somewhat as a layer of windows that move together. When any window in the group is brought to the front of the group, the entire group will also be brought to the front of the containing group's child hierarchy. Use of this constant is not recommended; its behavior is rarely useful.

Available in Mac OS X v10.1 and later.

Declared in `MacWindows.h`.

`kWindowGroupAttrMoveTogether`

The positions of the contents of this group with respect to each other cannot be changed. When one item moves, all other items are moved simultaneously. Note that if one window's position is changed by calling a Window Manager function in Mac OS X v10.4 and later, the positions of the other windows in the group are updated asynchronously—that is, their bounds are not necessarily updated during the function call itself, even though visually the windows move together.

Available in Mac OS X v10.1 and later.

Declared in `MacWindows.h`.

`kWindowGroupAttrLayerTogether`

The z-order of the contents of this group with respect to each other cannot be changed. When one item changes z-order, all other items are moved simultaneously. For purposes of z-ordering, the group and all its subgroups are effectively treated as if they were a single window in the parent group of this group.

Available in Mac OS X v10.1 and later.

Declared in `MacWindows.h`.

`kWindowGroupAttrSharedActivation`

The active state of the windows in this group is shared. The windows in the group are activated or deactivated according to the activation scope of the group, but when any window in the group changes activation, all other windows change to match.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowGroupAttrHideOnCollapse`

When any window in this group is collapsed, all other windows in this group are hidden. All subgroups of this group are also examined for this attribute, and any the windows of any subgroup with this attribute are also hidden. All windows will be shown again when the collapsed window is expanded.

Available in Mac OS X v10.1 and later.

Declared in `MacWindows.h`.

`kWindowGroupAttrFixedLevel`
If this attribute is specified, this window group's window level should be left unchanged. If this attribute is not specified, this window group's window level will be promoted to a value equal to the level of the next fixed-level window group beneath it in the window group hierarchy. (Available in Mac OS X v10.4 and later.)

Available in Mac OS X v10.4 and later.

Declared in `MacWindows.h`.

Version Notes
In Mac OS X v10.2.4 and later, the HIToolbox framework improved its use of the window group API so that showing a sheet on a window that was already grouped with another window would not break the existing grouping. To make this change work properly, applications that create their own window groups using the `kWindowGroupAttrMoveTogether` and `kWindowGroupAttrLayerTogether` attributes should also specify the `kWindowGroupAttrHideOnCollapse` and `kWindowGroupAttrSharedActivation` attributes.

Obsolete Window Group Attributes

Define obsolete window group attribute names.

```
enum {
    kWindowGroupAttrSelectable = kWindowGroupAttrSelectAsLayer,
    kWindowGroupAttrPositionFixed = kWindowGroupAttrMoveTogether,
    kWindowGroupAttrZOrderFixed = kWindowGroupAttrLayerTogether
};
```

Constants
`kWindowGroupAttrSelectable`
Obsolete name for `kWindowGroupAttrSelectAsLayer`.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowGroupAttrPositionFixed`
Obsolete name; use `kWindowGroupAttrMoveTogether` instead.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowGroupAttrZOrderFixed`
Obsolete name; use `kWindowGroupAttrLayerTogether` instead.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

Window Group Content Options

Window group counting options.

```
typedef UInt32 WindowGroupContentOptions;
enum {
    kWindowGroupContentsReturnWindows = 1 << 0,
    kWindowGroupContentsRecurse = 1 << 1,
    kWindowGroupContentsVisible = 1 << 2
};
```

Constants

`kWindowGroupContentsReturnWindows`
Count only windows in the window group.
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

`kWindowGroupContentsRecurse`
Recursively count windows of any subgroups of windows in the specified window group.
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

`kWindowGroupContentsVisible`
Counts only visible windows.
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

Discussion
You use these constants with the `CountWindowGroupContents` (page 40) function.

Window Class Position Constants

Specify which window in the class to select.

```
enum {
    kFirstWindowOfClass = -1,
    kLastWindowOfClass = 0
};
```

Constants

`kFirstWindowOfClass`
Select the first window in the class.

`kLastWindowOfClass`
Select the last window in the class.

Discussion
These constants describe special cases for the “behind” parameter in window creation calls.

Window Definition Type Constants

Defines the type of custom window definition.

```
typedef UInt32 WindowDefType;
enum {
    kWindowDefProcPtr = 0,
    kWindowDefObjectClass = 1,
    kWindowDefProcID = 2,
    kWindowDefHIView = 3
};
```

Constants

`kWindowDefProcPtr`
The definition is procedure pointer–based.
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

`kWindowDefObjectClass`
The definition is a toolbox object.
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

`kWindowDefProcID`
An ID that identifies a particular `'WDEF'` and would typically be one of the constants described in "Appearance-Compliant Window Definition ID Constants" (page 200).
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

`kWindowDefHIView`
The definition is an HIView-based object.
Available in Mac OS X v10.2 and later.
Declared in `MacWindows.h`.

Window Definition Procedure Constant

Define the window definition procedure constant.

```
enum {
    kWindowDefProcType = 'WDEF'
};
```

Constants

`kWindowDefProcType`
Window definition type.
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

Window Definition Hit Test Result Code Constants

Defines result constants to be used by window definition hit testing.

```
typedef SInt16 WindowDefPartCode;
enum {
    wNoHit = 0,
    wInContent = 1,
    wInDrag = 2,
    wInGrow = 3,
    wInGoAway = 4,
    wInZoomIn = 5,
    wInZoomOut = 6,
    wInCollapseBox = 9,
    wInProxyIcon = 10,
    wInToolbarButton = 11,
    wInStructure = 13
};
```

Constants

`wNoHit`

The mouse-down event did not occur in the content region or the drag region of any active or inactive window or in the close, size, zoom, or collapse box of an active window. The return value `wNoHit` might also mean that the point isn't in the window. The standard window definition functions, for example, return `wNoHit` if the point is in the window frame but not in the title bar.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`wInContent`

The mouse-down event occurred in the content region of an active or inactive window (with the exception of the size box).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`wInDrag`

The mouse-down event occurred in the drag region of an active or inactive window.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`wInGrow`

The mouse-down occurred in the size box of an active window.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`wInGoAway`

The mouse-down event occurred in the close box of an active window.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`wInZoomIn`

The mouse-down event occurred in the zoom box of an active window that is currently in the standard state.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`wInZoomOut`
: The mouse-down event occurred in the zoom box of an active window that is currently in the user state.

  Available in Mac OS X v10.0 and later.

  Declared in `MacWindows.h`.

`wInCollapseBox`
: The mouse-down event occurred in the collapse box of an active window.

  Available with Appearance Manager 1.0 and later.

  Declared in `MacWindows.h`.

`wInProxyIcon`
: The mouse-down event occurred in the proxy icon of a window.

  Available in Mac OS X v10.0 and later.

  Declared in `MacWindows.h`.

`wInToolbarButton`
: The mouse-down event occurred in the toolbar button.

  Available in Mac OS X v10.1 and later.

  Declared in `MacWindows.h`.

`wInStructure`
: The mouse-down event occurred in the window's structure region.

  Available in Mac OS X v10.1 and later.

  Declared in `MacWindows.h`.

Discussion
In response to the `wHit` message, your window definition function should return one of these constants.

Return the constants `wInGrow`, `wInGoAway`, `wInZoomIn`, `wInZoomOut`, and `wInCollapseBox` only if the window is active—by convention, the size box, close box, zoom box, and collapse box aren't drawn if the window is inactive. In an inactive document window, for example, a mouse-down event in the part of the title bar that would contain the close box if the window were active is reported as `wInDrag`.

With the Mac OS 8.5 Window Manager and later, your window definition function may return the `wInProxyIcon` constant to report that a mouse-down event occurred in your window's proxy icon.

Window Definition Message Constants

Defines messages sent to non Carbon Event–based window definitions.

```
enum {
    kWindowMsgDraw = 0,
    kWindowMsgHitTest = 1,
    kWindowMsgCalculateShape = 2,
    kWindowMsgInitialize = 3,
    kWindowMsgCleanUp = 4,
    kWindowMsgDrawGrowOutline = 5,
    kWindowMsgDrawGrowBox = 6
};
enum {
    kWindowMsgGetFeatures = 7,
    kWindowMsgGetRegion = 8
};
enum {
    kWindowMsgDragHilite = 9,
    kWindowMsgModified = 10,
    kWindowMsgDrawInCurrentPort = 11,
    kWindowMsgSetupProxyDragImage = 12,
    kWindowMsgStateChanged = 13,
    kWindowMsgMeasureTitle = 14
};
enum {
    kWindowMsgGetGrowImageRegion = 19
};
```

Constants

`kWindowMsgDraw`
Draw the window's frame.
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

`kWindowMsgHitTest`
Report the location of a mouse-down event.
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

`kWindowMsgCalculateShape`
Calculate the structure region and the content region.
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

`kWindowMsgInitialize`
Perform additional initialization.
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

`kWindowMsgCleanUp`
Perform additional disposal.
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

`kWindowMsgDrawGrowOutline`
Draw the dotted outline of the window that you see during a resizing operation.
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

`kWindowMsgDrawGrowBox`
Draw the outlines for the size box and the scroll bar.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowMsgGetFeatures`
Report the window's features.

Available with Appearance Manager 1.0 and later.

Declared in `MacWindows.h`.

`kWindowMsgGetRegion`
Report the location of a specific window region.

Available with Appearance Manager 1.0 and later.

Declared in `MacWindows.h`.

`kWindowMsgDragHilite`
Redraw the window's structure region to reflect the window's validity as a drag-and-drop destination. The Window Manager passes an accompanying Boolean value in your window definition function's `param` parameter. If the value passed is `true`, this indicates that the window's structure region should be highlighted. If the value passed is `false`, the structure region should be unhighlighted. Your window definition function should return 0 as the function result. (Mac OS 8.5 and later.)

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowMsgModified`
Track the window's modification state. The Window Manager sends this message when the function `SetWindowModified` (page 146) is called. The Window Manager passes an accompanying Boolean value in your window definition function's `param` parameter. If the value passed is `true`, the document contained in the window has been modified. If the value passed is `false`, the document has been saved to disk. You should redraw the window's structure region to reflect the new modification state, if appropriate. For example, system-defined document windows dim the proxy icon to indicate that the document has been modified by the user and cannot be moved at that time. Your window definition function should return 0 as the function result. (Mac OS 8.5 and later.)

