Carbon Event Manager Reference

2006-07-24

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

Simultaneously published in the United States and Canada.

Contents

Tables

Carbon Event Manager Reference

Framework:	Carbon/Carbon.h
Declared in:	CarbonEventsCore.h CarbonEvents.h

Introduction

The Carbon Event Manager is the preferred API for handling events in Carbon applications. You can use this interface to handle events generated in response to user input as well as to create your own custom events. Because event handling is so fundamental to all applications, this document is relevant for everyone writing Carbon applications. To use this document, you should be familiar with Macintosh terminology and understand the basics of creating and manipulating the Mac OS user interface (windows, controls, menus, and so on).

For more information about HIObjects and the HIView subclass, see HIView Programming Guide.

Functions by Task

Creating and Manipulating Event Handlers

`InstallEventHandler` (page 43)
Installs an event handler on a specified event target.

`InstallStandardEventHandler` (page 47)
Installs the standard event handler for the specified target.

`RemoveEventHandler` (page 61)
Removes the specified event handler.

`AddEventTypesToHandler` (page 19)
Adds events to an installed handler.

`RemoveEventTypesFromHandler` (page 62)
Removes events from an installed event handler.

`CallNextEventHandler` (page 20)
Calls the next handler in the handler chain.

`RegisterToolboxObjectClass` (page 59)
Registers events to be associated with a toolbox object. (Deprecated. Use the HIObject function `HIObjectRegisterSubclass` instead.)

`UnregisterToolboxObjectClass` (page 73)
Unregisters events for a given toolbox object class (Deprecated. Use the HIObject function `HIObjectUnregisterClass` instead.)

`RegisterEventHotKey` (page 58)
Registers a global hot key.

`UnregisterEventHotKey` (page 73)
Unregisters a global hot key.

Creating and Manipulating Event Timers

`InstallEventLoopTimer` (page 45)
Installs a timer.

`InstallEventLoopIdleTimer` (page 44)
Installs a timer that fires only when there is no user activity.

`RemoveEventLoopTimer` (page 61)
Removes the specified timer.

`SetEventLoopTimerNextFireTime` (page 65)
Sets the next time that the specified timer will fire.

Creating and Manipulating Events

`GetEventClass` (page 33)
Returns the class of an event (for example, window, mouse, or keyboard).

`GetEventKind` (page 34)
Returns the event kind for the specified event.

`GetEventParameter` (page 36)
Obtains a parameter from the specified event.

`SetEventParameter` (page 66)
Sets a parameter associated with a particular event.

`CreateEvent` (page 24)
Creates an event.

`CopyEvent` (page 22)
Copies an event.

`CopyEventAs` (page 22)
Copies an existing event, allowing you to change the class and kind of the event.

`RetainEvent` (page 62)
Increments the reference count of an event.

`ReleaseEvent` (page 60)
Releases, and possibly disposes of, the specified event.

`GetEventRetainCount` (page 37)
Returns the reference count of an event.

`ConvertEventRefToEventRecord` (page 21)
Converts an event reference into an event record.

`IsEventInMask` (page 49)
Determines whether an event reference matches a `WaitNextEvent`-style event mask.

`GetEventMonitorTarget` (page 35)
Obtains an event monitor target.

Dispatching Events

`SendEventToEventTarget` (page 64)
Sends an event to the specified event target.

`SendEventToEventTargetWithOptions` (page 65)
Sends an event to the specified event target with propagation options.

`GetControlEventTarget` (page 31)
Obtains the event target reference for the specified control.

`GetWindowEventTarget` (page 42)
Obtains the event target reference for a specified window.

`GetMenuEventTarget` (page 38)
Obtains an event target reference for the specified menu.

`GetApplicationEventTarget` (page 30)
Obtains the event target reference for the application.

`GetEventDispatcherTarget` (page 34)
Obtains the event target reference for the standard toolbox dispatcher.

`GetUserFocusEventTarget` (page 40)
Obtains the event target reference for the user focus.

`ProcessHICommand` (page 54)
Sends a command to the command chain.

Managing Secure Event Input

`EnableSecureEventInput` (page 27)
Enables secure event input mode.

`DisableSecureEventInput` (page 25)
Disables secure event input mode.

`IsSecureEventInputEnabled` (page 51)
Determines whether secure event input mode is enabled.

Managing Event Queues

`GetCurrentEventQueue` (page 33)
Obtains the current event queue.

`GetMainEventQueue` (page 38)
Obtains the main event queue.

`PostEventToQueue` (page 54)
Adds an event to the specified event queue.

`RemoveEventFromQueue` (page 60)
Removes an event from the event queue.

`IsEventInQueue` (page 50)
Determines whether an event is in a particular queue.

`AcquireFirstMatchingEventInQueue` (page 18)
Obtains the first event that matches the specified list of event classes and kinds.

`FlushEventsMatchingListFromQueue` (page 29)
Removes events from the event queue by kind and class.

`FindSpecificEventInQueue` (page 28)
Finds a specific event in the event queue.

`FlushSpecificEventsFromQueue` (page 30)
Removes specified events from the event queue.

`FlushEventQueue` (page 29)
Removes all events from the event queue.

`GetNumEventsInQueue` (page 39)
Returns the number of events in the event queue.

Managing the Event Loop

`RunApplicationEventLoop` (page 63)
Runs the application event loop.

`QuitApplicationEventLoop` (page 56)
Terminates the application event loop.

`GetMainEventLoop` (page 38)
Obtains a reference to the main event loop.

`GetCurrentEventLoop` (page 32)
Obtains a reference to the current event loop.

`GetCFRunLoopFromEventLoop` (page 31)
Obtains a Core Foundation `CFRunLoop` from an Carbon event loop reference.

`RunCurrentEventLoop` (page 64)
Executes the event loop in the current thread.

`QuitEventLoop` (page 56)
Causes a specific event loop to terminate.

`ReceiveNextEvent` (page 57)
Waits for the next event of a specified type.

Manipulating Event Time

`GetCurrentEventTime` (page 33)
Returns the current time since last system startup, in seconds.

`GetEventTime` (page 37)
Returns the time a specific event occurred.

`SetEventTime` (page 67)
Sets the event time for a given event.

`GetLastUserEventTime` (page 37)
Returns the last time a user input event arrived in the main event queue of the application.

Implementing Modal Windows

`RunAppModalLoopForWindow` (page 63)
Puts the window in an application-modal state.

`QuitAppModalLoopForWindow` (page 56)
Quits the application-modal state for a window.

`BeginAppModalStateForWindow` (page 20)
Puts the window in an application-modal state, but does not process events.

`EndAppModalStateForWindow` (page 28)
Ends the application-modal state entered using the function `BeginAppModalStateForWindow`.

`SetWindowDefaultButton` (page 69)
Specifies a default button for a window.

`GetWindowDefaultButton` (page 41)
Returns the current default button for a window.

`SetWindowCancelButton` (page 68)
Specifies a cancel button for a window.

`GetWindowCancelButton` (page 41)
Returns the current cancel button for a window.

Tracking the Mouse

`TrackMouseLocation` (page 70)
Tracks the mouse, blocking your application when there is no activity.

`TrackMouseLocationWithOptions` (page 71)
Tracks the mouse with additional options.

`TrackMouseRegion` (page 72)
Tracks the mouse within a region.

`HIMouseTrackingGetParameters` (page 42)
Obtains information about how mouse tracking loops should behave.

`ChangeMouseTrackingRegion` (page 207) Deprecated in Mac OS X v10.4
(Deprecated. Use `HIViewChangeTrackingArea` instead.)

`ClipMouseTrackingRegion` (page 207) Deprecated in Mac OS X v10.4
(Deprecated. No replacement function. Use HIView-based mouse tracking areas instead.)

`ClipWindowMouseTrackingRegions` (page 208) Deprecated in Mac OS X v10.4
(Deprecated. No replacement function. Use HIView-based tracking areas instead.)

`CreateMouseTrackingRegion` (page 208) Deprecated in Mac OS X v10.4
Creates a mouse tracking region. (Deprecated. Use the HIView function `HIViewNewTrackingArea` instead.)

`GetMouseTrackingRegionID` (page 209) Deprecated in Mac OS X v10.4
(Deprecated. Use `HIViewGetTrackingAreaID` instead.)

`GetMouseTrackingRegionRefCon` (page 210) Deprecated in Mac OS X v10.4
Obtains the reference constant for a mouse tracking region. (Deprecated. No replacement function. Use HIView-based mouse tracking areas instead.)

`MoveMouseTrackingRegion` (page 210) Deprecated in Mac OS X v10.4
(Deprecated. No replacement function. Use HIView-based tracking areas instead.)

`MoveWindowMouseTrackingRegions` (page 211) Deprecated in Mac OS X v10.4
(Deprecated. No replacement function. Use HIView-based tracking areas instead.)

`ReleaseMouseTrackingRegion` (page 211) Deprecated in Mac OS X v10.4
Releases a mouse tracking region. (Deprecated. Use `HIViewDisposeTrackingArea` instead.)

`ReleaseWindowMouseTrackingRegions` (page 212) Deprecated in Mac OS X v10.4
(Deprecated. No replacement function. Use HIView-based tracking areas instead.)

`RetainMouseTrackingRegion` (page 212) Deprecated in Mac OS X v10.4
Retains a mouse tracking region. (Deprecated. No replacement function. Use HIView-based tracking areas instead.)

`SetMouseTrackingRegionEnabled` (page 213) Deprecated in Mac OS X v10.4
(Deprecated. No replacement function. Use HIView-based tracking areas instead.)

SetWindowMouseTrackingRegionsEnabled (page 213) Deprecated in Mac OS X v10.4
(Deprecated. Use HIView-based tracking areas instead.)

Working with Hot Keys

CopySymbolicHotKeys (page 23)
Obtains information about symbolic hot keys in the Keyboard preferences pane.

PushSymbolicHotKeyMode (page 55)
Sets a new mode for enabling and disabling symbolic hot keys.

PopSymbolicHotKeyMode (page 53)
Removes a hot key mode request from the hot key mode stack.

GetSymbolicHotKeyMode (page 39)
Obtains the current hot key mode.

Callback-Related Functions

NewEventComparatorUPP (page 52)
Creates an event comparator UPP.

InvokeEventComparatorUPP (page 47)
Calls an event comparator function through a UPP.

DisposeEventComparatorUPP (page 26)
Disposes of an event comparator UPP.

NewEventHandlerUPP (page 52)
Creates an event handler UPP.

DisposeEventHandlerUPP (page 26)
Disposes of an event handler UPP.

InvokeEventHandlerUPP (page 48)
Calls an event handler through a UPP.

NewEventLoopTimerUPP (page 53)
Creates an event loop timer UPP.

InvokeEventLoopTimerUPP (page 49)
Calls an event loop timer through a UPP.

DisposeEventLoopTimerUPP (page 27)
Disposes of an event loop timer.

NewEventLoopIdleTimerUPP (page 53)
Creates an event loop idle timer UPP.

InvokeEventLoopIdleTimerUPP (page 49)
Calls an event loop idle timer through a UPP.

`DisposeEventLoopIdleTimerUPP` (page 27)
Disposes of an event loop idle timer.

Miscellaneous

`CopyServicesMenuCommandKeys` (page 23)
Obtains information about command key shortcuts in an application's Services menu.

`CreateTypeStringWithOSType` (page 25)
Converts an `OSType` string to a Core Foundation string.

`GetCurrentEventKeyModifiers` (page 31)
Obtains the queue-synchronized keyboard modifier state.

`IsMouseCoalescingEnabled` (page 50)
Indicates whether mouse coalescing is enabled.

`SetMouseCoalescingEnabled` (page 67)
Turns mouse coalescing on or off.

`IsUserCancelEventRef` (page 51)
Returns whether the specified event indicates the user wishes to cancel an operation.

`SetUserFocusWindow` (page 68)
Designates a window to receive user focus.

`GetUserFocusWindow` (page 40)
Returns the current user focus window.

Functions

AcquireFirstMatchingEventInQueue

Obtains the first event that matches the specified list of event classes and kinds.

```
EventRef AcquireFirstMatchingEventInQueue (
    EventQueueRef inQueue,
    UInt32 inNumTypes,
    const EventTypeSpec * inList,
    OptionBits inOptions
);
```

Parameters

inQueue
The queue to check.

inNumTypes
The number of event kinds for which to search. You may pass `0` if you also pass `NULL` for `inList`.

inList

The list of event classes and kinds to search for in the queue. You may pass `NULL` if `inNumTypes` is `0`. Doing so effectively matches *any* event in the queue and causes this function to return the first event in the queue.

inOptions

Must be `kEventQueueOptionsNone`.

Return value

An event reference, or `NULL` if no events match. The reference count for the event has been incremented (that is, it has been retained), so you must release the event reference.

Discussion

This function does not remove the event from the queue. To remove the event, call `RemoveEventFromQueue` (page 60).

This function does not call the run loop, so no timers fire as a result of calling this function. This function does not cause any window flushing to occur, but it does get new events from the CoreGraphics window server.

This function should have better performance characteristics than the older `EventAvail` API.

Availability

Available in Mac OS X v10.3 and later.

Declared In

CarbonEventsCore.h

AddEventTypesToHandler

Adds events to an installed handler.

```
OSStatus AddEventTypesToHandler (
    EventHandlerRef inHandlerRef,
    UInt32 inNumTypes,
    const EventTypeSpec * inList
);
```

Parameters

inHandlerRef

The event handler to add events to.

inNumTypes

The number of events to add.

inList

A pointer to an array of `EventTypeSpec` structures.

Return value

A result code. See "Carbon Event Manager Result Codes" (page 204).

Discussion

You can use this function to dynamically change which events you want your handler to respond to.

Availability

Available in Mac OS X v10.0 and later.