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowMsgDrawInCurrentPort`
Draw the window's frame in the current graphics port. Other than restricting drawing to the current port, this message is similar to the pre–Mac OS 8.5 Window Manager window definition message constant `wDraw`. (Mac OS 8.5 and later.)

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowMsgSetupProxyDragImage`

Create the image of the window's proxy icon that the Drag Manager uses to represent the icon while it is being dragged. When your application calls the function `TrackWindowProxyDrag` (page 159), the Window Manager passes this message in your window definition function's `message` parameter and an accompanying pointer to a structure of type `SetupWindowProxyDragImageRec` (page 180) in the `param` parameter. Your window definition function is responsible for setting the contents of the structure to contain the data describing the proxy icon's drag image. Your window definition function should return 0 as the function result. (Mac OS 8.5 and later.)

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowMsgStateChanged`

Be informed that some aspect of the window's public state has changed. The Window Manager passes this message in your window definition function's `message` parameter and an accompanying flag in the `param` parameter that indicates what part of the window's state has been altered. This message is simply a notification message—no response by the window definition function is required. Your window definition function should return 0 as the function result. The `kWindowMsgStateChanged` message is sent after the window's internal data has been updated, but before any redraw occurs onscreen. A window definition function should not redraw the window frame in response to this message. If it is necessary to redraw the window frame, the Window Manager notifies the window definition function with a `wDraw` message. See "Window Definition State-Changed Constant" (page 235) for descriptions of the values that the Window Manager can pass to specify the state change that has occurred. (Mac OS 8.5 and later.)

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowMsgMeasureTitle`

Measure and return the ideal title width. The Window Manager passes this message in the window definition function's `message` parameter and an accompanying pointer to a structure of type `MeasureWindowTitleRec` (page 178) in the `param` parameter. Your window definition function is responsible for setting the contents of the structure to contain data describing the title width. You should return 0 as the function result. (Mac OS 8.5 and later.)

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kWindowMsgGetGrowImageRegion`

Obtain a region to XOR with window during grow or resize. Alter the `GetGrowImageRegionRec` structure passed with the message to the region to be XOR'd. (

Available in Carbon only.)

Declared in `MacWindows.h`.

Discussion

The Window Manager may pass one of these constants in the `message` parameter of your window definition function to specify the action that your function must perform. For descriptions of the feature bits that correspond to these messages, see "Window Feature Bits" (page 207). Other messages are reserved for internal use by the system.

Window Definition State-Changed Constant

Define the window definition state-changed constant.

```
enum {
    kWindowStateTitleChanged = (1 << 0)
};
```

Constants

`kWindowStateTitleChanged`

If the bit specified by this mask is set, the window's title has changed.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

Discussion

If you implement a custom window definition function, when the Window Manager passes the `kWindowMsgStateChanged` message in your window definition function's `message` parameter it may also pass a value in the `param` parameter with one or more bits set to indicate what part of the window's state has changed. You may use this mask to test this value. For a description of the `kWindowMsgStateChanged` message, see "Window Definition Message Constants" (page 232).

Special Considerations

Drawer State Constants

Define constants that indicate the current drawer state.

```
typedef UInt32 WindowDrawerState;
enum {
    kWindowDrawerOpening = 1,
    kWindowDrawerOpen = 2,
    kWindowDrawerClosing = 3,
    kWindowDrawerClosed = 4
};
```

Constants

`kWindowDrawerOpening`

The drawer is opening.

Available in Mac OS X v10.2 and later.

Declared in `MacWindows.h`.

`kWindowDrawerOpen`

The drawer is open.

Available in Mac OS X v10.2 and later.

Declared in `MacWindows.h`.

`kWindowDrawerClosing`

The drawer is closing.

Available in Mac OS X v10.2 and later.

Declared in `MacWindows.h`.

`kWindowDrawerClosed`

The drawer is closed.

Available in Mac OS X v10.2 and later.

Declared in `MacWindows.h`.

Version Notes

Introduced in Mac OS X v10.2.

Window Edge Constants

Specify the edge from which a drawer should appear.

```
enum {
    kWindowEdgeDefault = 0,
    kWindowEdgeTop = 1 << 0,
    kWindowEdgeLeft = 1 << 1,
    kWindowEdgeBottom = 1 << 2,
    kWindowEdgeRight = 1 << 3
};
```

Constants

`kWindowEdgeDefault`
The drawer should be opened on whatever edge of the parent window has previously been set as the drawer's preferred edge.

Available in Mac OS X v10.2 and later.

Declared in `MacWindows.h`.

`kWindowEdgeTop`
The drawer should slide out from the top edge.

Available in Mac OS X v10.2 and later.

Declared in `MacWindows.h`.

`kWindowEdgeLeft`
The drawer should slide out from the left edge.

Available in Mac OS X v10.2 and later.

Declared in `MacWindows.h`.

`kWindowEdgeBottom`
The drawer should slide out from the bottom edge.

Available in Mac OS X v10.2 and later.

Declared in `MacWindows.h`.

`kWindowEdgeRight`
The drawer should slide out from the right edge.

Available in Mac OS X v10.2 and later.

Declared in `MacWindows.h`.

Rotating Window Menu Item Constant

Indicates whether to add the rotating window item to the Window menu.

```
enum {
    kWindowMenuIncludeRotate = 1 << 0
};
```

Constants

`kWindowMenuIncludeRotate`
Requests that the standard window menu include a Rotate Windows menu item.

Available in Mac OS X v10.2 and later.

Declared in `MacWindows.h`.

Discussion
This constant is used with the function `CreateStandardWindowMenu` (page 42).

Window Menu Item Property Constants

Constants used to access property data of items in the standard window menu.

```
enum {
    kHIWindowMenuCreator = 'wind',
    kHIWindowMenuWindowTag = 'wind'
};
```

Constants

`kHIWindowMenuCreator`
The property creator for accessing standard window menu item properties.

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

`kHIWindowMenuWindowTag`
The property tag for accessing standard window menu item properties that hold windows (values of type `WindowRef`). Menu items with the `kHICommandSelectWindow` command ID will have a property with this tag that contains the window to be activated when that item is selected.

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

Discussion
These constants are used with the Menu Manager functions `GetMenuItemProperty` and `SetMenuItemProperty`.

Toolbar View Background Tag

A tag used to inform a custom toolbar view whether to draw its background or leave its background transparent.

```
enum {
    kHIToolbarViewDrawBackgroundTag = 'back'
};
```

Constants

`kHIToolbarViewDrawBackgroundTag`
A `SetControlData` tag that is used by the standard window frame view to inform the toolbar view whether the view should draw its background or leave its background transparent. The data for this tag is a Boolean. If the data value is true, the toolbar view should draw its background as it desires. If the data value is false, the toolbar view should leave its background transparent so that the window's root view can show through the toolbar view. Currently, the toolbar view will be asked to leave its background transparent for windows with the textured or unified appearance.

Available in Mac OS X v10.4 and later.

Declared in `MacWindows.h`.

Window Paint Callback Options

Define options to use with the window paint callback function.

```
typedef OptionBits WindowPaintProcOptions;
enum {
    kWindowPaintProcOptionsNone = 0
};
```

Constants

`kWindowPaintProcOptionsNone`
No options.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

Part Identifier Constants

Used in the `value` field of the `ColorSpec` structure, define which part of the window the color affects.

```
enum {
    wContentColor = 0,
    wFrameColor = 1,
    wTextColor = 2,
    wHiliteColor = 3,
    wTitleBarColor = 4
};
```

Constants

`wContentColor`
Produces background color for content region of window.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`wFrameColor`
Produces color of window's outline.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`wTextColor`
Produces color of window's title and button text.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`wHiliteColor`
Reserved.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`wTitleBarColor`
Reserved.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

Discussion
When the Appearance Manager is available and you are using standard windows, all the fields of the window color table structure are ignored except the part identifier constant `wContentColor` in the `value` field of the `ColorSpec` structure, which produces the background color for the window's content region.

If you are creating your own custom windows, the window color table structure and all its part identifier constants can still be used.

Desk Pattern Resource ID

The resource ID of the desktop pattern.

```
enum {
    deskPatID = 16
};
```

Constants
`deskPatID`
The resource ID of the desktop pattern.
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

Discussion
The Window Manager provides the desk pattern resource ID constants, which is the ID of `Pattern` and `PixPat` resources that the operating system uses to draw the desktop. The operating system uses the `deskPatID` constant while the desktop is being drawn. It looks for a resource with this ID and uses the contents of the resource to draw the desktop.

Window Scrolling Options

Options for scrolling windows.

```
typedef UInt32 ScrollWindowOptions;
enum {
    kScrollWindowNoOptions = 0,
    kScrollWindowInvalidate = (1L << 0),
    kScrollWindowEraseToPortBackground = (1L << 1)
};
```

Constants
`kScrollWindowNoOptions`
No options.
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

`kScrollWindowInvalidate`
Add the exposed area to the window's update region.
Available in Mac OS X v10.0 and later.
Declared in `MacWindows.h`.

`kScrollWindowEraseToPortBackground`
Erase the exposed area using the background color/pattern of the window's graphics port.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

Discussion
Use these constants with the `ScrollWindowRect` (page 127) and `ScrollWindowRegion` (page 128) functions.

Availability
Available in Mac OS 8.1 and later.

'wind' Resource Default Collection Item Constants

Specify default collection items in a window (`'wind'`) resource.

```
enum {
    kStoredWindowSystemTag = 'appl',
    kStoredBasicWindowDescriptionID = 'sbas',
    kStoredWindowPascalTitleID = 's255',
    kStoredWindowTitleCFStringID = 'cfst'
};
```

Constants
`kStoredWindowSystemTag`
This item tag specifies a system-defined collection item. Note that the `'appl'` collection item tag is reserved for use by Apple Computer, Inc. Do not define new collection items using this tag.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kStoredBasicWindowDescriptionID`
In combination with `kStoredWindowSystemTag`, this item ID specifies an item of type `BasicWindowDescription`. See `BasicWindowDescription` (page 175) for details on this type.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kStoredWindowPascalTitleID`
In combination with `kStoredWindowSystemTag`, this item ID specifies a Pascal title string.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kStoredWindowTitleCFStringID`
This item tag specifies the CFString title string.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

Discussion
These constants specify the tag and the IDs that identify the default collection items contained in a resource of type `'wind'`.