Declared In
CarbonEventsCore.h

BeginAppModalStateForWindow

Puts the window in an application-modal state, but does not process events.

```
OSStatus BeginAppModalStateForWindow (
    WindowRef inWindow
);
```

Parameters

inWindow
The window you wish to behave modally. See the Window Manager documentation for a description of the `WindowRef` data type.

Return value
A result code. See "Carbon Event Manager Result Codes" (page 204).

Discussion
This function is a lower level function than `RunAppModalLoopForWindow` (page 63). You use it if you want to enter an application modal state for a window but need to control the event loop yourself. Once you begin your application modal state, the menu bar will disable and prepare for the modal situation. You can then call low-level functions (such as `ReceiveNextEvent`) to run the event loop and process events.

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

Declared In
CarbonEvents.h

CallNextEventHandler

Calls the next handler in the handler chain.

```
OSStatus CallNextEventHandler (
    EventHandlerCallRef inCallRef,
    EventRef inEvent
);
```

Parameters

inCallRef
The event handler call reference passed into your event handler.

inEvent
The event you want to pass to the next handler.

Return value
A result code. See "Carbon Event Manager Result Codes" (page 204).

Discussion
Calls through to the event handlers below you in the event handler stack of the target to which your handler is bound. You might use this to call through to the default toolbox handling in order to post-process the event. You can only call this routine from within an event handler.

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

Declared In
CarbonEventsCore.h

ConvertEventRefToEventRecord

Converts an event reference into an event record.

```
Boolean ConvertEventRefToEventRecord (
    EventRef inEvent,
    EventRecord * outEvent
);
```

Parameters
inEvent
The event reference to convert.

outEvent
The event record to fill out. See the Event Manager documentation for a description of the `EventRecord` data type.

Return value
A Boolean value indicating whether the conversion was successful (`true`) or not (`false`).

Discussion
This function helps you when you need an `EventRecord` structure and all you have is a Carbon event reference. If the event can be converted, `outEvent` is filled in and the function returns `true`. If not, the function returns `false` and `outEvent` contains `nullEvent`.

This function can convert the following events:

- `kEventMouseDown`, `kEventMouseUp`, `kEventMouseMoved`, and `kEventMouseDragged` (`kEventClassMouse`)
- `kEventRawKeyDown`, `kEventRawKeyUp`, and `kEventRawKeyRepeat` (`kEventClassKeyboard`)
- `kEventWindowUpdate`, `kEventWindowActivate`, `kEventWindowDeactivate`, and `kEventWindowCursorChange` (`kEventClassWindow`)
- `kEventAppActivated` and `kEventAppDeactivate` (`kEventClassApplication`)
- `kEventAppleEvent` (`kEventClassAppleEvents`)
- `kEventControlTrack` (`kEventClassControl`) is converted to a mouse down event in Mac OS X v10.4 and later

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

Declared In
CarbonEvents.h

CopyEvent

Copies an event.

```
EventRef CopyEvent (
    EventRef inOther
);
```

Parameters

inOther
The event to copy.

Return value
A new event reference for the specified event.

Discussion
The `CopyEvent` function makes an exact duplicate of an existing event reference. The reference count for the duplicate event reference is set to `1`.

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

Declared In
CarbonEventsCore.h

CopyEventAs

Copies an existing event, allowing you to change the class and kind of the event.

```
EventRef CopyEventAs (
    CFAllocatorRef inAllocator,
    EventRef inOther,
    UInt32 inEventClass,
    UInt32 inEventKind
);
```

Parameters

inOther
The allocator to use to allocate the event data. Pass `NULL` or `kCFAllocatorDefault` to use the standard allocator.

inOther
The event to copy.

inEventClass
The new event class for the copy of the event.

inEventKind
The new event kind for the copy of the event.

Return value
A new event reference or `NULL` if the `inOther` was `NULL` or memory for the new event could not be allocated.

Discussion
The `CopyEventAs` is useful during event flow and transformation. For example, this function is used when upgrading a raw mouse down to a window click event, to ensure that the window click event has exactly the same parameters as the original mouse down event.

Availability
Available in Mac OS X v10.3 and later.

Declared In
CarbonEventsCore.h

CopyServicesMenuCommandKeys

Obtains information about command key shortcuts in an application's Services menu.

```
OSStatus CopyServicesMenuCommandKeys (
    CFArrayRef* outCommandKeyArray
);
```

Parameters

`outCommandKeyArray`

On return, an array of items in the Services menu that have command key shortcuts associated with them.

Return value
A result code. See "Carbon Event Manager Result Codes" (page 204).

Discussion
Each array entry is a reference to a CFDictionary, and each dictionary contains information about a single command key shortcut for items in the application's Services menu. Each dictionary contains the following keys: `kHIServicesMenuProviderName`, `kHIServicesMenuItemName`, `kHIServicesMenuCharCode`, and `kHIServicesMenuKeyModifiers`. The array must be released by the caller. The dictionaries do not need to be released because they are released automatically when the array is released.

Availability
Available in Mac OS X v10.4 and later.

Declared In
CarbonEvents.h

CopySymbolicHotKeys

Obtains information about symbolic hot keys in the Keyboard preferences pane.

```
OSStatus CopySymbolicHotKeys(
    CFArrayRef * outHotKeyArray
);
```

Parameters

outHotKeyArray

An array of dictionaries containing information about the systemwide symbolic hot keys defined in the Keyboard preferences pane, such as the Screen Capture, Universal Access, and Keyboard Navigation keys. The array does not include information about custom, application-specific command keys. You must release the array when you no longer need it. The dictionaries are automatically released when you release the array.

Return value

A result code. See "Carbon Event Manager Result Codes" (page 204).

Discussion

Each array entry is a reference for a CFDictionary, and each dictionary contains information about a single hot key. There is currently no way to determine which hot key in the Keyboards preference pane corresponds to a specific dictionary. Each dictionary contains the following keys: `kHISymbolicHotKeyCode`, `kHISymbolicHotKeyModifiers`, and `kHISymbolicHotKeyEnabled`. For details, see "Symbolic Hot Key Definitions" (page 133).

The number of hot keys will increase in the future, so do not call this function unnecessarily or in highly performance-sensitive code.

Availability

Available in Mac OS X v10.3 and later.

Declared In

CarbonEvents.h

CreateEvent

Creates an event.

```
OSStatus CreateEvent (
    CFAllocatorRef inAllocator,
    UInt32 inClassID,
    UInt32 kind,
    EventTime when,
    EventAttributes flags,
    EventRef * outEvent
);
```

Parameters

inAllocator

A reference to the desired memory allocator to use to allocate memory for the event. Pass `NULL` to use the default allocator. See the Base Services documentation for a description of the `CFAllocatorRef` data type.

inClassID

The event class of the event to create.

kind

The event kind of the event to create.

when

The time the event occurred. Pass `0` to specify the current event time (as returned by the `GetCurrentEventTime` (page 33) function).

flags

The event attributes to set. Currently you can pass `kEventAttributeNone` or `kEventAttributeUserEvent`.

outEvent

On return, a reference to the newly created event.

Return value
A result code. See "Carbon Event Manager Result Codes" (page 204).

Discussion
You can use this function to create your own custom events or to simulate existing events. If you are creating custom events, you must make sure that the event signature (the combination of event class and event kind) does not conflict with any existing events.

Declared In
CarbonEventsCore.h

CreateTypeStringWithOSType

Converts an `OSType` string to a Core Foundation string.

```
CFStringRef CreateTypeStringWithOSType (
    OSType inType
);
```

Return value
The Core Foundation string version of the `OSType` string. A return value of `NULL` indicates that an error occurred. See the Base Services documentation for a description of the `CFStringRef` data type.

Discussion
You can use this function to create `CFString` versions of `OSType` data types to pass to the Services Manager. As this is a creation function, you must call `CFRelease` on your Core Foundation string when you no longer need it.

Availability
Available in Mac OS X v10.1 and later.

Declared In
CarbonEvents.h

DisableSecureEventInput

Disables secure event input mode.

```
OSStatus DisableSecureEventInput (void);
```

Return value
A result code. See "Carbon Event Manager Result Codes" (page 204).

Discussion
When secure event input mode is enabled, keyboard input goes only to the application with keyboard focus and is not echoed to other applications that might be using the event monitor target to watch keyboard input. The `EditText` and `EditUnicodeText` controls automatically enter secure input mode when a password control has focus. If your application implements its own password entry, you should enable secure event input while the user enters text.

This function maintains a count of the number of times that it has been called. Secure event input is not disabled until DisableSecureEventInput has been called the same number of times. Be sure to disable secure event input if your application becomes inactive. If your application crashes, secure event input is automatically disabled if no other application has enabled it.

Availability
Available in Mac OS X v10.3 and later.

Declared In
CarbonEventsCore.h

DisposeEventComparatorUPP

Disposes of an event comparator UPP.

```
void DisposeEventComparatorUPP (
    EventComparatorUPP userUPP
);
```

Parameters

userUPP
The UPP you want to destroy.

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

Declared In
CarbonEventsCore.h

DisposeEventHandlerUPP

Disposes of an event handler UPP.

```
void DisposeEventHandlerUPP (
    EventHandlerUPP userUPP
);
```

Parameters

userUPP
The event handler UPP you want to destroy.

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

Declared In
CarbonEventsCore.h

DisposeEventLoopIdleTimerUPP

Disposes of an event loop idle timer.

```
void DisposeEventLoopIdleTimerUPP (
    EventLoopIdleTimerUPP userUPP
);
```

Availability
Available in Mac OS X v10.2 and later.

Declared In
CarbonEventsCore.h

DisposeEventLoopTimerUPP

Disposes of an event loop timer.

```
void DisposeEventLoopTimerUPP (
    EventLoopTimerUPP userUPP
);
```

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

Declared In
CarbonEventsCore.h

EnableSecureEventInput

Enables secure event input mode.

```
OSStatus EnableSecureEventInput (void);
```

Return value
A result code. See "Carbon Event Manager Result Codes" (page 204).

Discussion
When secure event input mode is enabled, keyboard input goes only to the application with keyboard focus and is not echoed to other applications that might be using the event monitor target to watch keyboard input. The `EditText` and `EditUnicodeText` controls automatically enter secure input mode when a password control has focus. If your application implements its own password entry, you should enable secure event input while the user enters text.

This function maintains a count of the number of times that it has been called. Secure event input is not disabled until `DisableSecureEventInput` (page 25) has been called the same number of times. Be sure to disable secure event input if your application becomes inactive. If your application crashes, secure event input is automatically disabled if no other application has enabled it.

Availability
Available in Mac OS X v10.3 and later.

Declared In
CarbonEventsCore.h

EndAppModalStateForWindow

Ends the application-modal state entered using the function `BeginAppModalStateForWindow`.

```
OSStatus EndAppModalStateForWindow (
    WindowRef inWindow
);
```

Parameters

inWindow

The window you wish to stop acting as application- modal. See the Window Manager documentation for a description of the `WindowRef` data type.

Return value

A result code. See "Carbon Event Manager Result Codes" (page 204).

Discussion

This routine ends an app modal state started with `BeginAppModalStateForWindow` (page 20).

Availability

Available in Mac OS X v10.0 and later.

Declared In

CarbonEvents.h

FindSpecificEventInQueue

Finds a specific event in the event queue.

```
EventRef FindSpecificEventInQueue (
    EventQueueRef inQueue,
    EventComparatorUPP inComparator,
    void * inCompareData
);
```

Parameters

inQueue

The event queue to search.

inComparator

The comparison function to invoke for each event in the queue. See `EventComparatorProcPtr` (page 74) for the required format of your comparison function A return value of `true` from the comparator indicates a match.

inCompareData

The data you wish to pass to your comparison function.

Return value

An event reference.

Discussion

Returns the first event that matches a comparator function, or `NULL` if no events match.

Availability

Available in Mac OS X v10.0 and later.

Declared In
CarbonEventsCore.h

FlushEventQueue

Removes all events from the event queue.

```
OSStatus FlushEventQueue (
    EventQueueRef inQueue
);
```

Parameters

inQueue
The event queue to flush.

Return value
A result code. See "Carbon Event Manager Result Codes" (page 204).

Discussion
Flushes all events from an event queue.

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

Declared In
CarbonEventsCore.h

FlushEventsMatchingListFromQueue

Removes events from the event queue by kind and class.

```
OSStatus FlushEventsMatchingListFromQueue (
    EventQueueRef inQueue,
    UInt32 inNumTypes,
    const EventTypeSpec * inList
);
```

Parameters

inQueue
The event queue to flush events from.

inNumTypes
The number of event kinds to flush.

inList
The list of event classes and kinds to flush from the queue.

Return value
A result code. See "Carbon Event Manager Result Codes" (page 204).

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

Declared In
CarbonEventsCore.h

FlushSpecificEventsFromQueue

Removes specified events from the event queue.

```
OSStatus FlushSpecificEventsFromQueue (
    EventQueueRef inQueue,
    EventComparatorUPP inComparator,
    void * inCompareData
);
```

Parameters

inQueue
The event queue to flush events from.

inComparator
The comparison function to invoke for each event in the queue. See `EventComparatorProcPtr` (page 74) for the required format of your comparison function. A return value of `true` from the comparator indicates that the event should be flushed.

inCompareData
The data you wish to pass to your comparison function.

Return value
A result code. See "Carbon Event Manager Result Codes" (page 204).

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

Declared In
CarbonEventsCore.h

GetApplicationEventTarget

Obtains the event target reference for the application.

```
EventTargetRef GetApplicationEventTarget ();
```

Return value
An event target reference.

Discussion
Once you obtain this reference, you can send events to the target and install event handlers on it.

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

Declared In
CarbonEvents.h

GetCFRunLoopFromEventLoop

Obtains a Core Foundation `CFRunLoop` from an Carbon event loop reference.

```
CFTypeRef GetCFRunLoopFromEventLoop (
    EventLoopRef inEventLoop
);
```

Parameters

inEventLoop

The event loop reference to translate.

Return value

A reference to the CFRunLoop.

Discussion

There isn't necessarily a one-to-one correspondence between Carbon event loops and Core Foundation event loops, so you should use this function instead of simply calling the Core Foundation function `CFRunLoopGetCurrent`.

Availability

Available in Mac OS X v10.1 and later.

Declared In

CarbonEventsCore.h

GetControlEventTarget

Obtains the event target reference for the specified control.

```
EventTargetRef GetControlEventTarget (
    ControlRef inControl
);
```

Parameters

inControl

The control to return the target for. See the Control Manager documentation for a description of the `ControlRef` data type.

Return value

An event target reference.

Discussion

Once you obtain this reference, you can send events to the target and install event handlers on it.

Availability

Available in Mac OS X v10.0 and later.

Declared In

CarbonEvents.h

GetCurrentEventKeyModifiers

Obtains the queue-synchronized keyboard modifier state.

```
UInt32 GetCurrentEventKeyModifiers(void)
```

Return value
A bit field indicating the queue-synchronized keyboard modifier state. This field is the same as the modifiers field returned in an Event Manager `EventRecord` structure, but it includes only the keyboard modifier flags.

Discussion
The queue-synchronized keyboard modifier state indicates the modifier state according to the event most recently dispatched through an event target. This state may be different from the hardware state obtained using `GetCurrentKeyModifiers`. For example, say the user invokes a Control-click with the mouse. If the user releases or changes a modifier key before the mouse down event is dispatched, the hardware state reflects the new modifier state, not the one that generated the original mouse event.

The most recently dispatched event may not necessarily be the event that your event handler is handling. For example, if a mouse-down event occurs, and you have a handler for the `kEventWindowHandleContentClick` event that is generated from the mouse-down, then the keyboard modifiers will be those that were attached to the mouse-down. The content-click event itself has a `kEventParamKeyModifiers` parameter, which is copied from the mouse-down event, but `GetCurrentEventKeyModifiers` returns the modifiers from the mouse-down, not from the content-click event, because it was the mouse-down event that was most recently dispatched through the event dispatcher.

Events that are not sent through the event dispatcher target will not update the current event key modifiers. Also, events arriving from outside the application, such as an AppleEvent or an Accessibility event, also will not update the modifiers. If your application modifies its behavior based on modifier state, you should parameterize your core code with the event modifiers, and determine the modifiers based on the origin of the behavior request. For a request that originates directly from user input, you can use `GetCurrentEventKeyModifiers`, but for a request that originates from an AppleEvent or Accessibility event, you would probably use no modifiers. `GetCurrentEventKeyModifiers` gives a more consistent user experience when the user input queue is being remoted controlled or manipulated via non-hardware event sources such as speech or AppleEvents; using `GetCurrentEventKeyModifiers` is also much faster than using EventAvail(0, &eventRecord) or GetCurrentKeyModifiers.

`GetCurrentEventKeyModifiers` returns a valid modifier state only if your application is the active application. If your application is not active, then user input events are not flowing through the event dispatcher and the queue-synchronized state is not updated.

Availability
Available in Mac OS X v10.2 and later.

Declared In
CarbonEventsCore.h

GetCurrentEventLoop

Obtains a reference to the current event loop.

```
EventLoopRef GetCurrentEventLoop ();
```

Return value
An event loop reference.

Discussion
This function returns the event loop for the current thread. If the current thread is a cooperative thread, the main event loop is returned.

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

Declared In
CarbonEventsCore.h

GetCurrentEventQueue

Obtains the current event queue.

```
EventQueueRef GetCurrentEventQueue ();
```

Return value
An event queue reference.

Discussion
This function obtains the event queue for the current thread. If the current thread is a cooperative thread, the main event queue is returned.

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

Declared In
CarbonEventsCore.h

GetCurrentEventTime

Returns the current time since last system startup, in seconds.

```
EventTime GetCurrentEventTime ();
```

Return value
EventTime.

Discussion
Returns the current time since last system startup in seconds.

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

Declared In
CarbonEventsCore.h

GetEventClass

Returns the class of an event (for example, window, mouse, or keyboard).

```
UInt32 GetEventClass (
    EventRef inEvent
```

```
);
```

Parameters

inEvent

The event in question.

Return value

The class ID of the event. See "Event Class Constants" (page 86) for more details.

Availability

Available in Mac OS X v10.0 and later.

Declared In

CarbonEventsCore.h

GetEventDispatcherTarget

Obtains the event target reference for the standard toolbox dispatcher.

```
EventTargetRef GetEventDispatcherTarget ();
```

Return value

An event target reference.

Discussion

The standard toolbox dispatcher is the default mechanism for dispatching events to the appropriate event targets. You typically don't need to call this, but some applications may need to pick events off the event queue and call the dispatcher themselves. This allows you to do just that instead of calling `RunApplicationEventLoop` to handle it all.

If desired, you can attach event handlers to the event dispatcher target. Doing so allows you to intercept any events before they can be sent to the appropriate event targets.

Availability

Available in Mac OS X v10.0 and later.

Declared In

CarbonEvents.h

GetEventKind

Returns the event kind for the specified event.

```
UInt32 GetEventKind (
    EventRef inEvent
);
```

Parameters

inEvent

The event in question.

Return value

The kind of the event.

Discussion
Event kind values overlap in different event classes. For example, `kEventMouseDown` and `kEventAppActivated` both have the same value (`1`). The combination of class and kind determines a unique event signature.

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

Declared In
CarbonEventsCore.h

GetEventMonitorTarget

Obtains an event monitor target.

```
EventTargetRef GetEventMonitorTarget (
    void
);
```

Return value
An event monitor target.

Discussion
The event monitor target is a special event target used to monitor user input events across all processes. When an event handler is installed on the event monitor target, the Carbon Event Manager examines the EventTypeSpec for user input event types, such as mouse-down, mouse-up, and key-down. It then requests that the WindowServer make copies of any of these events that are sent to any process, and delivers them to the current process. These events are queued into the main thread's event queue and are sent directly to the event handlers installed on the event monitor target during normal event dispatching. Monitored events are not sent through the normal event dispatching path for the current process. Instead, they pass through the event dispatcher target and are sent directly to the event monitor target.Handlers installed on the event monitor target receive events only when the current application is inactive. When the current application is active, all events flow through the event dispatcher target, and no events are sent to the event monitor target.Currently, the event monitor supports the following event kinds: `kEventRawKeyDown`, `kEventRawKeyUp`, `kEventRawKeyRepeat`, `kEventRawKeyModifiersChange`, `kEventMouseDown`, `kEventMouseUp`, `kEventMouseMoved`, `kEventMouseDragged`, `kEventMouseWheelMoved`, `kEventTabletPoint`, and `kEventTabletProximity`.To prevent keyboard events from being passed to other applications, Carbon and Cocoa password-edit-text controls enable a secure input mode while the focus is on the control. Their password-edit-text controls prevent the monitoring event target from being used to capture password keystrokes.For added security, `GetEventMonitorTarget` requires that "Enable access for assistive devices" be checked in the Universal Access preference pane in order to monitor `kEventRawKeyDown`, `kEventRawKeyUp`, and `kEventRawKeyRepeat` events. If this control is not checked, you can still install handlers for these events on the event monitor target, but no events of these types will be sent to your handler. Administrator privileges are required to enable this feature. You can determine whether this control is checked using the `AXAPIEnabled` function in `AXUIElement.h`.

Availability
Available in Mac OS X v10.3 and later.

Declared In
CarbonEvents.h

GetEventParameter

Obtains a parameter from the specified event.

```
OSStatus GetEventParameter (
    EventRef inEvent,
    EventParamName inName,
    EventParamType inDesiredType,
    EventParamType * outActualType,
    UInt32 inBufferSize,
    UInt32 * outActualSize,
    void * outData
);
```

Parameters

inEvent

The event to get the parameter from.

inName

The symbolic name of the parameter (for example, `kEventParamDirectObject`). The Carbon Event Manager defines a number of constants defining possible parameters.

inDesiredType

The desired type of the parameter (for example, `typeWindowRef`). The Carbon Event Manager automatically uses AppleEvent coercion handlers to convert the data in the event into the desired type, if possible. The Carbon Event Manager defines a number of constants to indicate possible parameter types. Pass `typeWildCard` to request that the data be returned in its original format.

outActualType

The actual type of the parameter (can be `NULL` if you are not interested in receiving this information).

inBufferSize

The size of the output buffer.

outActualSize

The actual size of the data, or `NULL` if you don't want this information.

outData

The pointer to the buffer receiving the parameter data.

Return value

A result code. See "Carbon Event Manager Result Codes" (page 204).

Discussion

Events often contain additional useful pieces of data, such as the location of a mouse-down event or the window in which an event occurred.

Availability

Available in Mac OS X v10.0 and later.

Declared In

CarbonEventsCore.h

GetEventRetainCount

Returns the reference count of an event.

```
UInt32 GetEventRetainCount (
    EventRef inEvent
);
```

Return value
The current reference count for the specified event.

Discussion
When an event is created, its reference count is 1. Calls to `RetainEvent` increment this count; calls to `ReleaseEvent` decrement the count.

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

Declared In
CarbonEventsCore.h

GetEventTime

Returns the time a specific event occurred.

```
EventTime GetEventTime (
    EventRef inEvent
);
```

Parameters
inEvent
The event in question.

Return value
The time the event occurred.

Discussion
Returns the time the event specified occurred, specified as an `EventTime` value, which is a floating point number representing seconds since the last system startup.

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

Declared In
CarbonEventsCore.h

GetLastUserEventTime

Returns the last time a user input event arrived in the main event queue of the application.

```
EventTime GetLastUserEventTime ();
```

Return value
The time of the last user event.

Discussion
A user input event is something generated by the user, typically a hardware event such as a mouse-click or key-down event.

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

Declared In
CarbonEvents.h

GetMainEventLoop

Obtains a reference to the main event loop.

```
EventLoopRef GetMainEventLoop ();
```

Return value
An event loop reference.

Discussion
The main loop is the event loop for the main application thread.

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

Declared In
CarbonEventsCore.h

GetMainEventQueue

Obtains the main event queue.

```
EventQueueRef GetMainEventQueue ();
```

Return value
An event queue reference.

Discussion
The main queue is the event queue for the main application thread.

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

Declared In
CarbonEventsCore.h

GetMenuEventTarget

Obtains an event target reference for the specified menu.

```
EventTargetRef GetMenuEventTarget (
    MenuRef inMenu
);
```

Parameters

inMenu

The menu to return the target for. See the Menu Manager documentation for a description of the `MenuRef` data type.

Return value

An event target reference.

Discussion

Once you obtain this reference, you can send events to the target and install event handlers on it.

Availability

Available in Mac OS X v10.0 and later.

Declared In

CarbonEvents.h

GetNumEventsInQueue

Returns the number of events in the event queue.

```
UInt32 GetNumEventsInQueue (
    EventQueueRef inQueue
);
```

Parameters

inQueue

The event queue to query.

Return value

The number of items in the queue.

Discussion

Returns the number of events in an event queue.

Availability

Available in Mac OS X v10.0 and later.

Declared In

CarbonEventsCore.h

GetSymbolicHotKeyMode

Obtains the current hot key mode.

```
OptionBits GetSymbolicHotKeyMode ();
```

Return value

The mode request at the top of the hot key mode stack. If there are no mode requests on the stack, this function returns `0` to indicate that hot keys are currently enabled.

Discussion
Unless the "Enable access for assistive devices" checkbox is checked in the Universal Access preference pane, all hot keys are enabled, even if this function returns a nonzero value. This means that hot keys enabled by the caller may be disabled for the current user session if they were disabled by another process.

Availability
Available in Mac OS X v10.4 and later.

Declared In
CarbonEvents.h

GetUserFocusEventTarget

Obtains the event target reference for the user focus.

```
EventTargetRef GetUserFocusEventTarget ();
```

Return value
An event target reference.

Discussion
This event target always references the current user focus. For example, if you install a handler on this target, then your handler will be called whenever an event is sent to the user focus. Keyboard events are always sent to this target.

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

Declared In
CarbonEvents.h

GetUserFocusWindow

Returns the current user focus window.

```
WindowRef GetUserFocusWindow ();
```

Return value
A reference to the window receiving user focus. See the QuickDraw Manager documentation for a description of the `WindowRef` data type.

Discussion
This function returns a reference to the current user focus window. This window receives menu commands and keyboard input as part of the standard event dispatching.

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

Declared In
CarbonEvents.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. See the Window Manager documentation for a description of the `WindowRef` data type.

outControl

On return, a reference to the Cancel control in the specified window. See the Control Manager documentation for a description of the `ControlRef` data type.

Return value

A result code. See "Carbon Event Manager Result Codes" (page 204).

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.

Declared In

CarbonEvents.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. See the Window Manager documentation for a description of the `WindowRef` data type.

outControl

On return, a reference to the default control. See the Control Manager documentation for a description of the `ControlRef` data type.

Return value

A result code. See "Carbon Event Manager Result Codes" (page 204).

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.

Declared In
CarbonEvents.h

GetWindowEventTarget

Obtains the event target reference for a specified window.

```
EventTargetRef GetWindowEventTarget (
    WindowRef inWindow
);
```

Parameters
inWindow

The window to return the event target for. See the QuickDraw Manager documentation for a description of the `WindowRef` data type.

Return value
An event target reference.

Discussion
Once you obtain this reference, you can send events to the target and install an event handler on it.