Window Resource IDs

Define standard resource IDs for windows.

```
enum {
    kStandardWindowDefinition = 0,
    kRoundWindowDefinition = 1,
    kFloatingWindowDefinition = 124
};
```

Constants

`kStandardWindowDefinition`

Defines pre-Appearance standard document windows and dialog boxes. When mapping is enabled, this resource ID is mapped to `kWindowDocumentDefProcResID` or `kWindowDialogDefProcResID`. When mapped to `kWindowDocumentDefProcResID`, this produces an Appearance-compliant standard document window with no size box and no vertical or horizontal zoom box. When mapped to `kWindowDialogDefProcResID`, this produces an Appearance-compliant dialog box with no size box and a 3-pixel space between the dialog box's content and structure region.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kRoundWindowDefinition`

Defines pre-Appearance standard desk-accessory style windows. This resource ID is not mapped to any Appearance-compliant resource ID when mapping is enabled.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`kFloatingWindowDefinition`

Defines pre-Appearance utility (floating) windows. When mapping is enabled, this resource ID is mapped to `kWindowUtilityDefProcResID` or `kWindowUtilitySideTitleDefProcResID`. When mapped to `kWindowUtilityDefProcResID`, this produces an Appearance-compliant utility window. When mapped to `kWindowUtilitySideTitleDefProcResID`, it produces an Appearance-compliant utility window with a side title bar.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

Discussion

Window resource IDs are changed with Appearance Manager 1.0. The Window Manager now provides many new standard, Appearance-compliant window resource IDs for your program. For a description of the Appearance-compliant window resource IDs, see "Appearance-Compliant Window Resource IDs" (page 199).

You can use a window resource ID constant to create a window definition ID; see "Pre-Appearance Window Definition IDs" in *Window Manager Legacy Reference* for more details.

Resource IDs 0 through 127 are reserved for use by the system.

Window Availability Constants

Define window availability constants for Exposé and Spaces.

```
typedef OptionBits HIWindowAvailability;
enum {
    kHIWindowExposeHidden = 1 << 0,
    kHIWindowVisibleInAllSpaces = 1 << 8
};
```

Constants

`kHIWindowExposeHidden`

If this bit is set, the window is hidden during the "All Windows" and "Application windows" modes of Exposé. If this bit is not set, the window is visible during those modes.

Available in Mac OS X v10.4 and later.

Declared in `MacWindows.h`.

`kHIWindowVisibleInAllSpaces`

If this bit is set, the window is visible in all Spaces workspaces. If this bit is not set, the window is only visible in the workspace in which it was created.

Available in Mac OS X v10.5 and later.

Declared in `MacWindows.h`.

Discussion

These mask bits are used with the function `HIWindowChangeAvailability` (page 87) to override the default behavior of the Window Manager in determining whether a window is visible during Exposé or in all Spaces workspaces. By default, newly created windows of class `kDocumentWindowClass` are given an availability of `0` (meaning that they are available during Exposé), and windows from all other window classes are given an availability of `kHIWindowExposeHidden`.

Window Scale Mode Constants

Define window scale mode constants.

```
typedef UInt32 HIWindowScaleMode;
enum {
    kHIWindowScaleModeUnscaled = 0,
    kHIWindowScaleModeMagnified = 1,
    kHIWindowScaleModeFrameworkScaled = 2
};
```

Constants

`kHIWindowScaleModeUnscaled`

The window is not scaled at all because the display scale factor is 1.0.

Available in Mac OS X v10.4 and later.

Declared in `MacWindows.h`.

`kHIWindowScaleModeMagnified`

The window's backing store is being magnified by the Window Server because the display scale factor is not equal to 1.0 and because the window was not created with the `kWindowFrameworkScaledAttribute`.

Available in Mac OS X v10.4 and later.

Declared in `MacWindows.h`.

`kHIWindowScaleModeFrameworkScaled`
The window's contents are scaled to match the display scale factor because the display scale factor is not equal to 1.0 and because the window was created with `kWindowFrameworkScaledAttribute`.

Available in Mac OS X v10.4 and later.

Declared in `MacWindows.h`.

Discussion
A window's scale mode indicates in which resolution-independent scale mode it is operating.

Window Group Level Constants

Define window group level constants.

```
enum {
kWindowGroupLevelActive = 1,
kWindowGroupLevelInactive = 2,
kWindowGroupLevelPromoted = 3,
};
```

Constants
`kWindowGroupLevelActive`
The window level that is nominally used for windows in the group when the application is active. However, if a group with a higher window level is positioned below this group in the window group hierarchy, this group's active level will be promoted to match the level of the group in front of it. Use `kWindowGroupLevelPromoted` to determine the actual window level in use for a group.

Available in Mac OS X v10.4 and later.

Declared in `MacWindows.h`.

`kWindowGroupLevelInactive`
The window level for windows in the group when the application is inactive.

Available in Mac OS X v10.4 and later.

Declared in `MacWindows.h`.

`kWindowGroupLevelPromoted`
The window level that is actually used for windows in the group when the application is active. This level is the same as the Active window level or is a larger value to match the level of a group below this group. Setting the promoted window level explicitly is not recommended because the promoted level is reset by the Window Manager whenever the window group hierarchy structure changes. Therefore any changes that you make to the promoted level can be overwritten.

Available in Mac OS X v10.4 and later.

Declared in `MacWindows.h`.

Discussion
These constants are used when calling `GetWindowGroupLevelOfType` (page 68) and `SetWindowGroupLevelOfType` (page 142).

Pre-Appearance Window Definition IDs

Older window definition IDs used before the introduction of the Appearance Manager.

```
enum {
    documentProc = 0,
    dBoxProc = 1,
    plainDBox = 2,
    altDBoxProc = 3,
    noGrowDocProc = 4,
    movableDBoxProc = 5,
    zoomDocProc = 8,
    zoomNoGrow = 12,
    rDocProc = 16,
    floatProc = 1985,
    floatGrowProc = 1987,
    floatZoomProc = 1989,
    floatZoomGrowProc = 1991,
    floatSideProc = 1993,
    floatSideGrowProc = 1995,
    floatSideZoomProc = 1997,
    floatSideZoomGrowProc = 1999
};
```

Constants

`documentProc`

Pre-Appearance document window (movable window with size box).

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`dBoxProc`

Pre-Appearance modal dialog box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`plainDBox`

Pre-Appearance modeless dialog box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`altDBoxProc`

Pre-Appearance modeless dialog box with shadow.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`noGrowDocProc`

Pre-Appearance movable window with no size box or zoom box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`movableDBoxProc`

Pre-Appearance movable modal dialog box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`zoomDocProc`
Pre-Appearance movable window with size box and full zoom box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`zoomNoGrow`
Pre-Appearance window with full zoom box and no size box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`rDocProc`
Pre-Appearance rounded-corner window. You can control the diameter of curvature of a rounded-corner window (window type `rDocProc`) by adding one of these integers to the `rDocProc` constant:

`rDocProc` (diameters of curvature: 16, 16)

`rDocProc` + 2 (diameters of curvature: 4, 4)

`rDocProc` + 4 (diameters of curvature: 6, 6)

`rDocProc` + 6 (diameters of curvature: 10, 10)

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared in `MacWindows.h`.

`floatProc`
Pre-Appearance utility (floating) window with no size box or zoom box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`floatGrowProc`
Pre-Appearance utility (floating) window with size box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`floatZoomProc`
Pre-Appearance utility (floating) window with zoom box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`floatZoomGrowProc`
Pre-Appearance utility (floating) window with size box and zoom box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`floatSideProc`
Pre-Appearance utility (floating) window with side title bar and no size or zoom box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`floatSideGrowProc`
Pre-Appearance utility (floating) window with side title bar and size box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`floatSideZoomProc`

Pre-Appearance utility (floating) window with side title bar and zoom box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

`floatSideZoomGrowProc`

Pre-Appearance utility (floating) window with side title bar, size box, and zoom box.

Available in Mac OS X v10.0 and later.

Declared in `MacWindows.h`.

Discussion

Note that window definition IDs are changed with Appearance Manager 1.0. The Window Manager now provides many new, standard, Appearance-compliant window types.

Your application typically supplies a window definition ID to a resource of type `'WIND'` or to a window-creation function to specify which window definition function to use in creating the window. A variation code may also be used to describe variations of the same basic window.

The window definition ID is an integer that contains the resource ID of the window definition function in its upper 12 bits and a variation code in its lower 4 bits. For a given resource ID and variation code, the window definition ID is derived as follows: window definition ID = (16 x resource ID) + variation code.

The window definition IDs for dialog boxes and utility (floating) windows pertain to the appearances of these windows only, not their behaviors. For example, if you want a utility window to have the proper behavior, that is, float, your application must provide for it.

When mapping is enabled, standard pre-Appearance window definition function IDs will be mapped to their Appearance-compliant equivalents.

Result Codes

The table below lists result codes defined for the Window Manager.

Result Code	Value	Description
`errInvalidWindowRef`	-5600	The window is not valid. Available in Mac OS X v10.0 and later.
`errUnsupportedWindowAttributesForClass`	-5601	Attribute bits are inappropriate for the specified window class. Available in Mac OS X v10.0 and later.
`errWindowDoesNotHaveProxy`	-5602	No proxy attached to window. Available in Mac OS X v10.0 and later.
`errInvalidWindowProperty`	-5603	`'appl'` creator code not allowed. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
errWindowPropertyNotFound	-5604	The window property does not exist. Available in Mac OS X v10.0 and later.
errUnrecognizedWindowClass	-5605	Unknown window class. Available in Mac OS X v10.0 and later.
errCorruptWindowDescription	-5606	Incorrect size or version supplied in the BasicWindowDescription structure. Available in Mac OS X v10.0 and later.
errUserWantsToDragWindow	-5607	Entire window is being dragged, not proxy icon. Available in Mac OS X v10.0 and later.
errWindowsAlreadyInitialized	-5608	Called InitFloatingWindows twice, or called InitWindows and then InitFloatingWindows. Available in Mac OS X v10.0 and later.
errFloatingWindowsNotInitialized	-5609	Called HideFloatingWindows or ShowFloatingWindows without calling InitFloatingWindows. Available in Mac OS X v10.0 and later.
errWindowNotFound	-5610	No window was found that satisfies the search criteria. Available in Mac OS X v10.0 and later.
errWindowDoesNotFitOnscreen	-5611	The window does not fit on a single screen. Available in Mac OS X v10.0 and later.
windowAttributeImmutableErr	-5612	Tried to change a window attribute that can't be changed after the window is created. Available in Mac OS X v10.0 and later.
windowAttributesConflictErr	-5613	Passed two window attributes that are mutually exclusive. Available in Mac OS X v10.0 and later.
windowManagerInternalErr	-5614	Internal error in the Window Manager. Available in Mac OS X v10.0 and later.
windowWrongStateErr	-5615	The window state makes the current action invalid. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
windowGroupInvalidErr	-5616	The window group is not valid. Available in Mac OS X v10.0 and later.
windowAppModalStateAlreadyExistsErr	-5617	The window is already application modal. Available in Mac OS X v10.1 and later.
windowNoAppModalStateErr	-5618	The window is not currently application modal. Available in Mac OS X v10.1 and later.
errWindowDoesntSupportFocus	-30583	Not used. Available in Mac OS X v10.0 and later.
errWindowRegionCodeInvalid	-30593	The window region code is not valid. Available in Mac OS X v10.0 and later.