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

Declared In
CarbonEvents.h

HIMouseTrackingGetParameters

Obtains information about how mouse tracking loops should behave.

```
OSStatus HIMouseTrackingGetParameters(
    OSType inSelector,
    EventTime * outTime,
    HISize * outDistance
);
```

Parameters
inSelector

The type of information to obtain. Currently, the only supported selector is `kMouseParamsSticky`.

outTime

When sticky mode is select, on return, the maximum time between mouse-down and mouse-up. If the time between events is longer than this value, sticky mode should not be invoked. Pass `NULL` if you don't need this information.

outDistance

When sticky mode is select, on return, the maximum distance between mouse-down and mouse-up. If the distance between events is longer than this value, sticky mode should not be invoked. Pass `NULL` if you don't need this information.

Return value

A result code. See "Carbon Event Manager Result Codes" (page 204).

Discussion

Mouse tracking loops use different timeouts and wander distances to determine their behavior. This function provides a generic service for requesting this information.

Availability

Available in Mac OS X v10.3 and later.

Declared In

CarbonEvents.h

InstallEventHandler

Installs an event handler on a specified event target.

```
OSStatus InstallEventHandler (
    EventTargetRef inTarget,
    EventHandlerUPP inHandler,
    UInt32 inNumTypes,
    const EventTypeSpec * inList,
    void * inUserData,
    EventHandlerRef * outRef
);
```

Parameters

inTarget

The event target to register your handler with.

inHandler

A pointer to your event handler function.

inNumTypes

The number of events you are registering for.

inList

A pointer to an array of `EventTypeSpec` entries representing the events you are interested in.

inUserData

The value you pass in this parameter is passed to your event handler function when it is called.

outRef

On return, an event handler reference, which you can use later to remove the handler. You can pass `NULL` if you don't want the reference—when the target is disposed, the handler is disposed as well.

Return value

A result code. See "Carbon Event Manager Result Codes" (page 204).

Discussion

After being installed, your handler will be called when an event you registered for is sent to the specified event target. Note that CarbonEvents.h defines several macros which you can use for particular event classes. These macros simply combine the appropriate `GetxxxEventTarget` call with `InstallEventHandler`.

- `InstallApplicationEventHandler`
- `InstallWindowEventHandler`
- `InstallControlEventHandler`
- `InstallMenuEventHandler`
- `InstallHIObjectEventHandler` (in Mac OS X v10.2 and later)
- `HIViewInstallEventHandler` (in Mac OS X v10.2 and later)

Be sure to remove the event handler when you no longer need it by callling `RemoveEventHandler` (page 61). Doing so is especially important if the handler calls code that may disappear. For example, if a plugin installs an event handler and is later removed without removing the handler, the system may attempt to call back to the now nonexistent plugin code.

Availability

Available in Mac OS X v10.0 and later.

Declared In

CarbonEventsCore.h

InstallEventLoopIdleTimer

Installs a timer that fires only when there is no user activity.

```
OSStatus InstallEventLoopIdleTimer (
    EventLoopRef inEventLoop,
    EventTimerInterval inFireDelay,
    EventTimerInterval inInterval,
    EventLoopIdleTimerUPP inTimerProc,
    void * inTimerData,
    EventLoopTimerRef * outTimer
);
```

Parameters

inEventLoop

The event loop to add the timer.

inFireDelay
The delay before first firing this timer, in seconds. In Mac OS X v10.3 and earlier, this delay must be greater than zero. In Mac OS X v10.4 and later, the delay must be greater than or equal to zero. You cannot pass `kEventDurationForever`.

inInterval
The timer interval, in seconds. Pass `0` or `kEventDurationForever` for a one-shot timer.

inTimerProc
The function to call when the timer fires.

inTimerData
Data to pass to the timer function when called.

outTimer
A reference to the newly installed timer.

Return value
A result code. See "Carbon Event Manager Result Codes" (page 204).

Discussion
An idle timer is the same as a standard event timer except that it fires only when no user events are being received. That is, if the system receives no user events for the `inFireDelay` delay time, the idle timer fires, and will continue to fire at the rate specified by `inInterval`. If the user begins activity again, the timer stops and resets. For example, you could use an idle timer in a search engine to begin a search 2 seconds after the user stops typing in the search text field.

The callback function for idle timers takes an additional parameter that tells the callback the user status. See `EventLoopIdleTimerProcPtr` (page 75) and "Idle Timer Event Constants" (page 170) for more information.

Be sure to dispose of the timer when you no longer need it by calling `RemoveEventLoopTimer` (page 61). Doing so is especially important if your timer calls code that may no longer exist. For example, if a plugin creates a timer that calls back to it, the timer will attempt to call it even after the plugin is removed.

Availability
Available in Mac OS X v10.2 and later.

Declared In
CarbonEventsCore.h

InstallEventLoopTimer

Installs a timer.

```
OSStatus InstallEventLoopTimer (
    EventLoopRef inEventLoop,
    EventTimerInterval inFireDelay,
    EventTimerInterval inInterval,
    EventLoopTimerUPP inTimerProc,
    void * inTimerData,
    EventLoopTimerRef * outTimer
);
```

Parameters

inEventLoop

The event loop to add the timer.

inFireDelay

The delay before first firing this timer, in seconds. In Mac OS X v10.3 and earlier, the delay must be greater than zero. In Mac OS X v10.4, the delay can be greater than or equal to zero.

In Mac OS X and CarbonLib 1.5 and later, you may pass `kEventDurationForever` to stop the timer from firing at all until `SetEventLoopTimerNextFireTime` is used to start it; in earlier versions of CarbonLib, to achieve the same effect, just pass zero and then immediately call `SetEventLoopTimerNextFireTime( timer, (kEventDurationForever )` before returning control to your event loop.

inInterval

The timer interval, in seconds. Pass `0` or (in Mac OS X and CarbonLib 1.5 and later) `kEventDurationForever` for a one-shot timer.

inTimerProc

The function to call when the timer fires.

inTimerData

Data to pass to the timer function when called.

outTimer

A reference to the newly installed timer.

Return value

A result code. See "Carbon Event Manager Result Codes" (page 204).

Discussion

Installs a timer onto the event loop specified. The timer can either fire once or repeatedly at a specified interval depending on the parameters passed to this function. It executes at task level and should not be confused with Time Manager tasks or any other interrupt-level callback. This means you can call toolbox functions, allocate memory, and draw without worrying about consequences. When a timer fires, it calls the callback you specified when the timer was installed.

Timers in general have two uses: as a timeout mechanism and as a periodic task. An everyday example of using a timer for a timeout might be a light that goes out if no motion is detected in a room for 5 minutes. For this, you might install a timer which will fire in 5 minutes. If motion is detected, you would reset the timer fire time and let the clock start over. If no motion is detected for the full 5 minutes, the timer will fire and you could power off the light. A periodic timer is one that fires at regular intervals (say every second or so). You might use such a timer to blink the insertion point in your editor, and so on.

One advantage of timers is that you can install the timer right from the code that wants the time. For example, the standard editable text control can install a timer to blink the cursor when it's active, meaning that the Control Manager function `IdleControls` is a no-op for that control and doesn't need to be called. When the control is inactive, it removes its timer and doesn't waste CPU time in that state.

Currently, if you do decide to draw when your timer is called, be sure to save and restore the current port so that calling your timer doesn't inadvertently change the port out from under someone.

Be sure to dispose of the timer when you no longer need it by calling `RemoveEventLoopTimer` (page 61). Doing so is especially important if your timer calls code that may no longer exist. For example, if a plugin creates a timer that calls back to it, the timer will attempt to call it even after the plugin is removed.

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

Declared In
CarbonEventsCore.h

InstallStandardEventHandler

Installs the standard event handler for the specified target.

```
OSStatus InstallStandardEventHandler (
    EventTargetRef inTarget
);
```

Parameters
inTarget
The event target for which you want to install the standard handler.

Return value
A result code. See "Carbon Event Manager Result Codes" (page 204).

Discussion
Currently you can install the standard handler only for window event targets. To install the standard application handler, you must call `RunApplicationEventLoop` (page 63).

Note that events may also have default behaviors or standard definitions which define how and event is handled if you choose not to handle it yourself. Default behavior is the response that occurs whenever you choose not to handle the event, whether or not you have a standard handler installed. Standard definition behavior defines how an event is handled based on that element's standard definition. For example, the standard menu definition provides some default responses for menu events you do not handle. However if you are using your own custom definition, you cannot assume that these default responses will occur.

You can also install the standard handler for a window event target by calling `ChangeWindowAttributes` to set the `kWindowStandardHandlerAttribute` window attribute on the window.

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

Declared In
CarbonEventsCore.h

InvokeEventComparatorUPP

Calls an event comparator function through a UPP.

```
Boolean InvokeEventComparatorUPP (
    EventRef inEvent,
```

```
    void * inCompareData,
    EventComparatorUPP userUPP
);
```

Parameters

inEvent

The event to compare against.

inCompareData

Application-specific data. Typically this is the data you passed when calling `FindSpecificEventInQueue` (page 28) or `FlushSpecificEventsFromQueue` (page 30).

userUPP

A UPP to the comparator function you want to invoke.

Return value

Returns `true` if the comparator function indicates a match with the specified event, `false` otherwise.

Discussion

You call this function only if you need to invoke your event comparator callback yourself. In most cases you don't need to call this function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

CarbonEventsCore.h

InvokeEventHandlerUPP

Calls an event handler through a UPP.

```
OSStatus InvokeEventHandlerUPP (
    EventHandlerCallRef inHandlerCallRef,
    EventRef inEvent,
    void * inUserData,
    EventHandlerUPP userUPP
);
```

Return value

A result code. See "Carbon Event Manager Result Codes" (page 204).

Discussion

You use this function only if you need to invoke an event handler yourself. In most cases you don't need to call this function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

CarbonEventsCore.h

InvokeEventLoopIdleTimerUPP

Calls an event loop idle timer through a UPP.

```
void InvokeEventLoopTimerUPP (
    EventLoopTimerRef inTimer,
    EventLoopIdleTimerMessage inState,
    void * inUserData,
    EventLoopTimerUPP userUPP
);
```

Discussion

You use this function only if you need to invoke an idle event timer callback yourself. In most cases you don't need to call this function.

Availability

Available in Mac OS X v10.2 and later.

Declared In

CarbonEventsCore.h

InvokeEventLoopTimerUPP

Calls an event loop timer through a UPP.

```
void InvokeEventLoopTimerUPP (
    EventLoopTimerRef inTimer,
    void * inUserData,
    EventLoopTimerUPP userUPP
);
```

Discussion

You use this function only if you need to invoke an event timer callback yourself. In most cases you don't need to call this function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

CarbonEventsCore.h

IsEventInMask

Determines whether an event reference matches a `WaitNextEvent`-style event mask.

```
Boolean IsEventInMask (
    EventRef inEvent,
    EventMask inMask
);
```

Parameters

inEvent

The event reference to check against the event mask.

inMask

The mask to consider. See the Event Manager documentation for a description of the `EventMask` data type.

Return value

A Boolean whose value is `TRUE` if the event was in the mask; otherwise, `FALSE`.

Discussion

This is a companion function for `ConvertEventRefToEventRecord` (page 21), and is provided as a convenience function to help you if there are places in your application where you want to check an `EventRef` to see if it matches a classic `EventMask` bitfield. If the event matches, the function returns true.

Availability

Available in Mac OS X v10.0 and later.

Declared In

CarbonEvents.h

IsEventInQueue

Determines whether an event is in a particular queue.

```
Boolean IsEventInQueue (
    EventQueueRef inQueue,
    EventRef inEvent
);
```

Parameters

inQueue

The queue to check.

inEvent

The event in question.

Return value

Returns true if the specified event is posted to a queue.

Availability

Available in Mac OS X v10.0 and later.

Declared In

CarbonEventsCore.h

IsMouseCoalescingEnabled

Indicates whether mouse coalescing is enabled.

```
Boolean IsMouseCoalescingEnabled ();
```

Return value

A Boolean whose value is `TRUE` if mouse coalescing is on; otherwise, `FALSE`.

Discussion
If mouse coalescing is enabled, intermediate mouse movement events are merged into the most recent event, so that only one mouse moved or mouse dragged event is in the event queue at any time. For example, when the user moves the mouse across the screen, more mouse moved events are generated than most applications care about. Rather than place all these events in the queue (which would probably slow down the application), the Carbon Event Manager first checks to see if a mouse moved event already exists. If a mouse moved event already exists, that event is updated with the position and delta information from the more recently-generated event.

Availability
Available in Mac OS X v10.1 and later.

Declared In
CarbonEvents.h

IsSecureEventInputEnabled

Determines whether secure event input mode is enabled.

```
Boolean IsSecureEventInputEnabled (void);
```

Return value
A Boolean whose value is `TRUE` if secure event input mode is enabled; otherwise, `FALSE`.

Discussion
This function determines whether secure event input is enabled by *any* process, not just the current process. Secure event input may be disabled in the current process but enabled in another process, in which case, this function returns `TRUE`.

Availability
Available in Mac OS X v10.3 and later.

Declared In
CarbonEventsCore.h

IsUserCancelEventRef

Returns whether the specified event indicates the user wishes to cancel an operation.

```
Boolean IsUserCancelEventRef (
    EventRef event
);
```

Return value
A Boolean value indicating whether the event is a user cancel event.

Discussion
Tests the event given to see whether the event represents a user cancel event. Currently this is defined to be either the escape key being pressed or command-period being pressed.

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

Declared In
CarbonEvents.h

NewEventComparatorUPP

Creates an event comparator UPP.

```
EventComparatorUPP NewEventComparatorUPP (
    EventComparatorProcPtr userRoutine
);
```

Parameters

userRoutine
A pointer to your event comparator callback function.

Return value
The UPP for your callback function.

Discussion
When calling `FindSpecificEventInQueue` (page 28) or `FlushSpecificEventsFromQueue` (page 30), you must pass a universal procedure pointer (UPP) to your event comparator instead of a standard procedure pointer.

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

Declared In
CarbonEventsCore.h

NewEventHandlerUPP

Creates an event handler UPP.

```
EventHandlerUPP NewEventHandlerUPP (
    EventHandlerProcPtr userRoutine
);
```

Parameters

userRoutine
A pointer to your event handler.

Return value
The UPP for your event handler.

Discussion
When registering your event handler with `InstallEventHandler` (page 43), you must pass a universal procedure pointer (UPP) to your event handler instead of a standard procedure pointer.

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

Declared In
CarbonEventsCore.h

NewEventLoopIdleTimerUPP

Creates an event loop idle timer UPP.

```
EventLoopIdleTimerUPP NewEventLoopIdleTimerUPP (
    EventLoopIdleTimerProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your idle event timer callback function.

Return value

The UPP for your event loop idle timer callback function.

Availability

Available in Mac OS X v10.2 and later.

Declared In

CarbonEventsCore.h

NewEventLoopTimerUPP

Creates an event loop timer UPP.

```
EventLoopTimerUPP NewEventLoopTimerUPP (
    EventLoopTimerProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to your event timer callback function.

Return value

The UPP for your event timer callback function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

CarbonEventsCore.h

PopSymbolicHotKeyMode

Removes a hot key mode request from the hot key mode stack.

```
void PopSymbolicHotKeyMode (
    void* inToken
);
```

Parameters

inToken

The hot key mode token that was returned by a previous call to `PushSymbolicHotKeyMode` (page 55).

Discussion
If the request is the topmost request on the stack, the hot key mode changes to the next request on the stack. If there are other mode requests on top of this request on the stack, the mode does not change.

Availability
Available in Mac OS X v10.4 and later.

Declared In
CarbonEvents.h

PostEventToQueue

Adds an event to the specified event queue.

```
OSStatus PostEventToQueue (
    EventQueueRef inQueue,
    EventRef inEvent,
    EventPriority inPriority
);
```

Parameters

inQueue
The event queue to post the event onto.

inEvent
The event to post.

inPriority
The priority of the event. See "Event Priority Constants" (page 89) for a list of possible constants to pass.

Return value
A result code. See "Carbon Event Manager Result Codes" (page 204).

Discussion
Posts an event to the queue specified and increments its retain count. This automatically wakes up the event loop of the thread the queue belongs to.

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

Declared In
CarbonEventsCore.h

ProcessHICommand

Sends a command to the command chain.

```
OSStatus ProcessHICommand (
    const HICommand * inCommand
);
```

Return value
A result code. See "Carbon Event Manager Result Codes" (page 204)

Discussion
`ProcessHICommand` is a convenience function for sending a "process command" event through the command chain (for example, from menu to user focus to application). The command event is sent initially to either a menu (if the command represents a menu command) or the current user focus. If the function returns `eventNotHandledErr`, the command was not handled by any element in the chain.

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

Declared In
CarbonEvents.h

PushSymbolicHotKeyMode

Sets a new mode for enabling and disabling symbolic hot keys.

```
void* PushSymbolicHotKeyMode(
   OptionBits inOptions
);
```

Parameters
inOptions
The requested symbolic hot key mode. For details, see "Hot Key Constants" (page 133).

Return value
A token that is passed to `PopSymbolicHotKeyMode` (page 53) to remove this mode request when it is no longer needed.

Discussion
The Event Manager maintains a stack of hot key modes that have been requested by calls to this function. The most recently pushed mode is the mode that is currently in use.

Disabling hot keys can significantly affect the usability of Mac OS X. For this reason, applications are allowed to disable hot keys only if the "Enable access for assistive devices" checkbox is checked in the Universal Access preference pane. If this checkbox is not checked when this function is called, the requested hot key mode is pushed onto the mode stack and a valid token is returned but the actual hot key mode is unchanged.

If the frontmost application pushes a new hot key mode that disables any hot keys, the new mode is active only while the application remains the frontmost application. If the application is deactivated or exits without reenabling hot keys, the hot key mode automatically reverts to the previous mode.

Availability
Available in Mac OS X v10.4 and later.

Declared In
CarbonEvents.h

QuitApplicationEventLoop

Terminates the application event loop.

```
void QuitApplicationEventLoop ();
```

Discussion

This function is used to quit the `RunApplicationEventLoop` (page 63) function. Typically, your application doesn't need to call this. If your application has the Quit menu item tagged with the `kHICommandQuit` menu command ID, the toolbox will automatically call this for your application, automatically terminating your event loop. If your application wants to do pre-processing before the event loop exits, it should intercept either the `kHICommandQuit` menu command, or the `kEventAppQuit` event.

Availability

Available in Mac OS X v10.0 and later.

Declared In

CarbonEvents.h

QuitAppModalLoopForWindow

Quits the application-modal state for a window.

```
OSStatus QuitAppModalLoopForWindow (
    WindowRef inWindow
);
```

Parameters

inWindow

The window that is leaving the modal state. See the Window Manager documentation for a description of the `WindowRef` data type.

Return value

A result code. See "Carbon Event Manager Result Codes" (page 204).

Discussion

This function is used to quit a currently running call to `RunAppModalLoopForWindow` (page 63) (that is, it terminates a modal loop). Typically you call this from a handler you have installed on the modal window in question when the user clicks the appropriate button (Ok, Cancel, and so on).

Availability

Available in Mac OS X v10.0 and later.

Declared In

CarbonEvents.h

QuitEventLoop

Causes a specific event loop to terminate.

```
OSStatus QuitEventLoop (
    EventLoopRef inEventLoop
);
```

Parameters

inEventLoop

The event loop to terminate.

Return value

A result code. See "Carbon Event Manager Result Codes" (page 204).

Discussion

Usage of this is similar to `WakeUpProcess`, in that it causes the event loop specified to return immediately (as opposed to timing out). Typically you use this call in conjunction with `RunCurrentEventLoop` (page 64).

Availability

Available in Mac OS X v10.0 and later.

Declared In

CarbonEventsCore.h

ReceiveNextEvent

Waits for the next event of a specified type.

```
OSStatus ReceiveNextEvent (
    UInt32 inNumTypes,
    const EventTypeSpec * inList,
    EventTimeout inTimeout,
    Boolean inPullEvent,
    EventRef * outEvent
);
```

Parameters

inNumTypes

The number of event types to wait for (0 if any event should cause this function to return).

inList

The list of event types we are waiting for (pass `NULL` if any event should cause this function to return).

inTimeout

The time to wait before returning (passing `kEventDurationForever` is preferred).

inPullEvent

Pass `true` for this parameter to remove the next matching event from the queue.

outEvent

A pointer to the next event that matches the list passed in. If you passed `true` in the `inPullEvent` parameter, the event is owned by you, and you should release it when done.

Return value

A result indicating whether an event was received, the timeout expired, or the current event loop was quit. See "Carbon Event Manager Result Codes" (page 204) for possible values.

Discussion
This function tries to fetch the next event of a specified type. If no events in the event queue match, this function will run the current event loop until an event that matches arrives, or the timeout expires. Except for timers firing, your application is blocked waiting for events to arrive when inside this function.

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

Declared In
CarbonEventsCore.h

RegisterEventHotKey

Registers a global hot key.

```
OSStatus RegisterEventHotKey (
    UInt32 inHotKeyCode,
    UInt32 inHotKeyModifiers,
    EventHotKeyID inHotKeyID,
    EventTargetRef inTarget,
    OptionBits inOptions,
    EventHotKeyRef * outRef
);
```

Parameters
inHotKeyCode
The virtual key code of the hot key you want to register.

inHotKeyModifiers
The keyboard modifiers to look for. In Mac OS X v10.2 and earlier, if you do not specify a modifier key, this function returns `paramErr`. In Mac OS X v10.3 and later, passing `0` does not cause an error.

inHotKeyID
The application-specified hot key ID. You will receive this ID in the `kEventHotKeyPressed` event as the direct object parameter.

inTarget
The target to notify when the hot key is pressed.

inOptions
Currently unused. You must pass `0`.

outRef
On return, a reference to the new hot key. You need this reference if you later wish to unregister it.

Return value
A result code. See "Carbon Event Manager Result Codes" (page 204).

Discussion
This function registers a global hot key based on the virtual key code and modifiers you pass in. When the user enters the hot-key combination, a `kEventHotKeyPressed` event is sent to the target you specified. Only one such combination can exist for the current application (that is, multiple entities

in the same application cannot register for the same hot key combination). The same hot key can, however, be registered by multiple applications. This means that multiple applications can potentially be notified when a particular hot key is requested. This might not necessarily be desirable, but it is how it works at present.

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

Declared In
CarbonEvents.h

RegisterToolboxObjectClass

Registers events to be associated with a toolbox object. (**Deprecated.** Use the HIObject function `HIObjectRegisterSubclass` instead.)

```
OSStatus RegisterToolboxObjectClass (
    CFStringRef inClassID,
    ToolboxObjectClassRef inBaseClass,
    UInt32 inNumEvents,
    const EventTypeSpec * inEventList,
    EventHandlerUPP inEventHandler,
    void * inEventHandlerData,
    ToolboxObjectClassRef * outClassRef
);
```

Parameters

inClassID
: The class ID of the toolbox object you want to register. This value should be a Core Foundation string in the form `com.myCorp.myApp.myWidget`.

inBaseClass
: The class reference of this toolbox object's base class. Pass `NULL` if there is no base class.

inNumEvents
: The number of events to register for this object class.

inEventList
: An array of events you want to register for this object class. You define these events just as you would for any other Carbon event handler.

inEventHandler
: A universal procedure pointer to the event handler for this object class.

inEventHandlerData
: Any application-specific data you want passed to your event handler when it is called.

outClassRef
: On return, `outClassRef` contains a reference to the new object class. You use this value in your custom definition specification (such as a `ControlDefSpec` or `WindowDefSpec`) to define your new object class.

Return value
A result code. See "Carbon Event Manager Result Codes" (page 204).

Discussion
You use this function to register event handlers to implement what were formerly called defproc messages; that is, you can use toolbox objects in place of older custom window, menu, and control definitions.

Special Considerations
HIObject allows you to create subclasses that you can use for creating custom HIViews. HIViews support compositing and Quartz and provide an easier way to handle user elements in windows. Use `HIObjectRegisterSubclass` to create custom HIObjects and HIViews. See *HIView Programming Guide* for more details.

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

Declared In
CarbonEvents.h

ReleaseEvent
Releases, and possibly disposes of, the specified event.

```
void ReleaseEvent (
    EventRef inEvent
);
```

Parameters
inEvent
The event to release.

Discussion
This function decrements the reference count of an event. If the reference count reaches 0, the event is disposed.

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

Declared In
CarbonEventsCore.h

RemoveEventFromQueue
Removes an event from the event queue.

```
OSStatus RemoveEventFromQueue (
    EventQueueRef inQueue,
    EventRef inEvent
);
```

Parameters
inQueue
The queue to remove the event from.

inEvent
The event to remove.

Return value
A result code. See "Carbon Event Manager Result Codes" (page 204).

Discussion
Removes the given event from the queue on which it was posted and decrements its retain count.

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

Declared In
CarbonEventsCore.h

RemoveEventHandler

Removes the specified event handler.

```
OSStatus RemoveEventHandler (
    EventHandlerRef inHandlerRef
);
```

Parameters
inHandlerRef
The handler ref to remove (returned in a call to `InstallEventHandler`). After you call this function, the handler reference is considered invalid and can no longer be used.

Return value
A result code. See "Carbon Event Manager Result Codes" (page 204).

Discussion
Removes an event handler from the event target to which it was bound.

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

Declared In
CarbonEventsCore.h

RemoveEventLoopTimer

Removes the specified timer.

```
OSStatus RemoveEventLoopTimer (
    EventLoopTimerRef inTimer
);
```

Parameters
inTimer
The timer to remove.

Return value
A result code. See "Carbon Event Manager Result Codes" (page 204).

Discussion
Removes a timer that was previously installed by a call to `InstallEventLoopTimer` (page 45) or `InstallEventLoopIdleTimer` (page 44). You call this function when you are done using a timer.

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

Declared In
CarbonEventsCore.h

RemoveEventTypesFromHandler

Removes events from an installed event handler.

```
OSStatus RemoveEventTypesFromHandler (
    EventHandlerRef inHandlerRef,
    UInt32 inNumTypes,
    const EventTypeSpec * inList
);
```

Parameters

inHandlerRef
The event handler to remove the events from.

inNumTypes
The number of events to remove.

inList
A pointer to an array of `EventTypeSpec` structures.

Return value
A result code. See "Carbon Event Manager Result Codes" (page 204).

Discussion
You can use this function dynamically change which events you want your handler to respond to.

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

Declared In
CarbonEventsCore.h

RetainEvent

Increments the reference count of an event.

```
EventRef RetainEvent (
    EventRef inEvent
);
```

Parameters

inEvent
The event to retain.

Return value
The event reference you passed in the `inEvent` parameter. A value of `NULL` indicates an error condition.

Discussion
The `RetainEvent` function increments an event's reference count by 1. You can use this function to ensure that an event is never disposed of by another event handler. However, if the event system or some other event handler changes the event, those changes are reflected in your reference. To create a separate, unique copy of an event, use `CopyEvent` (page 22) instead.

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

Declared In
CarbonEventsCore.h

RunApplicationEventLoop

Runs the application event loop.

```
void RunApplicationEventLoop ();
```

Discussion
This function is used as the main event loop for a Carbon Event-based application. Once entered, this function waits for events to arrive and dispatches them to your event handlers automatically.

Note that calling `RunApplicationEventLoop` also installs the standard application handler, which provides standard handler responses for menu and application events.

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

Declared In
CarbonEvents.h

RunAppModalLoopForWindow

Puts the window in an application-modal state.

```
OSStatus RunAppModalLoopForWindow (
    WindowRef inWindow
);
```

Parameters

inWindow
The window you wish to behave modally. See the Window Manager documentation for a description of the `WindowRef` data type.