Deprecated Window Manager Functions

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

Deprecated in Mac OS X v10.5

CalcVis

Calculates the visible region of a window. (Deprecated in Mac OS X v10.5. There is no replacement function.)

```
void CalcVis (
   WindowRef window
);
```

Parameters

window

On input, a pointer to the window's complete window structure.

Discussion

The Window Manager calls the `CalcVis` function; your application does not normally need to. `CalcVis` calculates the visible region of the specified window by starting with its content region and subtracting the structure region of each window in front of it.

Special Considerations

In Mac OS X, the visible region of a window is managed by the window server. Applications never need to call this function.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

CalcVisBehind

Calculates the visible regions of a series of windows. (Deprecated in Mac OS X v10.5. There is no replacement function.)

```
void CalcVisBehind (
   WindowRef startWindow,
   RgnHandle clobberedRgn
);
```

Parameters

startWindow

On input, a pointer to a window structure.

clobberedRgn

On input, a handle to the desktop region that has become invalid.

Discussion

The Window Manager calls the `CalcVisBehind` function; your application does not normally need to. `CalcVisBehind` calculates the visible regions of the window specified by the `startWindow` parameter and all windows behind `startWindow` that intersect `clobberedRgn`. It is called after `PaintBehind`.

Special Considerations

In Mac OS X, the visible region of a window is managed by the window server. Applications never need to call this function.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

CheckUpdate

Scans the window list for windows that need updating. (Deprecated in Mac OS X v10.5. Use `FindSpecificEventInQueue` or `AcquireFirstMatchingEventInQueue` instead.)

```
Boolean CheckUpdate (
   EventRecord *theEvent
);
```

Parameters

theEvent

On input, a pointer to an event structure to be filled in if a window needs updating.

Return Value

A Boolean value. If `CheckUpdate` finds a window structure whose update region is not empty and whose window structure does not contain a picture handle, it stores an update event in the event structure referenced through the parameter `theEvent` and returns `true`. If it finds no such window, it returns `false`.

Discussion

The Event Manager calls the `CheckUpdate` function; your application does not normally need to. `CheckUpdate` scans the window list from front to back, checking for a visible window that needs updating (that is, a visible window whose update region is not empty). If it finds one whose window structure contains a picture handle, it redraws the window itself and continues through the list.

Special Considerations

If you are using a compositing window, the Window Manager never generates update events for the window and you will never find update events in the event queue.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

ClipAbove

Determines the clip region of the Window Manager port. (Deprecated in Mac OS X v10.5. There is no replacement function.)

```
void ClipAbove (
   WindowRef window
);
```

Parameters

window

On input, a pointer to the window.

Discussion

The Window Manager calls the `ClipAbove` function; your application does not normally need to. `ClipAbove` sets the clip region of the Window Manager port to be the area of the desktop that intersects the current clip region, minus the structure regions of all the windows in front of the specified window.

`ClipAbove` retrieves the desktop region from the global variable `GrayRgn`.

Special Considerations

Mac OS X applications never need to call this function.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

CloneWindow

Increments the number of references to a window. (Deprecated in Mac OS X v10.5. Use `CFRetain` instead.)

```
OSStatus CloneWindow (
   WindowRef window
);
```

Parameters

window

The window whose reference count is to be incremented.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Discussion
You should call `CloneWindow` if you are using a window and want to ensure that it is not disposed while you are using it. With the Window Manager in Mac OS 8.5 and later, all windows are created with a reference count (owner count) of one. The function `CloneWindow` increments the number of references to a window, and the earlier function `DisposeWindow` decrements the number of references. When the reference count reaches zero, `DisposeWindow` disposes of the window.

In Mac OS X v10.2 and later, you can also call `CFRetain` to increment the reference count of a window.

Special Considerations

To maintain an accurate reference count, you must follow every call to the `CloneWindow` function with a matching call to the `DisposeWindow` function when your application is ready to release its reference to the window.

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

Declared In
`MacWindows.h`

CreateQDContextForCollapsedWindowDockTile

Obtains a `CGrafPtr` for a collapsed window's tile in the dock. (Deprecated in Mac OS X v10.5. Use `HIWindowCreateCollapsedDockTileContext` (page 93) instead.)

```
OSStatus CreateQDContextForCollapsedWindowDockTile (
   WindowRef inWindow,
   CGrafPtr *outContext
);
```

Parameters

inWindow

The window whose `CGrafPtr` is to be obtained.

outContext

On output, a pointer to the window's `CGrafPtr`.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

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

Deprecated in Mac OS X v10.5.
Not available to 64-bit applications.

Related Sample Code
QTCarbonShell

Declared In
`MacWindows.h`

CreateWindowFromCollection

Creates a window from collection data. (Deprecated in Mac OS X v10.5. Use `HIArchiveCopyDecodedCFType` to decode a window from an archive instead.)

```
OSStatus CreateWindowFromCollection (
   Collection collection,
   WindowRef *outWindow
);
```

Parameters

`collection`

A reference to the collection to be used in creating the window. You pass a reference to a previously created collection, such as that returned by the Collection Manager function `NewCollection`. The collection used to create the window must contain the required items for a resource of type `'wind'` or window creation fails.

`outWindow`

On input, a pointer to a value of type `WindowRef`. On return, the window pointer points to the newly created window.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Discussion
This function creates a window invisibly and places it at the front of the window's window group. After calling `CreateWindowFromCollection`, you should set any desired associated data—using Window Manager or Control Manager accessor functions—then call the function `TransitionWindow` (page 162) or `ShowWindow` (page 156) to display the window. The number of references to the collection (that is, its owner count) is incremented by a minimum of one for the duration of this call.

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

Declared In
`MacWindows.h`

CreateWindowFromResource

Creates a window from `'wind'` resource data. (Deprecated in Mac OS X v10.5. Use nib files and `CreateWindowFromNib` instead.)

```
OSStatus CreateWindowFromResource (
   SInt16 resID,
   WindowRef *outWindow
);
```

Parameters

resID

The resource ID of a resource of type ‘wind’. Pass in the ID of the 'wind' resource to be used to create the window.

outWindow

On input, a pointer to a value of type WindowRef. On return, the window pointer points to the newly created window.

Return Value

A result code. See “Window Manager Result Codes” (page 247).

Discussion

The CreateWindowFromResource function loads a window from a 'wind' resource. The Window Manager creates the window invisibly and places it at the front of the window’s window group. After calling CreateWindowFromResource, you should set any desired associated data—using Window Manager or Control Manager accessor functions—then call the function TransitionWindow (page 162) or ShowWindow (page 156) to display the window.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

MacWindows.h

DisposeWindowDefUPP

Disposes of the UPP for your window definition. (Deprecated in Mac OS X v10.5. The WDEF interface is deprecated; use a custom HIView to draw your custom window frame instead.)

```
void DisposeWindowDefUPP (
   WindowDefUPP userUPP
);
```

Parameters

userUPP

The UPP that is to be disposed of.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

MacWindows.h

DisposeWindowPaintUPP

Disposes of the UPP to your region painting callback function. (Deprecated in Mac OS X v10.5. The window content painting interface is deprecated; use a `kEventControlDraw` Carbon event handler on a compositing window's content view instead.)

```
void DisposeWindowPaintUPP (
   WindowPaintUPP userUPP
);
```

Parameters

userUPP

The UPP that is to be disposed of.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`MacWindows.h`

DragGrayRgn

Moves a gray outline of a region on the screen, following the movements of the cursor, until the mouse button is released. (Deprecated in Mac OS X v10.5. Use an overlay window or other custom drawing instead.)

```
long DragGrayRgn (
   RgnHandle theRgn,
   Point startPt,
   const Rect *limitRect,
   const Rect *slopRect,
   short axis,
   DragGrayRgnUPP actionProc
);
```

Parameters

theRgn

On input, a handle to the region to be dragged.

startPt

On input, the location, in the local coordinates of the current graphics port, of the cursor when the mouse button was pressed.

limitRect

On input, a pointer to a rectangle, given in the local coordinates of the current graphics port, that limits where the region can be dragged. This parameter works with the `slopRect` parameter.

slopRect

On input, a pointer to a rectangle, given in the local coordinates of the current graphics port, that gives the user some leeway in moving the mouse without violating the limits of the `limitRect` parameter. The `slopRect` rectangle should be larger than the `limitRect` rectangle.

axis

On input, a constant that constrains the region's motion. The `axis` parameter can have one of three values: `noConstraint` (0), `hAxisOnly` (1), or `vAxisOnly` (2).

If an axis constraint is in effect, the outline follows the cursor's movements along only the specified axis, ignoring motion along the other axis. With or without an axis constraint, the outline appears only when the mouse is inside the `slopRect` rectangle.

actionProc

On input, a pointer to a function that defines an action to be performed repeatedly as long as the user holds down the mouse button. The function can have no parameters. If the value of `actionProc` is `null`, `DragGrayRgn` simply retains control until the mouse button is released.

Return Value
A long integer that specifies the difference between the point where the mouse button was pressed and the offset point.