Return value
A result code. See "Carbon Event Manager Result Codes" (page 204).

Discussion
This function is used as a replacement for the Dialog Manager function `ModalDialog` to drive a Carbon Event-based modal dialog. Once called, this function will not exit until `QuitAppModalLoopForWindow` (page 56) is called.

While in the modal state, the standard toolbox dispatcher processes events only for the modal window and any that are above it (that is, closer to the front). This feature allows you to create stacked modal dialogs, if desired.

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

Declared In
CarbonEvents.h

RunCurrentEventLoop

Executes the event loop in the current thread.

```
OSStatus RunCurrentEventLoop (
    EventTimeout inTimeout
);
```

Parameters

inTimeout
The time to wait until returning (can be `kEventDurationForever`).

Return value
A result code. See "Carbon Event Manager Result Codes" (page 204).

Discussion
This function "runs" the event loop, returning only if aborted or the timeout specified is reached. The event loop is mostly blocked while in this function, occasionally waking up to fire timers or pick up events. The typical use of this function is to cause the current thread to wait for some operation to complete, most likely on another thread of execution.

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

Declared In
CarbonEventsCore.h

SendEventToEventTarget

Sends an event to the specified event target.

```
OSStatus SendEventToEventTarget (
    EventRef inEvent,
    EventTargetRef inTarget
);
```

Parameters

inEvent
The event to send.

inTarget
The target to send it to.

Return value
A result code. See "Carbon Event Manager Result Codes" (page 204).

Discussion
If you are creating your own events, you can dispatch them immediately to an event target by calling this function.

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

Declared In
CarbonEventsCore.h

SendEventToEventTargetWithOptions

Sends an event to the specified event target with propagation options.

```
OSStatus SendEventToEventTargetWithOptions (
    EventRef inEvent,
    EventTargetRef inTarget
    OptionBits inOptions
);
```

Parameters

inEvent
The event to send.

inTarget
The target to send it to.

inOptions
Options indicating how the event should be propagated. See "Event Target Propagation Options" (page 90) for a list of possible values.

Return value
A result code. See "Carbon Event Manager Result Codes" (page 204).

Discussion
This function is identical to `SendEventToEventTarget` (page 64) except that you can specify how the event is propagated using options.

Availability
Available in Mac OS X v10.2 and later.

Declared In
CarbonEventsCore.h

SetEventLoopTimerNextFireTime

Sets the next time that the specified timer will fire.

```
OSStatus SetEventLoopTimerNextFireTime (
    EventLoopTimerRef inTimer,
    EventTimerInterval inNextFire
```

```
);
```

Parameters

inTimer

The timer whose firing time you want to set.

inNextFire

The interval from the current time to wait until firing the timer again.

Return value

A result code. See "Carbon Event Manager Result Codes" (page 204).

Discussion

This function is used to "reset" a timer. It controls the next time the timer fires. This will override any interval you might have set. For example, if you have a timer that fires every second, and you call this function setting the next time to 5 seconds from now, the timer will sleep for 5 seconds, then fire. The timer will then resume its one second interval. This function acts as if you removed the timer and reinstalled it with a new first-fire delay.

Availability

Available in Mac OS X v10.0 and later.

Declared In

CarbonEventsCore.h

SetEventParameter

Sets a parameter associated with a particular event.

```
OSStatus SetEventParameter (
    EventRef inEvent,
    EventParamName inName,
    EventParamType inType,
    UInt32 inSize,
    const void * inDataPtr
);
```

Parameters

inEvent

The event to set the data for.

inName

The symbolic name of the parameter.

inType

The symbolic type of the parameter.

inSize

The size of the parameter data.

inDataPtr

A pointer to the parameter data.

Return value

A result code. See "Carbon Event Manager Result Codes" (page 204).

Discussion
When creating events, you may want to specify additional event-related information, such as the mouse location or the window in which the event occurred. To set these you call `SetEventParameter`, specifying the type and value for the desired parameter.

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

Declared In
CarbonEventsCore.h

SetEventTime

Sets the event time for a given event.

```
OSStatus SetEventTime (
    EventRef inEvent,
    EventTime inTime
);
```

Parameters
inEvent
The event in question.

inTime
The new time.

Return value
A result code. See "Carbon Event Manager Result Codes" (page 204).

Discussion
This function allows you to set the time of a given event, if you so desire. In general, you would never use this function, except for those special cases where you reuse an event from time to time instead of creating a new event each time.

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

Declared In
CarbonEventsCore.h

SetMouseCoalescingEnabled

Turns mouse coalescing on or off.

```
OSStatus SetMouseCoalescingEnabled (
    Boolean inNewState,
    Boolean * outOldState
);
```

Parameters
inNewState
Pass `true` to turn mouse coalescing on, `false` otherwise.

outOldState

A Boolean value indicating the previous mouse coalescing state (that is, before you called this function to set it). You can use this value if you want to save the previous state for later restoration. If you don't need this state information, pass `NULL`.

Return value
A result code. See "Carbon Event Manager Result Codes" (page 204).

Discussion
See `IsMouseCoalescingEnabled` (page 50) for a definition of mouse coalescing.

Availability
Available in Mac OS X v10.1 and later.

Declared In
CarbonEvents.h

SetUserFocusWindow

Designates a window to receive user focus.

```
OSStatus SetUserFocusWindow (
    WindowRef inWindow
);
```

Return value
A result code. See "Carbon Event Manager Result Codes" (page 204).

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 you pass `kUserFocusAuto` in the `inWindow` parameter, the system picks the best candidate for user focus. 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.

Declared In
CarbonEvents.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. See the Window Manager documentation for a description of the `WindowRef` data type.

inControl

The control to designate as the Cancel button. See the Control Manager documentation for a description of the `ControlRef` data type.

Return value

A result code. See "Carbon Event Manager Result Codes" (page 204).

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.

Declared In

CarbonEvents.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. See the Window Manager documentation for a description of the `WindowRef` data type.

inControl

The control to designate as the default. See the Control Manager documentation for a description of the `ControlRef` data type.

Return value

A result code. See "Carbon Event Manager Result Codes" (page 204).

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.

Declared In
CarbonEvents.h

TrackMouseLocation

Tracks the mouse, blocking your application when there is no activity.

```
OSStatus TrackMouseLocation (
    GrafPtr inPort,
    Point * outPt,
    MouseTrackingResult * outResult
);
```

Parameters

inPort

The grafport to consider for mouse coordinates. You can pass `NULL` for this parameter to indicate the current port. The mouse location is returned in terms of local coordinates of this port. See the QuickDraw Manager documentation for a description of the `GrafPtr` data type.

outPt

On exit, a pointer to the mouse location from the last mouse event that caused this function to exit.

outResult

On exit, a pointer to a value representing what kind of event was received that cause the function to exit, such as `kMouseTrackingMouseReleased`.

Return value
A result code. See "Carbon Event Manager Result Codes" (page 204).

Discussion
Once entered, this function waits for certain mouse events (move, mouse down, mouse up). When one of these events occurs, the function returns and tells the caller what happened and where the mouse is currently located. While there is no activity, the current event loop is run, effectively blocking the current thread (save for any timers that fire). This helps to minimize CPU usage when there is nothing going on.

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

Declared In
CarbonEvents.h

TrackMouseLocationWithOptions

Tracks the mouse with additional options.

```
OSStatus TrackMouseLocationWithOptions (
    GrafPtr inPort,
    OptionBits inOptions,
    EventTimeout inTimeout,
    Point * outPt,
    UInt32 * outModifiers,
    MouseTrackingResult * outResult
);
```

Parameters

inPort

The graphics port (`GrafPort`) to consider for mouse coordinates. You can pass `NULL` for this parameter to indicate the current port. The mouse location is returned in global coordinates. See the QuickDraw Manager documentation for a description of the `GrafPtr` data type.

inOptions

The only option supported by this function at present is the option to have the toolbox leave mouse up events in the queue, rather than pulling them (which is the default). See "Mouse Tracking Option Constant" (page 154) for more information.

inTimeout

The amount of time to wait for an event. If no events arrive within this time, `kMouseTrackingTimedOut` is returned in `outResult`.

outPt

On return, a pointer to the mouse location from the last mouse event that caused this function to exit. If a timeout or key modifiers changed event caused this function to exit, the current mouse position at the time is returned.

outModifiers

On return, a pointer to the most recent state of the keyboard modifiers.

outResult

On return, a pointer to a value indicating the kind of event that caused the function to exit, such as `kMouseTrackingMouseReleased`.

Return value

A result code. See "Carbon Event Manager Result Codes" (page 204).

Discussion

Once entered, this function waits for certain mouse events (move, mouse down, mouse up). When one of these events occurs, the function returns and tells the caller what happened and where the mouse is currently located. While there is no activity, the current event loop is run, effectively blocking the current thread (save for any timers that fire). This helps to minimize CPU usage when there is nothing going on.

Availability

Available in Mac OS X v10.0 and later.

Declared In

CarbonEvents.h

TrackMouseRegion

Tracks the mouse within a region.

```
OSStatus TrackMouseRegion (
    GrafPtr inPort,
    RgnHandle inRegion,
    Boolean * ioWasInRgn,
    MouseTrackingResult * outResult
);
```

Parameters

inPort

The graphics port to consider for mouse coordinates. You can pass `NULL` for this parameter to indicate the current port. See the QuickDraw Manager documentation for a description of the `GrafPtr` data type.

inRegion

The region to consider. This should be in the coordinates of the port you passed to `inPort`. See the QuickDraw Manager documentation for a description of the `RgnHandle` data type.

ioWasInRgn

On entering the region, this parameter should be set to `true` if the mouse is currently inside the region passed in `inRegion`, or `false` if the mouse is currently outside the region. On exit, this parameter is updated to reflect the current reality. For example, if the `outResult` parameter returns `kMouseTrackingMouseExited`, `ioWasInRgn` will be set to `false` when this function exits. Because it is updated from within, you should only need to set this yourself before the first call to this function in your tracking loop.

outResult

On exit, a pointer to a value indicating the kind of event that caused the function to exit, such as `kMouseTrackingMouseEntered`.

Return value

A result code. See "Carbon Event Manager Result Codes" (page 204).

Discussion

This function is largely identical to `TrackMouseLocation` (page 70). The difference between `TrackMouseLocation` and `TrackMouseRegion` is that `TrackMouseRegion` only returns when the mouse enters or exits a specified region that you pass in to the function, as opposed to whenever the mouse moves (it also returns for mouse up/down events). This is useful if you don't need to know intermediate mouse events, but rather just if the mouse enters or leaves an area.

Note that in some cases you may prefer to register one or more special mouse tracking regions and receive events when the mouse enters or exits the region. However, this alternative method does not automatically inform you about mouse up and mouse down actions. See `CreateMouseTrackingRegion` (page 208) for more details.

Availability

Available in Mac OS X v10.0 and later.

Declared In

CarbonEvents.h

UnregisterEventHotKey

Unregisters a global hot key.

```
OSStatus UnregisterEventHotKey (
    EventHotKeyRef inHotKey
);
```

Parameters

inHotKey

The event hot key reference of the hot key you want to unregister.

Return value

A result code. See "Carbon Event Manager Result Codes" (page 204).

Discussion

Unregisters a global hot key that was previously registered with the function `RegisterEventHotKey` (page 58). You do not need to unregister a hot key when your application terminates; the system takes care of that for you. You can use this function if the user changes a hot key for something in your application—you would unregister the previous key and register your new key.

Availability

Available in Mac OS X v10.0 and later.

Declared In

CarbonEvents.h

UnregisterToolboxObjectClass

Unregisters events for a given toolbox object class (**Deprecated.** Use the HIObject function `HIObjectUnregisterClass` instead.)

```
OSStatus UnregisterToolboxObjectClass (
    ToolboxObjectClassRef inClassRef
);
```

Parameters

inClassRef

A reference to the toolbox object class you want to unregister.

Return value

A result code. See "Carbon Event Manager Result Codes" (page 204).

Special Considerations

HIObject allows you to create subclasses that you can use for creating custom HIViews. HIViews support compositing and Quartz and provide an easier way to handle user elements in windows. Use `HIObjectUnregisterClass` to unregister custom HIObjects and HIViews. See *HIView Programming Guide* for more details.

Availability

Available in Mac OS X v10.0 and later.

Declared In

CarbonEvents.h

Callbacks

EventComparatorProcPtr

Defines the format of your event comparator callback function.

```
typedef Boolean (*EventComparatorProcPtr) (
   EventRef inEvent,
   void * inCompareData
);
```

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

```
Boolean MyEventComparatorProc (
   EventRef inEvent,
   void * inCompareData
);
```

Parameters

inEvent

The event to compare.

inCompareData

The data you passed to `FindSpecificEventInQueue` (page 28) or `FlushSpecificEventsFromQueue` (page 30).

Return value

A Boolean value indicating whether the event matches (`true`) or not (`false`).

Discussion

You use this callback function when searching the event queue using functions such as `FindSpecificEventInQueue` (page 28).

EventHandlerProcPtr

Defines the format of your event handler.

```
typedef OSStatus (*EventHandlerProcPtr) (
   EventHandlerCallRef inHandlerCallRef,
   EventRef inEvent,
   void * inUserData
);
```

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

```
OSStatus MyEventHandlerProc (
   EventHandlerCallRef inHandlerCallRef,
   EventRef inEvent,
   void * inUserData
);
```

Parameters

inHandlerCallRef

A reference to the current handler call chain. This is passed to your handler so that you can call `CallNextEventHandler` if you need to.

inEvent

The event that triggered this call.

inUserData

The application-specific data you passed in to `InstallEventHandler` (page 43).

Return value

A result code. See "Carbon Event Manager Result Codes" (page 204). Returning `noErr` indicates you handled the event. Returning `eventNotHandledErr` indicates you did not handle the event and perhaps other handlers in the calling chain should take action.

Discussion

Callback to install on an event target.

EventLoopIdleTimerProcPtr

Defines the format of your idle timer callback function.

```
typedef void (*EventLoopIdleTimerProcPtr) (
   EventLoopTimerRef inTimer,
   EventLoopIdleTimerMessage inState,
   void * inUserData
);
```

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

```
void MyEventLoopTimerProc (
   EventLoopTimerRef inTimer,
   EventLoopIdleTimerMessage inState,
   void * inUserData
);
```

Parameters

inTimer

The timer that fired.

inState

The state of the idle period. See "Idle Timer Event Constants" (page 170) for a list of possible constants you can receive.

inUserData

The application-specific data you passed into `InstallEventLoopIdleTimer` (page 44).

Discussion

Called when an idle timer fires.

EventLoopTimerProcPtr

Defines the format of your event loop timer callback function.

```
typedef void (*EventLoopTimerProcPtr) (
   EventLoopTimerRef inTimer,
   void * inUserData
);
```

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

```
void MyEventLoopTimerProc (
   EventLoopTimerRef inTimer,
   void * inUserData
);
```

Parameters

inTimer

The timer that fired.

inUserData

The data you passed into `InstallEventLoopTimer` (page 45).

Discussion

Called when a timer fires.

Data Types

EventClassID

Represents an event class ID.

```
typedef UInt32 EventClassID;
```

Availability

Available in Mac OS X v10.0 and later.

EventComparatorUPP

Represents a universal procedure pointer to an event comparator callback function.

```
typedef EventComparatorProcPtr EventComparatorUPP
```

EventHandlerCallRef

Indicates the next handler in the event handler calling hierarchy.

```
typedef struct OpaqueEventHandlerCallRef * EventHandlerCallRef;
```

Discussion
This structure is passed to your event handler, which can then choose to pass control to the next handler in the calling hierarchy (such as a standard event handler). Doing so is a convenient way to add pre- or post-processing to the standard event handler. See the `CallNextEventHandler` (page 20) function for more information.

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

EventHandlerUPP

Represents a universal procedure pointer for an event handler callback function.

```
typedef EventHandlerProcPtr EventHandlerUPP;
```

EventLoopTimerUPP

Represents a universal procedure pointer for an event timer callback function.

```
typedef EventLoopTimerProcPtr EventLoopTimerUPP;
```

EventLoopIdleTimerUPP

Represents a universal procedure pointer for an idle event timer callback function.

```
typedef EventLoopIdleTimerProcPtr EventLoopIdleTimerUPP;
```

EventHandlerRef

Represents an installed event handler.

```
typedef struct OpaqueEventHandlerRef * EventHandlerRef;
```

Discussion
You receive an event handler reference when you install your handler using `InstallEventHandler` (page 43). You can use this reference when calling functions such as `RemoveEventHandler` (page 61) and `AddEventTypesToHandler` (page 19).

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

EventHotKeyID

Represents the ID of a global hot key.

```
struct EventHotKeyID {
    OSType signature;
    UInt32 id;
};
typedef struct EventHotKeyID EventHotKeyID;
```

Discussion
You register a hot key using the `RegisterEventHotKey` (page 58) function.

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

EventHotKeyRef

Represents a registered global hot key.

```
typedef struct OpaqueEventHotKeyRef * EventHotKeyRef;
```

Discussion
You register a hot key using the `RegisterEventHotKey` (page 58) function.

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

EventLoopIdleTimerMessage

Represents an idle timer message.

```
typedef UInt16 EventLoopIdleTimerMessage;
```

Discussion
Sent to idle timer callback functions to indicate the current idle status. See "Idle Timer Event Constants" (page 170) for a list of possible values.

Availability
Available in Mac OS X v10.2 and later.

EventLoopRef

Represents an event loop.

```
typedef struct OpaqueEventLoopRef * EventLoopRef;
```

Discussion
The `EventLoopRef` type represents an event loop, which is the conceptual entity that you run to fetch events from hardware and other sources and also fires timers that might be installed with `InstallEventLoopTimer` (page 45) or `InstallEventLoopIdleTimer` (page 44). The term "run" is a bit of a misnomer, as the event loop's goal is to stay as blocked as possible to minimize CPU usage for the current application. The event loop is run implicitly through calls to functions like `ReceiveNextEvent` (page 57), `RunApplicationEventLoop` (page 63), or even the Classic Event Manager function `WaitNextEvent`. It can also be run explicitly through a call to `RunCurrentEventLoop` (page 64). Each preemptive thread can have an event loop. Cooperative threads share the main thread's event loop.

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

EventLoopTimerRef

Represents an installed event timer.

```
typedef struct __EventLoopTimer * EventLoopTimerRef;
```

Discussion

The `EventLoopTimerRef` type represents a timer function that is called either once or at regular intervals. See `InstallEventLoopTimer` (page 45) and `InstallEventLoopIdleTimer` (page 44) for more information about event timers.

Availability

Available in Mac OS X v10.0 and later.

EventParamName

Represents an event parameter constant.

```
typedef OSType EventParamName;
```

Discussion

You specify an event parameter name when calling `GetEventParameter` (page 36) or `SetEventParameter` (page 66). Parameter names indicate what kind of event parameter you want to set or obtain (such as `kEventParamDirectObject`). For specific types, see the tables of event parameters and types associated with each class of events (for example, Table 8 (page 149)).

Availability

Available in Mac OS X v10.0 and later.

EventParamType

Represents an event parameter type constant.

```
typedef OSType EventParamType;
```

Discussion

You specify an event parameter type when calling `GetEventParameter` (page 36) or `SetEventParameter` (page 66). Event parameter types indicate the data type of the parameter you want to set or obtain (such as `typeBoolean`). For specific types, see the tables of event parameters and types associated with each class of events (for example, Table 11 (page 178)).

Availability

Available in Mac OS X v10.0 and later.

EventQueueRef

Represents an event queue.

```
typedef struct OpaqueEventQueueRef * EventQueueRef;
```

Availability

Available in Mac OS X v10.0 and later.

EventRef

Represents an opaque data structure that identifies individual events.

```
typedef struct OpaqueEventRef * EventRef;
```

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

EventTargetRef

Represents an event target (such as a window or control).

```
typedef struct OpaqueEventTargetRef * EventTargetRef;
```

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

EventTime

Represents a time value in seconds. An absolute `EventTime` value is seconds since boot time.

```
typedef double EventTime;
```

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

EventTimeout

Represents a timeout interval, in seconds.

```
typedef EventTime EventTimeout;
```

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

EventTimerInterval

Specifies the period of an event timer, in seconds.

```
typedef EventTime EventTimerInterval;
```

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

EventType

Represents an event type.

```
typedef UInt32 EventType;
```

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

EventTypeSpec

Describes the class and kind of an event.

```
struct EventTypeSpec {
    UInt32 eventClass;
    UInt32 eventKind;
};
typedef struct EventTypeSpec EventTypeSpec;
```

Discussion
This structure is used to specify an event. Typically, you pass a static array of `EventTypeSpec` structures into functions such as `InstallEventHandler` (page 43), as well as functions such as `FlushEventsMatchingListFromQueue` (page 29).

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

HICommand

Represents a command event; this structure has been superseded by the `HICommandExtended` structure.

```
struct HICommand {
    UInt32 attributes
    UInt32 commandID
    struct {
        MenuRef menuRef;
        MenuItemIndex menuItemIndex;
    } menu;
};
typedef struct HICommand HICommand;
```

Fields
`attributes`
Attributes of the command event.

`commandID`
The command ID of the command event.

`menuRef`
A reference to the menu containing the HICommand.

`menuItemIndex`
The index number of the menu item containing the HICommand.

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

HICommandExtended

Represents an extended command event.

```
struct HICommandExtended {
    UInt32 attributes;
    UInt32 commandID;
    union {
        controlRef control;
        windowRef window;
        struct {
            MenuRef menuRef;
            MenuItemIndex menuItemIndex;
        } menu;
    } source;
};
typedef struct HICommandExtended HICommandExtended;
```

Fields

`attributes`
: Attributes of the command event. The value of this field (indicating whether the source of the command event is a control, window, or menu) determines what reference is stored in the union. See "Command Event Source Constants" (page 109) for a list of possible values.

`commandID`
: The command ID of the command event.

`controlRef`
: The control that produced the command event.

`windowRef`
: The window that produced the command event.

`menuRef`
: A reference to the menu containing the command event.

`menuItemIndex`
: The index number of the menu item containing the command event.

Discussion

The `HICommandExtended` structure was introduced in Mac OS X v10.2 and CarbonLib 1.6. Because the `HICommand` and `HICommandExtended` structures are exactly the same size and have the same fields at the same offsets, you can use an `HICommandExtended` structure at runtime while running on any version of CarbonLib or Mac OS X. The only difference is that the `HICommandExtended` structure has a union that allows you to get type-safe access to the source object. The originator of the command determines whether the structure actually contains a ControlRef, WindowRef, MenuRef, or nothing at all. You can determine what is in the command by checking the `attributes` field.

For example, in Mac OS X v10.2 and later, when a push button is clicked, the Control Manager sends a command event containing the push button's command ID, sets the `kHICommandFromControl` bit in the `attributes` field, and stores the button's ControlRef in the `source.control` field. In Mac OS X v10.0 and v10.1, the same command event is sent, but the `kHICommandFromControl`, `kHICommandFromMenu`, and `kHICommandFromWindow` attributes are not set, and the `source.controlRef`, `source.menu.menuRef` and `source.windowRef` fields are not initialized, respectively. Your code can use an `HICommandExtended` structure when running on Mac OS X v10.0 and v10.1 as long as it first checks the `kHICommandFromControl`, `kHICommandFromMenu`, and `kHICommandFromWindow` attributes before accessing the `source.control`, `menu.control`, and `window.control` fields.

Availability
Available in Mac OS X v10.2 and later.

MouseTrackingRef

Represents a mouse tracking region

```
typedef struct OpaqueMouseTrackingRef * MouseTrackingRef;
```

Discussion
Use `CreateMouseTrackingRegion` (page 208) to create a mouse tracking region.

Availability
Available in Mac OS X v10.2 and later.

MouseTrackingRegionID

Represents a mouse tracking region identifier.

```
struct MouseTrackingRegionID {
    OSType signature;
    SInt32 id;
};
typedef struct MouseTrackingRegionID MouseTrackingRegionID;
```

Fields
`signature`

A four-character code (such as `'moof'`)that uniquely identifies the application that owns this mouse tracking region.

`id`

An integer that identifies the mouse tracking region in this application.

Discussion
Each application can register multiple mouse tracking regions as long as each region has a unique ID. Use `CreateMouseTrackingRegion` (page 208) to create a mouse tracking region.

Availability
Available in Mac OS X v10.2 and later.

TabletPointRec

Defines a tablet point structure.

```
struct TabletPointRec {
    SInt32 absX;
    SInt32 absY;
    SInt32 absZ;
    UInt16 buttons;
    UInt16 pressure;
    SInt16 tiltX;
    SInt16 tiltY;
    UInt16 rotation;
```

```
    SInt16 tangentialPressure;
    UInt16 deviceID;
    SInt16 vendor1;
    SInt16 vendor2;
    SInt16 vendor3;
};
typedef struct TabletPointRec TabletPointRec;
typedef TabletPointRec TabletPointerRec;
```

Fields

`absX`
: The x-coordinate of the pointer, in tablet space (at full tablet resolution).

`absY`
: The y-coordinate of the pointer, in tablet space (at full tablet resolution).

`absZ`
: The z-coordinate of the pointer, in tablet space (at full tablet resolution).

`buttons`
: The buttons that are pressed. This integer is interpreted as a bit field, with bit 0 indicating the first button, bit 1 the second button, and so on. A value of `1` indicates that the button is down.

`pressure`
: The scaled pressure value. The pressure value is in the range 0 to 65535.

`tiltX`
: The scaled tilt x value. The tilt value is in the range -32767 to 32767.

`tiltY`
: The scaled tilt y value. The tilt value is in the range -32767 to 32767.

`rotation`
: The device rotation as a fixed-point value in a 10.6 format.

`tangentialPressure`
: The tangential pressure on the device. This pressure is in the range -32767 to 32767.

`deviceID`
: A unique system-assigned device ID. This ID matches the device ID you receive for the `kEventTabletProximity` event.

`vendor1`
: A vendor-defined value.

`vendor2`
: A vendor-defined value.

`vendor3`
: A vendor-defined value.

Discussion

You receive this structure in the `kEventParamTabletPointRec` parameter for the `kEventTabletPoint` event.

Availability

Available in Mac OS X v10.1 and later.

TabletProximityRec

Defines a tablet proximity structure.

```
struct TabletProximityRec {
    UInt16 vendorID;
    UInt16 tabletID;
    UInt16 pointerID;
    UInt16 deviceID;
    UInt16 systemTabletID;
    UInt16 vendorPointerType;
    UInt32 pointerSerialNumber;
    UInt64 uniqueID;
    UInt32 capabilityMask;
    UInt8 pointerType;
    UInt8 enterProximity;
};
typedef struct TabletProximityRec TabletProximityRec;
```

Fields

`vendorID`
: A vendor-defined ID. This value is typically the USB vendor ID.

`tabletID`
: A vendor-defined ID for the tablet. This value is typically the USB product ID for the tablet.

`pointerID`
: A vendor-defined ID for the pointing device (for example, a pen).

`deviceID`
: A unique system-assigned device ID. This ID matches the device ID you receive for the `kEventTabletPoint` event.

`systemTabletID`
: A system-assigned unique tablet ID.

`vendorPointerType`
: A vendor-defined pointer type.

`pointerSerialNumber`
: A vendor-defined serial number for the pointing device.

`uniqueID`
: A vendor-defined ID for this pointer.