Discussion
The `DragGrayRgn` function is called by `DragWindow` to move an outline of a window around the screen as the user drags a window. It returns the difference between the point where the mouse button was pressed and the offset point (the point in the region whose horizontal and vertical offsets from the upper-left corner of the region's enclosing rectangle are the same as the offsets of the starting point when the user pressed the mouse button. `DragGrayRgn` stores the vertical difference between the starting point and the offset point in the high-order word of the return value and the horizontal difference in the low-order word.

It limits the movement of the region according to constraints set by the `limitRect` and `slopRect` parameters:

- As long as the cursor is inside the `limitRect` rectangle, the region's outline follows it normally. If the mouse button is released while the cursor is within this rectangle, the return value reflects the simple distance that the cursor moved in each dimension.
- When the cursor moves outside the `limitRect` rectangle, the offset point stops at the edge of the `limitRect` rectangle. If the mouse button is released while the cursor is outside the `limitRect` rectangle but inside the `slopRect` rectangle, the return value reflects only the difference between the starting point and the offset point, regardless of how far outside of the `limitRect` rectangle the cursor may have moved. (Note that part of the region can fall outside the `limitRect` rectangle, but not the offset point.)
- When the cursor moves outside the `slopRect` rectangle, the region's outline disappears from the screen. The `DragGrayRgn` function continues to track the cursor, however, and if the cursor moves back into the `slopRect` rectangle, the outline reappears. If the mouse button is released while the cursor is outside the `slopRect` rectangle, both words of the return value are set to `0x8000`. In this case, the Window Manager does not move the window from its original location.
- To accommodate systems with multiple monitors, QuickDraw recognizes a port rectangle of `screenBits.bounds` as a special case and allows drawing on all parts of the desktop.

The region stops moving when the offset point reaches the edge of the `limitRect` rectangle. The cursor continues to move, but the region does not.

If the mouse button is released while the cursor is anywhere inside the `slopRect` rectangle, the Window Manager redraws the window in its new location, which is calculated from the value returned by `DragGrayRgn`.

Carbon Porting Notes

Can't be used for live dragging. If you are implementing your own window dragging, use `DragWindow` instead. If you need to override window positioning during a drag, register a Carbon event handler for `kEventWindowBoundsChanging`. Okay to use if you're dragging objects within a window.

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

Declared In
`MacWindows.h`

DragTheRgn

Tracks the mouse as the user drags the outline of a region. (Deprecated in Mac OS X v10.5. Use an overlay window or other custom drawing instead.)

```
long DragTheRgn (
   RgnHandle theRgn,
   Point startPt,
   const Rect *limitRect,
   const Rect *slopRect,
   short axis,
   DragGrayRgnUPP actionProc
);
```

Carbon Porting Notes

Can't be used for live dragging. If you are implementing your own window dragging, use `DragWindow` instead. If you need to override window positioning during a drag, register a Carbon event handler for `kEventWindowBoundsChanging`. Okay to use if you're dragging objects within a window.

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

Declared In
`MacWindows.h`

DrawGrowIcon

Draws a grow icon in the window frame. (Deprecated in Mac OS X v10.5. There is no replacement function.)

```
void DrawGrowIcon (
    WindowRef window
);
```

Special Considerations

This function is not needed in Mac OS X. Theme-savvy windows include the grow box in the window frame.

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

Declared In
MacWindows.h

FrontWindow

Identifies the frontmost visible window. (Deprecated in Mac OS X v10.5. Use ActiveNonFloatingWindow (page 31), FrontNonFloatingWindow (page 50), or GetFrontWindowOfClass (page 55) instead.)

```
WindowRef FrontWindow (
   void
);
```

Return Value
The first visible window in the window list. If there are no visible windows, FrontWindow returns NULL.

Discussion
Most applications should call ActiveNonFloatingWindow (page 31) or FrontNonFloatingWindow (page 50) instead of FrontWindow because ActiveNonFloatingWindow and FrontNonFloatingWindow return the active and frontmost document window, respectively, skipping over other types of windows that may be in front of the active document, such as the menubar window, floating windows, help tags and toolbars.

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

Related Sample Code
HideMenuBar

Declared In
MacWindows.h

GetGrayRgn

Returns a region that covers the desktop area of all active displays. (Deprecated in Mac OS X v10.5. To determine the area in which a window may be positioned, use HIWindowGetAvailablePositioningBounds (page 96) or HIWindowCopyAvailablePositioningShape (page 90).)

```
RgnHandle GetGrayRgn (
    void
);
```

Return Value
A handle to the current desktop region from the global variable GrayRgn.

Discussion
When your application calls DragWindow to let the user drag a window, it can use GetGrayRgn to set the limiting rectangle to the entire desktop area. The desktop region represents all available screen space, that is, the desktop area displayed by all monitors attached to the computer.

Special Considerations

Your application should not modify the desktop region.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Related Sample Code

HideMenuBar

Declared In

`MacWindows.h`

GetNewCWindow

Creates a color window from a window resource. (Deprecated in Mac OS X v10.5. Use nib files and `CreateWindowFromNib` instead.)

```
WindowRef GetNewCWindow (
   short windowID,
   void *wStorage,
   WindowRef behind
);
```

Parameters

`windowID`

On input, the resource ID of the '`WIND`' resource that defines the properties of the window.

`wStorage`

On input, a pointer to memory space for the window structure. If you specify a value of `null` for `wStorage`, the `GetNewCWindow` function allocates the window structure as a nonrelocatable object in the heap. You can reduce the chances of heap fragmentation by allocating the memory your application needs for window structures early in your initialization code. Whenever you need to create a window, you can allocate memory from your own block and pass a pointer to it in the `wStorage` parameter.

`behind`

On input, a pointer to the window that appears immediately in front of the new window on the desktop. To place a new window in front of all other windows on the desktop, specify a value of (`WindowRef`)-1L. When you place a window in front of all others, `GetNewCWindow` removes the highlighting from the previously active window, highlights the newly created window, and generates the appropriate activate events. Note that if you create an invisible window in front of all others on the desktop, the user sees no active window until you make the new window visible (or make another window active).

To place a new window behind all other windows, specify a value of `null`.

Return Value

A pointer to the newly created window structure.

Discussion
The `GetNewCWindow` function creates a new color window from the specified window resource and returns a pointer to the newly created window structure. You can use the returned window pointer to refer to this window in most Window Manager functions. If `GetNewCWindow` is unable to read the window or window definition function from the resource file, it returns `null`.

The `GetNewCWindow` function looks for a 'wctb' resource with the same resource ID as that of the '`WIND `' resource. If it finds one, it uses the window color information in the '`wctb`' resource for coloring the window content area.

If the window's definition function (specified in the window resource) is not already in memory, `GetNewCWindow` reads it into memory and stores a handle to it in the window structure.

To create the window, `GetNewCWindow` retrieves the window characteristics from the window resource and then calls the `NewCWindow` function, passing the characteristics as parameters.

The `GetNewCWindow` function creates a window in a color graphics port. Your application typically sets up its own global variables reflecting the system setup during initialization by calling the `Gestalt` function.

Special Considerations
If you must get your window definition from a resource, use `CreateWindowFromResource`. Otherwise, use `CreateWindowFromNib` or `CreateNewWindow`.

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

Declared In
`MacWindows.h`

GetNewWindow

Creates a window from a window resource. (Deprecated in Mac OS X v10.5. Use nib files and `CreateWindowFromNib` instead.)

```
WindowRef GetNewWindow (
   short windowID,
   void *wStorage,
   WindowRef behind
);
```

Parameters
`windowID`
On input, the resource ID of the '`WIND`' resource that defines the properties of the window.
`wStorage`
On input, a pointer to memory space for the window structure. If you specify a value of `null` for `wStorage`, the `GetNewWindow` function allocates the window structure as a nonrelocatable object in the heap. You can reduce the chances of heap fragmentation by allocating the memory your application needs for window structures early in your initialization code. Whenever you need to create a window, you can allocate memory from your own block and pass a pointer to it in the `wStorage` parameter.

behind

On input, a pointer to the window that appears immediately in front of the new window on the desktop. To place a new window in front of all other windows on the desktop, specify a value of (WindowRef)-1l. When you place a window in front of all others, GetNewWindow removes the highlighting from the previously active window, highlights the newly created window, and generates the appropriate activate events. Note that if you create an invisible window in front of all others on the desktop, the user sees no active window until you make the new window visible (or make another window active). To place a new window behind all other windows, specify a value of null.

Return Value
A pointer to the newly created color window structure.

Discussion
The GetNewWindow function takes the same parameters as GetNewCWindow (page 261) and returns a value of type WindowRef. The only difference is that it creates a monochrome graphics port, not a color graphics port, regardless of the presence of a corresponding 'wctb ' resource (it loads the resource but doesn't use it). The window structure and graphics port structure that describe monochrome and color graphics ports are the same size and can be used interchangeably in most Window Manager functions.

The GetNewWindow function creates a new window from the specified window resource and returns a pointer to the newly created window structure. You can use the returned window pointer to refer to this window in most Window Manager functions. If GetNewWindow is unable to read the window or window definition function from the resource file, it returns null.

If the window's definition function (specified in the window resource) is not already in memory, GetNewWindow reads it into memory and stores a handle to it in the window structure. It allocates space in the application heap for the structure and content regions of the window.

To create the window, GetNewWindow retrieves the window characteristics from the window resource and then calls the function NewWindow, passing the characteristics as parameters.

Special Considerations

If you must get your window definition from a resource, use CreateWindowFromResource. Otherwise, use CreateWindowFromNib or CreateNewWindow.

Version Notes
The GetNewWindow function was originally implemented prior to Color QuickDraw. In Mac OS 8, you should call the Color QuickDraw function GetNewCWindow instead of GetNewWindow to programmatically create a window, because Color QuickDraw is always available in Mac OS 8. Use of this function is not recommended with Mac OS 8 and later. GetNewWindow is described here only for completeness.

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

Declared In
MacWindows.h

GetWindowOwnerCount

Obtains the number of existing references to a window. (Deprecated in Mac OS X v10.5. Use CFGetRetainCount instead.)

```
OSStatus GetWindowOwnerCount (
   WindowRef window,
   ItemCount *outCount
);
```

Parameters

window

The window whose reference (owner) count is to be determined.

outCount

A pointer to a value that, on return, contains the current number of references to the window.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Discussion

All windows are created with a reference count (owner count) of one. The function `CloneWindow` (page 253) increments the number of references to a window, and the earlier function `DisposeWindow` decrements the number of references. When the reference count reaches zero, `DisposeWindow` disposes of the window.

In Mac OS X v10.2 and later, you can also call `CFGetRetainCount` to get the number of existing references to a window.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

GetWindowPic

Returns a handle to a window's picture. (Deprecated in Mac OS X v10.5. Use an HIImageView object to draw a window's content and ask the view for its image instead.)

```
PicHandle GetWindowPic (
   WindowRef window
);
```

Parameters

window

The window whose picture handle is to be returned.

Return Value

A handle to the picture to be drawn in a specified window's content region. The handle must have been stored previously with the function `SetWindowPic` (page 281).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

GetWindowProxyFSSpec

Obtains a file system specification structure for the file that is associated with a window. (Deprecated in Mac OS X v10.5. Use `HIWindowGetProxyFSRef` (page 100) instead.)

```
OSStatus GetWindowProxyFSSpec (
   WindowRef window,
   FSSpec *outFile
);
```

Parameters

window

A pointer to the window for which you wish to determine the associated file.

outFile

On input, a pointer to an `FSSpec` structure. On return, this structure contains a copy of the file system specification data for the file associated with the specified window.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Discussion

You can use the `GetWindowProxyFSSpec` function to obtain identifying information about a proxy file: its volume reference number, directory ID, and file name.

See also the function `SetWindowProxyFSSpec`.

Special Considerations

The use of file specification structures is no longer recommended.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Related Sample Code

QTCarbonShell

Declared In

`MacWindows.h`

GetWindowRegion

Obtains a handle to a specific window region. (Deprecated in Mac OS X v10.5.)

```
OSStatus GetWindowRegion (
   WindowRef window,
   WindowRegionCode inRegionCode,
   RgnHandle ioWinRgn
);
```

Parameters

window

The window for which a window region handle is to be obtained.

inRegionCode

A constant representing the window region whose handle you want to obtain; see "Window Region Constants" (page 217) for a list of possible values.

ioWinRgn

On input, a handle to a region created by your application. On return, the handle is set to the specified window region.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Discussion
The `GetWindowRegion` function produces a handle to a window definition function's window region in response to a `kWindowMsgGetRegion` message. The visibility of the window is unimportant for `GetWindowRegion` to work correctly.

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

See Also
`HIWindowCopyShape` (page 91)

Declared In
`MacWindows.h`

GetWindowRetainCount

Returns the retain count of a window. (Deprecated in Mac OS X v10.5. Use `CFGetRetainCount` instead.)

```
ItemCount GetWindowRetainCount (
   WindowRef inWindow
);
```

Parameters

inWindow

The window whose retain count to retrieve.

Discussion
This API is equivalent to `GetWindowOwnerCount` (page 263). For consistency with Core Foundation and Carbon Events, it is preferred over `GetWindowOwnerCount`. Both APIs will continue to be supported.

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

Declared In
`MacWindows.h`

GetWTitle

Retrieves the title of a window as a Pascal string. (Deprecated in Mac OS X v10.5. Use `CopyWindowTitleAsCFString` (page 39) instead.)

```
void GetWTitle (
   WindowRef window,
   Str255 title
);
```

Parameters

window

On input, a pointer to the window structure.

title

A Pascal string. On output, the string contains the window title.

Discussion

The `GetWTitle` function returns the title of the window in the `title` parameter.

When you need to retrieve a window's title, you should always use `GetWTitle` instead of reading the title from a window structure.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

GetWVariant

Returns a window's variation code. (Deprecated in Mac OS X v10.5. Use `GetWindowAttributes` (page 59) to determine aspects of a window's appearance or behavior.)

```
short GetWVariant (
    WindowRef window
);
```

Parameters

window

On input, a pointer to the window structure.

Return Value

A short integer that specifies the variation code of the specified window. Depending on the window definition function, the result of `GetWVariant` can represent one of the standard variation codes or a variation code defined by your own window definition function.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

GrowWindow

Allows the user to change the size of a window. (Deprecated in Mac OS X v10.5. Use `ResizeWindow` (page 125) instead.)

```
long GrowWindow (
   WindowRef window,
   Point startPt,
   const Rect *bBox
);
```

Parameters

window

On input, a pointer to the window structure of the window to drag.

startPt

On input, the location of the cursor at the time the mouse button was first pressed, in global coordinates. Your application retrieves this point from the `where` field of the event structure.

bBox

On input, a pointer to a rectangle structure that specifies the limits on the vertical and horizontal measurements of the port rectangle, in pixels.

Although the `bBox` parameter gives the address of a structure which is in the form of the `Rect` data type, the four numbers in the structure represent lengths, not screen coordinates. The `top`, `left`, `bottom`, and `right` fields of the `bBox` parameter specify the minimum vertical measurement (`top`), the minimum horizontal measurement (`left`), the maximum vertical measurement (`bottom`), and the maximum horizontal measurement (`right`).

The minimum measurements must be large enough to allow a manageable rectangle 64 pixels on a side is typical. Because the user cannot ordinarily move the cursor off the screen, you can safely set the upper bounds to the largest possible length (65,535 pixels) when you're using `GrowWindow` to follow cursor movements.

Return Value

A long integer that specifies the new dimensions, in pixels, of the resulting window: the height in the high-order word of the returned long-integer value and the width in the low-order word. A return value of 0 means that the new size is the same as the size of the current port rectangle.

Discussion

The `GrowWindow` function displays an outline (grow image) of the window as the user moves the cursor to make the window larger or smaller; it handles all user interaction until the user releases the mouse button. After calling `GrowWindow`, you call the function `SizeWindow` to change the size of the window.

The `GrowWindow` function moves a dotted-line image of the window's right and lower edges around the screen, following the movements of the cursor until the mouse button is released. You can use the functions `HiWord` and `LoWord`, described in the *Mathematical and Logical Utilities Reference*, to retrieve only the high-order and low-order words, respectively.

Special Considerations

In non-Carbon implementations of `GrowWindow` on Mac OS 8 and 9, the maximum size that the specified window is allowed to grow to is actually one less than the values specified in the `bBox` parameter. For example, if you pass the values 500 in the `bBox.bottom` field and 600 in the `bBox.right` field, the maximum height and width of the window would actually be 499 and 599, respectively.

However, in Carbon, the maximum height and width allowed for the specified window is equal to the values passed in the `bBox.bottom` and `bBox.right` fields, respectively.

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

Declared In
`MacWindows.h`

InstallWindowContentPaintProc

Installs a window content painting callback. (Deprecated in Mac OS X v10.5. Use a `kEventControlDraw` Carbon event handler on a window's content view instead.)

```
OSStatus InstallWindowContentPaintProc (
   WindowRef window,
   WindowPaintUPP paintProc,
   WindowPaintProcOptions options,
   void *refCon
);
```

Parameters

window
The window whose default content painting function you want to override.

paintProc
A UPP to your window painting callback function. See `WindowPaintProcPtr` (page 174) for more information about the format of this function.

options
The options that are to be set. See "Window Paint Callback Options" (page 239) for a list of possible values.

refCon
Application-defined data. This data is passed to your callback when it is called.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Discussion
To remove a previously-installed paint proc (returning to the standard window manager erase-to-white content painting), pass `NULL` in the `paintProc` and `refCon` parameters.

Special Considerations

Instead of using this function, you should install a Carbon event handler for the `kEventControlDraw` event on a window's content view.

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

Declared In
`MacWindows.h`

InvokeWindowDefUPP

Invokes the UPP for a window definition. (Deprecated in Mac OS X v10.5. The WDEF interface is deprecated; use a custom HIView to draw your custom window frame instead.)

```
long InvokeWindowDefUPP (
   short varCode,
   WindowRef window,
   short message,
   long param,
   WindowDefUPP userUPP
);
```

Parameters

varCode
The window's variation code.

window
The window whose UPP is to be invoked.

message
The message.

param
The parameter.

userUPP
The UPP to invoke.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

InvokeWindowPaintUPP

Invokes the UPP for the specified painting region. (Deprecated in Mac OS X v10.5. The window content painting interface is deprecated; use a `kEventControlDraw` Carbon event handler on a compositing window's content view instead.)

```
OSStatus InvokeWindowPaintUPP (
   GDHandle device,
   GrafPtr qdContext,
   WindowRef window,
   RgnHandle inClientPaintRgn,
   RgnHandle outSystemPaintRgn,
   void *refCon,
   WindowPaintUPP userUPP
);
```

Parameters

device
The graphics device on which the window background should be painted.

qdContext
The QuickDraw port in which the window background should be painted.

window
The window whose UPP is to be invoked.

inClientPaintRgn
The region of the window background that needs to be painted, in local coordinates.

outSystemPaintRgn
On return, the region of the window background that the paint proc requests the Window Manager to paint.

refCon
Application-defined data.

userUPP
The UPP to invoke. For more information on this data type, see `WindowPaintUPP` (page 183).

Return Value
A result code. See "Window Manager Result Codes" (page 247).

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

Declared In
`MacWindows.h`

IsWindowPathSelectClick

Reports whether a mouse click should activate the window path pop-up menu. (Deprecated in Mac OS X v10.5. Use `IsWindowPathSelectEvent` (page 117) instead.)

```
Boolean IsWindowPathSelectClick (
   WindowRef window,
   const EventRecord *event
);
```

Parameters

window
The window in which the mouse-down event occurred.

event
A pointer to the `EventRecord` structure containing the mouse-down event that `IsWindowPathSelectClick` is to examine.

Return Value
A Boolean whose value is `true` f the mouse click should activate the window path pop-up menu; otherwise `false`.

Discussion
The Window Manager provides system support for your application to display window path pop-up menus, such as those used in Finder windows. When the user presses the Command key and clicks on the window's title, the window displays a pop-up menu containing a standard file system path, informing the user of the location of the document displayed in the window and allowing the user to open windows for folders along the path.

Because the window title includes both the proxy icon region and part of the drag region of the window, your application must be prepared to respond to a click in either region by displaying a window path pop-up menu. Therefore, when the `FindWindow` (page 48) function returns either the `inDrag` or the `inProxyIcon` result code—you should pass the event to the `IsWindowPathSelectClick` function to determine whether the mouse-down event should activate the window path pop-up menu. If `IsWindowPathSelectClick` returns a value of `true`, your application should then call the function `WindowPathSelect` (page 166) to display the menu.

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

Declared In
`MacWindows.h`

NewCWindow

Creates a window with a specified list of characteristics. (Deprecated in Mac OS X v10.5. Use `CreateNewWindow` (page 41) instead.)

```
WindowRef NewCWindow (
   void *wStorage,
   const Rect *boundsRect,
   ConstStr255Param title,
   Boolean visible,
   short procID,
   WindowRef behind,
   Boolean goAwayFlag,
   SRefCon refCon
);
```

Parameters

wStorage

On input, a pointer to the window structure. If you specify `null` as the value of `wStorage`, `NewCWindow` allocates the window structure as a nonrelocatable object in the application heap. You can reduce the chances of heap fragmentation by allocating memory from a block of memory reserved for this purpose by your application and passing a pointer to it in the `wStorage` parameter.

boundsRect

On input, a pointer to a rectangle, given in global coordinates, that specifies the window's initial size and location. This rectangle becomes the port rectangle of the window's graphics port. For the standard window types, the `boundsRect` field defines the content region of the window. The `NewCWindow` function places the origin of the local coordinate system at the upper-left corner of the port rectangle. `NewCWindow` calls the QuickDraw function `OpenCPort` to create the graphics port. The bitmap, pen pattern, and other characteristics of the window's graphics port are the same as the default values set by `OpenCPort`, except for the character font, which is set to the application font instead of the system font.

title

On input, a pascal string that specifies the window's title. If the title is too long to fit in the title bar, the title is truncated. To suppress the title in a window with a title bar, pass an empty string, not `null`, in the title parameter. `null` is an invalid value and may cause runtime errors.

visible

On input, a Boolean value indicating visibility status: `true` means that the Window Manager displays the window; `false` means it does not. If the value of the `visible` parameter is `true`, the Window Manager draws a new window as soon as the window exists. The Window Manager first calls the window definition function to draw the window frame. If the value of the `goAwayFlag` parameter is also `true` and the window is frontmost (that is, if the value of the behind parameter is (`WindowRef`)–`1L`), the Window Manager instructs the window definition function to draw a close box in the window frame. After drawing the frame, the Window Manager generates an update event to trigger your application's drawing of the content region.

When you create a window, you typically specify `false` as the value of the `visible` parameter. When you're ready to display the window, call `ShowWindow`.

procID

On input, the window's definition ID, a value that specifies both the window definition function and the variation code within that definition function. For a list of possible values, see "Pre-Appearance Window Definition IDs" (page 244).

behind

On input, a pointer to the window that appears immediately in front of the new window on the desktop. To place a new window in front of all other windows on the desktop, specify a value of (`WindowRef`)–`1L`. When you place a new window in front of all others, `NewCWindow` removes highlighting from the previously active window, highlights the newly created window, and generates activate events that trigger your application's updating of both windows. Note that if you create an invisible window in front of all others on the desktop, the user sees no active window until you make the new window visible (or make another window active).

To place a new window behind all other windows, specify a value of `null`.

goAwayFlag

On input, a Boolean value that determines whether the window has a close box. If the value of `goAwayFlag` is `true` and the window type supports a close box, the Window Manager draws a close box in the title bar and recognizes mouse clicks in the close region; if the value of `goAwayFlag` is `false` or the window type does not support a close box, it does not.

refCon

On input, a window's reference constant, set and used only by your application.

Return Value
A pointer to the newly created window structure.

Discussion
The `NewCWindow` function creates a window as specified by its parameters, adds it to the window list, and returns a pointer to the newly created window structure. You can use the returned window pointer to refer to this window in most Window Manager functions. If `NewCWindow` is unable to read the window definition function from the resource file, it returns `null`.

The `NewCWindow` function looks for a `'wctb '` resource with the same resource ID as the `'WIND '` resource. If it finds one, it uses the window color information in the `'wctb '` resource for coloring the window content region.

If the window's definition function is not already in memory, `NewCWindow` reads it into memory and stores a handle to it in the window structure. It allocates space for the structure and content regions of the window.

Storing the characteristics of your windows as resources, especially window titles and window items, makes your application easier to localize.

The `NewCWindow` function creates a window in a color graphics port. Creating color windows whenever possible ensures that your windows appear on color monitors with whatever color options the user has selected. Your application typically sets up its own set of global variables reflecting the system setup during initialization by calling the `Gestalt` function.

Special Considerations

If you let the Window Manager create the window structure in your application's heap, call `DisposeWindow` to close the window and dispose of its window structure. If you allocated the memory for the window structure yourself and passed a pointer to `NewCWindow`, use the function `CloseWindow` to close the window and the appropriate disposal function (determined by how you have allocated memory) to dispose of the window structure.

Carbon Porting Notes

In Carbon, you cannot pass your own storage in to the `wStorage` parameter.

Carbon does not support custom window definitions stored in 'WDEF' resources. If you want to specify a custom window definition for `NewCWindow`, you must compile your definition function directly in your application and then register the function by calling `RegisterWindowDefinition`. When `NewCWindow` gets a `procID` value that doesn't recognize, it checks a special mapping table to find the pointer that's registered for the resource ID embedded in the `procID` parameter. It then calls that function to implement your window.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

NewWindow

Creates a window from a parameter list. (Deprecated in Mac OS X v10.5. Use `CreateNewWindow` (page 41) instead.)

```
WindowRef NewWindow (
   void *wStorage,
   const Rect *boundsRect,
   ConstStr255Param title,
   Boolean visible,
   short theProc,
   WindowRef behind,
   Boolean goAwayFlag,
   SRefCon refCon
);
```

Parameters

wStorage

On input, a pointer to the window structure. If you specify `null` as the value of `wStorage`, `NewWindow` allocates the window structure as a nonrelocatable object in the heap. You can reduce the chances of heap fragmentation by allocating the storage from a block of memory reserved for this purpose by your application and passing a pointer to it in the `wStorage` parameter.

boundsRect

On input, a pointer to a rectangle, given in global coordinates, which specifies the window's initial size and location. This rectangle becomes the port rectangle of the window's graphics port. For the standard window types, `boundsRect` defines the content region of the window. The `NewWindow` function places the origin of the local coordinate system at the upper-left corner of the port rectangle. `NewWindow` calls the QuickDraw function `OpenPort` to create the graphics port. The bitmap, pen pattern, and other characteristics of the window's graphics port are the same as the default values set by `OpenPort`, except for the character font, which is set to the application font instead of the system font. The coordinates of the graphics port's port boundaries and visible region are changed along with its port rectangle.

title

On input, a pascal string that specifies the window's title. If the title is too long to fit in the title bar, the title is truncated. To suppress the title in a window with a title bar, pass an empty string, not `null`. `null` is an invalid value and may cause runtime errors.

visible

On input, a Boolean value indicating visibility status: `true` means that the Window Manager displays the window; `false` means it does not.

If the value of the `visible` parameter is `true`, the Window Manager draws a new window as soon as the window exists. The Window Manager first calls the window definition function to draw the window frame. If the value of the `goAwayFlag` parameter (described below) is also `true` and the window is frontmost (that is, if the value of the behind parameter is (`WindowRef`(–`1L`), the Window Manager instructs the window definition function to draw a close box in the window frame. After drawing the frame, the Window Manager generates an update event to trigger your application's drawing of the content region.

When you create a window, you typically specify `false` as the value of the `visible` parameter. When you're ready to display the window, you call the function `ShowWindow`.

theProc

On input, the window's definition ID, which specifies both the window definition function and the variation code for that definition function. For a list of possible values, see "Pre-Appearance Window Definition IDs" (page 244).

behind

On input, a pointer to the window that appears immediately in front of the new window on the desktop. To place a new window in front of all other windows on the desktop, specify a value of (`WindowRef`)–`1L`. When you place a new window in front of all others, `NewWindow` removes highlighting from the previously active window, highlights the newly created window, and generates activate events that trigger your application's updating of both windows. Note that if you create an invisible window in front of all others on the desktop, the user sees no active window until you make the new window visible (or make another window active).

To place a new window behind all other windows, specify a value of `null`.

goAwayFlag

On input, a Boolean value that determines whether or not the window has a close box. If the value of `goAwayFlag` is `true` and the window type supports a close box, the Window Manager draws a close box in the title bar and recognizes mouse clicks in the close region; if the value of `goAwayFlag` is `false` or the window type does not support a close box, it does not. The `goAwayFlag` parameter is ignored for movable modal or modal dialog boxes which do not support a close box.

refCon

On input, the window's reference constant, set and used only by your application.

Discussion

The `NewWindow` function takes the same parameters as `NewCWindow` and returns a `WindowRef` as its function result. The only difference is that `NewWindow` creates a window in a monochrome graphics port, not a color graphics port. The window structure and graphics port structure that describe monochrome and color graphics ports are the same size and can be used interchangeably in most Window Manager functions.

The `NewWindow` function creates a window as specified by its parameters, adds it to the window list, and returns a pointer to the newly created window structure. You can use the returned window pointer to refer to this window in most Window Manager functions. If `NewWindow` is unable to read the window definition function from the resource file, it returns `null`.

If the window's definition function is not already in memory, `NewWindow` reads it into memory and stores a handle to it in the window structure. It allocates space for the structure and content regions of the window.

Storing the characteristics of your windows as resources, especially window titles and window items, makes your application easier to localize.

Special Considerations

If you let the Window Manager create the window structure in your application's heap, call `DisposeWindow` to close the window and dispose of its window structure. If you allocated the memory for the window structure yourself and passed a pointer to `NewWindow`, use the function `CloseWindow` to close the window and the appropriate disposal function (determined by how you have allocated memory) to dispose of the window structure.

Version Notes

The `NewWindow` function was originally implemented prior to Color QuickDraw. In Mac OS 8, you should call the Color QuickDraw function `NewCWindow` instead of `NewWindow` to programmatically create a window, because Color QuickDraw is always available in Mac OS 8. Use of this function is not recommended with Mac OS 8 and later. `NewWindow` is described here only for completeness.

Carbon Porting Notes

In Carbon, you cannot pass your own storage in to the `wStorage` parameter.

In Carbon, `NewWindow` is functionally equivalent to the `NewCWindow`, in that `NewWindow` returns a color window instead of a monochrome window.

Carbon does not support custom window definitions stored in 'WDEF' resources. If you want to specify a custom window definition for `NewWindow`, you must compile your definition function directly in your application and then register the function by calling `RegisterWindowDefinition`. When `NewWindow` gets a `procID` value that doesn't recognize, it checks a special mapping table to find the pointer that's registered for the resource ID embedded in the `procID` parameter. It then calls that function to implement your window.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

NewWindowDefUPP

Creates a new UPP for a window definition. (Deprecated in Mac OS X v10.5. The WDEF interface is deprecated; use a custom HIView to draw your custom window frame instead.)

```
WindowDefUPP NewWindowDefUPP (
   WindowDefProcPtr userRoutine
);
```

Parameters

userRoutine

For information, see `WindowDefProcPtr` (page 169).

Return Value

For a description of the `WindowDefUPP` data type, see `WindowDefUPP` (page 182).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

NewWindowPaintUPP

Creates a new UPP for a painting region. (Deprecated in Mac OS X v10.5. The window content painting interface is deprecated; use a `kEventControlDraw` Carbon event handler on a compositing window's content view instead.)

```
WindowPaintUPP NewWindowPaintUPP (
   WindowPaintProcPtr userRoutine
);
```

Parameters

userRoutine

For information, see `WindowPaintProcPtr` (page 174).

Return Value

A UPP to the window paint function.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`MacWindows.h`

PaintBehind

Redraws a series of windows in the window list. (Deprecated in Mac OS X v10.5. Use `InvalWindowRect` (page 109), `InvalWindowRgn` (page 110), or `HIViewSetNeedsDisplay` to invalidate a portion of a window.)

```
void PaintBehind (
   WindowRef startWindow,
   RgnHandle clobberedRgn
);
```

Parameters

startWindow

On input, a pointer to the window's complete window structure.

clobberedRgn

On input, a handle to the region that has become invalid.

Discussion

The Window Manager calls the `PaintBehind` function; your application does not normally need to. `PaintBehind` calls `PaintOne` for `startWindow` and all the windows behind `startWindow`, clipped to `clobberedRgn`.

Special Considerations

Mac OS X applications never need to call this function.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

PaintOne

Redraws the invalid, exposed portions of one window on the desktop. (Deprecated in Mac OS X v10.5. Use `InvalWindowRect` (page 109), `InvalWindowRgn` (page 110), or `HIViewSetNeedsDisplay` to invalidate a portion of a window.)

```
void PaintOne (
   WindowRef window,
   RgnHandle clobberedRgn
);
```

Parameters

window

On input, a pointer to the window structure.

clobberedRgn

On input, a handle to the region that has become invalid.

Discussion

The Window Manager calls the `PaintOne` function; your application does not normally need to. `PaintOne` "paints" the invalid portion of the specified window and all windows above it. `PaintOne` draws as much of the window frame as is in `clobberedRgn` and, if some content region is exposed, erases the exposed area (paints it with the content pattern rather than the background pattern using `SetWinColor` or `SetThemeWindowBackground`) and adds it to the window's update region.

If the value of the `window` parameter is `null`, the window is the desktop, and `PaintOne` paints it with the desktop pattern.

Special Considerations

Mac OS X applications never need to call this function.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

ReleaseQDContextForCollapsedWindowDockTile

Releases a port and other state created by `CreateQDContextForCollapsedWindowDockTile`. (Deprecated in Mac OS X v10.5. Use `HIWindowReleaseCollapsedDockTileContext` (page 104) instead.)

```
OSStatus ReleaseQDContextForCollapsedWindowDockTile (
   WindowRef inWindow,
   CGrafPtr inContext
);
```

Parameters

inWindow

The window whose port is to be released.

inContext

The port that is to be released.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Discussion

You must call this function instead of calling `DisposePort` directly, or you may leak system resources.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Related Sample Code

QTCarbonShell

Declared In

`MacWindows.h`

ReleaseWindow

Decrements the retain count of a window, and destroys the window if the retain count falls to zero. (Deprecated in Mac OS X v10.5. Use `CFRelease` instead.)

```
OSStatus ReleaseWindow (
   WindowRef inWindow
);
```

Parameters

inWindow

The window whose retain count is to be decremented.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Discussion
This API is equivalent to `DisposeWindow` (page 45). For consistency with Core Foundation and Carbon Events, it is preferred over `DisposeWindow`. Both APIs will continue to be supported.

In Mac OS X v10.2 and later, you can also call `CFRelease` to decrement the retain count of a window.

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

Declared In
`MacWindows.h`

RetainWindow

Increments the retain count of a window. (Deprecated in Mac OS X v10.5. Use `CFRetain` instead.)

```
OSStatus RetainWindow (
   WindowRef inWindow
);
```

Parameters

inWindow

The window whose retain count is to be incremented.

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Discussion
This API is equivalent to `CloneWindow` (page 253). For consistency with Core Foundation and Carbon Events, it is preferred over `CloneWindow`. Both APIs will continue to be supported.

In Mac OS X v10.2 and later, you can also call `CFRetain` to increment the retain count of a window.

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

Declared In
`MacWindows.h`

SetWindowClass

Sets the class of a window. (Deprecated in Mac OS X v10.5. Use `HIWindowChangeClass` (page 87), `SetWindowGroup` (page 140), or `HIWindowChangeAttributes` (page 86) instead.)

```
OSStatus SetWindowClass (
   WindowRef inWindow,
   WindowClass inWindowClass
);
```

Parameters

window

The window whose class you want to set.

inClass

The class that is to be set. See "Window Class Constants" (page 184) for a list of possible window classes.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Discussion

This function changes the class of a window. It also changes the window's z-order so that it is grouped with other windows of the same class. It does not change the visual appearance of the window.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

SetWindowPic

Sets a picture for the Window Manager to draw in a window's content region. (Deprecated in Mac OS X v10.5. Use an HIImageView object to draw a window's content instead.)

```
void SetWindowPic (
   WindowRef window,
   PicHandle pic
);
```

Parameters

window

The window whose picture is to be set.

pic

On input, a handle to the picture to be drawn in the window.

Discussion

The `SetWindowPic` function stores in a window structure a handle to a picture to be drawn in the window. When the window's content region must be updated, the Window Manager then draws the picture or part of the picture, as necessary, instead of generating an update event.

The DisposeWindow (page 45) function assumes that any picture pointed to by the window structure field windowPic is stored as data, not as a resource. If your application uses a picture stored as a resource, you must release the memory it occupies by calling the Resource Manager's ReleaseResource function and set the WindowPic field to NULL before you close the window.

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

Declared In
MacWindows.h

SetWindowProxyFSSpec

Associates a file with a window. (Deprecated in Mac OS X v10.5. Use HIWindowSetProxyFSRef (page 106) instead.)

```
OSStatus SetWindowProxyFSSpec (
   WindowRef window,
   const FSSpec *inFile
);
```

Parameters

window

A pointer to the window with which the specified file is to be associated.

inFile

Set the file system specification structure to contain the data for the file to associate with the specified window. You can obtain an FSSpec structure by calling the function GetWindowProxyFSSpec (page 265).

Return Value
A result code. See "Window Manager Result Codes" (page 247).

Discussion
Your application should call the SetWindowProxyFSSpec function to establish a proxy icon for a given window. The creator code and file type of the file associated with a window determine the proxy icon that is displayed for the window.

Because the SetWindowProxyFSSpec function won't work without a saved file, you must establish the initial proxy icon for a new, untitled window with the function SetWindowProxyCreatorAndType, which requires that you know the file type and creator code for the file, but does not require that the file have been saved.

You must save and restore the current graphics port—by calling the QuickDraw functions GetPort and SetPort—around each call to the SetWindowProxyFSSpec function.

See also the function SetWindowProxyAlias.

Special Considerations

The use of file specifications is no longer recommended.

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

Not available to 64-bit applications.

Related Sample Code
QTCarbonShell

Declared In
`MacWindows.h`

SetWTitle

Specifies a window's title. (Deprecated in Mac OS X v10.5. Use `SetWindowTitleWithCFString` (page 151) instead.)

```
void SetWTitle (
   WindowRef window,
   ConstStr255Param title
);
```

Parameters

window

On input, a pointer to the window structure.

title

On input, a Pascal string containing the window title. To suppress the title in a window with a title bar, pass an empty string, not `null`.

Discussion
The `SetWTitle` function changes a window's title to the specified string, both in the window structure and on the screen, and redraws the window's frame as necessary. Always use `SetWTitle` instead of changing the title in a window structure.

When the user opens a previously saved document, you typically create a new (invisible) window with the title "untitled" and then call `SetWTitle` to give the window the document's name before displaying it. You also call `SetWTitle` when the user saves a document under a new name.

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

Declared In
`MacWindows.h`

StoreWindowIntoCollection

Stores data describing a window into a collection. (Deprecated in Mac OS X v10.5. Use `HIArchiveEncodeCFType` to encode a window to an archive instead.)

```
OSStatus StoreWindowIntoCollection (
   WindowRef window,
   Collection collection
);
```

Parameters

window

The window to be stored.

collection

A reference to the collection into which the window is to be stored. You pass a reference to a previously created collection, such as that returned by the Collection Manager function `NewCollection`.

Return Value

A result code. See "Window Manager Result Codes" (page 247).

Discussion

The `StoreWindowIntoCollection` function stores any window—including those not created by the Window Manager calls—into the specified collection. The Window Manager does not empty the collection beforehand, so any existing items in the collection remain.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`MacWindows.h`

Document Revision History

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

Date	Notes
2007-10-31	Made minor technical corrections.
2006-12-06	Updated for Mac OS X v10.5.
	Added information about new and deprecated functions. Added new "Window Attribute Identifiers" (page 188). Retired the Window Manager Legacy Reference and moved supported legacy API back into this document.
2005-09-08	Revised for Mac OS X v10.4.
2003-03-01	Added "Window Group Attributes" (page 227), and deprecated the previously listed group attributes (now under "Obsolete Window Group Attributes" (page 228).
2003-02-01	Updated formatting and linking.
	Added abstracts and availability info for new Jaguar APIs.
	Folded in API information from *Carbon Window Manager API Preliminary Documentation for CarbonLib 1.0*

Document Revision History

Index

E

F

G

H

I

K

M

N

O

P

R

S

T

U

V

W

Z