`capabilityMask`
: A bit mask representing the capabilities of this device.

`pointerType`
: The type of pointing device.

`enterProximity`
: The proximity value. A nonzero value indicates that the pointer is entering the tablet proximity; zero indicates that it is leaving.

Discussion

You receive this structure in the `kEventParamTabletProximityRec` parameter for the `kEventTabletProximity` event.

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

ToolboxObjectClassRef

Represents a toolbox object class.

```
typedef struct OpaqueToolboxObjectClassRef * ToolboxObjectClassRef;
```

Discussion
Typically you use toolbox object classes to specify custom user interface elements. See `RegisterToolboxObjectClass` (page 59) for more information.

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

Constants

Basic Event Constants

Event Class Constants

Define constants for specifying event classes.

```
typedef UInt32 EventClass;
enum {
    kEventClassMouse = 'mous',
    kEventClassKeyboard = 'keyb',
    kEventClassTextInput = 'text',
    kEventClassApplication = 'appl',
    kEventClassAppleEvent = 'eppc',
    kEventClassMenu = 'menu',
    kEventClassWindow = 'wind',
    kEventClassControl = 'cntl',
    kEventClassCommand = 'cmds',
    kEventClassTablet = 'tblt',
    kEventClassVolume = 'vol ',
    kEventClassAppearance = 'appm',
    kEventClassService = 'serv',
    kEventClassToolbar = 'tbar',
    kEventClassToolbarItem = 'tbit',
    kEventClassToolbarItemView = 'tbiv',
    kEventClassAccessibility = 'acce'.
    kEventClassSystem = 'macs',
    kEventClassInk = 'ink ',
    kEventClassTSMDocumentAccess = 'tdac'
};
```

Constants

`kEventClassMouse`
Events related to the mouse (mouse down/up/moved).

Available in Mac OS X v10.0 and later.

`kEventClassKeyboard`
Events related to the keyboard.

Available in Mac OS X v10.0 and later.

`kEventClassTextInput`
Events related to text input (by keyboard or by input method).

Available in Mac OS X v10.0 and later.

`kEventClassApplication`
Application-level events (launch, quit, and so on).

Available in Mac OS X v10.0 and later.

`kEventClassAppleEvent`
Apple Events.

Available in Mac OS X v10.0 and later.

`kEventClassMenu`
Menu-related events.

Available in Mac OS X v10.0 and later.

`kEventClassWindow`
Window-related events.

Available in Mac OS X v10.0 and later.

`kEventClassControl`
Control-related events.

Available in Mac OS X v10.0 and later.

`kEventClassCommand`
Command events (HICommands).

Available in Mac OS X v10.0 and later.

`kEventClassTablet`
Events related to tablet input.

Available in Mac OS X v10.0 and later.

`kEventClassVolume`
Events related to File Manager volumes.

Available in Mac OS X v10.0 and later.

`kEventClassAppearance`
Events related to the Appearance Manager.

Available in Mac OS X v10.1 and later.

`kEventClassService`
Events related to the Services Manager.

Available in Mac OS X v10.1 and later.

`kEventClassToolbar`
Events related to the toolbar (not the toolbar window class).

Available in Mac OS X v10.2 and later.

`kEventClassToolbarItem`
Events related to toolbar items.

Available in Mac OS X v10.2 and later.

`kEventClassToolbarItemView`
Events related to toolbar item views.

Available in Mac OS X v10.3 and later.

`kEventClassAccessibility`
Events related to application accessibility features.

Available in Mac OS X v10.2 and later.

`kEventClassSystem`
Events related to the system.

Available in Mac OS X v10.3 and later.

`kEventClassInk`
Events related to ink.

Available in Mac OS X v10.3 and later.

`kEventClassTSMDocumentAccess`
Events related to Text Services Manager document access.

Available in Mac OS X v10.3 and later.

Discussion
Event classes specify broad categories of events, grouped according to the object they are associated with. Within an event class are specific event types.

Event Attributes

Define constants for special attributes of an event.

```
typedef UInt32 EventAttributes;
enum {
    kEventAttributeNone = 0,
    kEventAttributeUserEvent = (1 << 0),
    kEventAttributeMonitored= 1 << 3
};
```

Constants

`kEventAttributeNone`
No attributes.

Available in Mac OS X v10.0 and later.

`kEventAttributeUserEvent`
An event generated in response to a user action.

Available in Mac OS X v10.0 and later.

`kEventAttributeMonitored`
An event that was not originally targeted to this process but has been provided to this process because an event handler for this event type has been installed on the event monitoring target. The event dispatcher sends events with this attribute directly to the event monitor target. (Available in Mac OS X v10.3 and later.)

Available in Mac OS X v10.3 and later.

Discussion
You use these attributes only if you are creating your own events.

Event Priority Constants

Define event priority constants.

```
typedef SInt16 EventPriority;
enum {
    kEventPriorityLow = 0,
    kEventPriorityStandard = 1,
    kEventPriorityHigh = 2
};
```

Constants

`kEventPriorityLow`
Lowest priority. Currently only window update events are posted at this priority.

Available in Mac OS X v10.0 and later.

`kEventPriorityStandard`
Normal priority of events. Most events are standard priority.

Available in Mac OS X v10.0 and later.

`kEventPriorityHigh`
Highest priority.

Available in Mac OS X v10.0 and later.

Discussion
These values define the relative priority of an event, and are used when posting events with `PostEventToQueue` (page 54). In general events are pulled from the queue in order of first posted to last posted. These priorities are a way to alter that behavior when posting events. You can post a standard priority event and then a high priority event, and the high priority event will be pulled from the queue first.

Event Target Propagation Options

Define options for the `SendEventToEventTargetWithOptions` function.

```
enum {
    kEventTargetDontPropagate = (1 << 0),
    kEventTargetSendToAllHandlers = (1 << 1)
};
```

Constants

`kEventTargetDontPropagate`

Do not propagate this event to any other event target. That is, even if the handler returns `eventNotHandledErr`, the event is not propagated up the handler chain. When passed an event sent with this option, `CallNextEventHandler` only calls other event handlers installed on the current event target; it does not propagate the event to other event targets.

Available in Mac OS X v10.2 and later.

`kEventTargetSendToAllHandlers`

Send this event to all event targets in the handler chain, regardless of any handler's return value. For example, if sent to a control, after returning, the event is sent to the owning window and then to the application. Note that the Carbon Event Manager keeps track of the strongest result code when progressing up the handler chain. That is, if the first handler returns `noErr`, and the second handler returns `eventNotHandledErr`, the result returned is `noErr`.

Available in Mac OS X v10.2 and later.

Event Queue Constants

Define constants for specifying how events should be handled on the queue.

```
enum {
    kEventLeaveInQueue = false,
    kEventRemoveFromQueue = true
};
```

Constants

`kEventLeaveInQueue`

Leave the event on the queue after examining.

Available in Mac OS X v10.0 and later.

`kEventRemoveFromQueue`

Remove the event from the queue after examining.

Available in Mac OS X v10.0 and later.

Discussion

When calling a function such as `ReceiveNextEvent` (page 57), you can specify whether to leave the event on the queue (peeking at it to determine its class, type, and so on), or to pull it before dispatching it to an event handler.

Direct Object Parameter

Define the direct object parameter.

```
enum {
    kEventParamDirectObject = '----'
};
```

Constants

`kEventParamDirectObject`
Type varies depending on event.

Available in Mac OS X v10.0 and later.

Discussion

The direct object parameter is usable for a wide variety of events. It defines the "object the event acted upon or within." For example, for window events, the direct object parameter returns a reference (that is a `WindowRef`) to the window in which the event occurred.

Event Target Parameter

Define constants for a special event target parameter and its type, that you can set for any created event.

```
enum {
    kEventParamPostTarget = 'ptrg',
    typeEventTargetRef = 'etrg'
};
```

Constants

`kEventParamPostTarget`
Specifies the target the event should be sent to. Instead of sending an event directly to a given target, you can set this parameter and post the event onto the event queue.

Available in Mac OS X v10.2 and later.

`typeEventTargetRef`
The parameter type for `kEventParamPostTarget`.

Available in Mac OS X v10.2 and later.

Object Reference Parameters and Types

Define constants for parameters that specify various objects and their types.

```
enum {
    kEventParamWindowRef = 'wind',
    kEventParamGrafPort = 'graf',
    kEventParamDragRef = 'drag',
    kEventParamMenuRef = 'menu',
    kEventParamEventRef = 'evnt',
    kEventParamControlRef = 'ctrl',
    kEventParamRgnHandle = 'rgnh',
    kEventParamEnabled = 'enab',
    kEventParamDimensions = 'dims',
    kEventParamBounds = 'boun',
    kEventParamAvailableBounds = 'avlb',
    kEventParamAEEventID = keyAEEventID,
    kEventParamAEEventClass = keyAEEventClass,
    kEventParamCGContextRef = 'cntx',
```

```
    kEventParamDeviceDepth = 'devd',
    kEventParamDeviceColor = 'devc',
    kEventParamMutableArray = 'marr',
    kEventParamResult = 'ansr',
    kEventParamMinimumSize = 'mnsz',
    kEventParamMaximumSize = 'mxsz',
    kEventParamAttributes = 'attr',
    kEventParamReason = 'why?',
    kEventParamTransactionID = 'trns',
    kEventGDevice = 'gdev',
    kEventParamIndex = 'indx',
    kEventParamUserData = 'usrd',
    kEventParamShape = 'shap',
    typeWindowRef = 'wind',
    typeGrafPtr = 'graf',
    typeGWorldPtr = 'gwld',
    typeDragRef = 'drag',
    typeMenuRef = 'menu',
    typeControlRef = 'ctrl',
    typeCollection = 'cltn',
    typeQDRgnHandle = 'rgnh',
    typeOSStatus = 'osst',
    typeCFIndex = 'cfix',
    typeCFStringRef = 'cfst',
    typeCFMutableStringRef = 'cfms',
    typeCFTypeRef = 'cfty',
    typeCGContextRef = 'cntx',
    typeHIPoint = 'hipt',
    typeHISize = 'hisz',
    typeHIRect = 'hirc',
    typeHIShapeRef = 'shap',
    typeVoidPtr = 'void',
    typeGDHandle = 'gdev'
};
```

Constants

`kEventParamWindowRef`
A window reference. (typeWindowRef)

Available in Mac OS X v10.0 and later.

`kEventParamGrafPort`
typeGrafPtr

Available in Mac OS X v10.0 and later.

`kEventParamDragRef`
typeDragRef

Available in Mac OS X v10.0 and later.

`kEventParamMenuRef`
typeMenuRef

Available in Mac OS X v10.0 and later.

`kEventParamEventRef`
typeEventRef

Available in Mac OS X v10.0 and later.

`kEventParamControlRef`
typeControlRef

Available in Mac OS X v10.0 and later.

`kEventParamRgnHandle`
typeQDRgnHandle

Available in Mac OS X v10.0 and later.

`kEventParamEnabled`
typeBoolean

Available in Mac OS X v10.0 and later.

`kEventParamDimensions`
typeQDPoint

Available in Mac OS X v10.0 and later.

`kEventParamBounds`
typeQDRectangle

Available in Mac OS X v10.3 and later.

`kEventParamAvailableBounds`
typeQDRectangle

Available in Mac OS X v10.0 and later.

`kEventParamAEEventID`
typeType

Available in Mac OS X v10.0 and later.

`kEventParamAEEventClass`
typeType

Available in Mac OS X v10.0 and later.

`kEventParamCGContextRef`
typeCGContextRef

Available in Mac OS X v10.0 and later.

`kEventParamDeviceDepth`
typeShortInteger

Available in Mac OS X v10.1 and later.

`kEventParamDeviceColor`
typeBoolean

Available in Mac OS X v10.1 and later.

`kEventParamMutableArray`
typeCFMutableArrayRef

Available in Mac OS X v10.2 and later.

`kEventParamResult`
Any type, depending on the event

Available in Mac OS X v10.2 and later.

`kEventParamMinimumSize`
typeHISize

Available in Mac OS X v10.2 and later.

`kEventParamMaximumSize`
typeHISize

Available in Mac OS X v10.2 and later.

`kEventParamAttributes`
typeUInt32

Available in Mac OS X v10.0 and later.

`kEventParamReason`
typeUInt32

Available in Mac OS X v10.3 and later.

`kEventParamTransactionID`
typeUInt32

Available in Mac OS X v10.3 and later.

`kEventParamGDevice`
typeGDHandle

Available in Mac OS X v10.3 and later.

`kEventParamIndex`
typeCFIndex

Available in Mac OS X v10.3 and later.

`kEventParamUserData`
typeVoidPtr

Available in Mac OS X v10.3 and later.

`kEventParamShape`
typeHIShapeRef

Available in Mac OS X v10.4 and later.

`typeWindowRef`
WindowRef

Available in Mac OS X v10.0 and later.

`typeGrafPtr`
CGrafPtr

Available in Mac OS X v10.0 and later.

`typeGWorldPtr`
GWorldPtr

Available in Mac OS X v10.0 and later.

`typeDragRef`
DragRef

Available in Mac OS X v10.0 and later.

`typeMenuRef`
MenuRef

Available in Mac OS X v10.0 and later.

`typeControlRef`
ControlRef

Available in Mac OS X v10.0 and later.

`typeCollection`
Collection

Available in Mac OS X v10.0 and later.

`typeQDRgnHandle`
RgnHandle

Available in Mac OS X v10.0 and later.

`typeOSStatus`
OSStatus

Available in Mac OS X v10.0 and later.

`typeCFIndex`
CFIndex

Available in Mac OS X v10.2 and later.

`typeCFStringRef`
CFStringRef

Available in Mac OS X v10.1 and later.

`typeCFMutableStringRef`
CFMutableStringRef

Available in Mac OS X v10.2 and later.

`typeCFTypeRef`
CFTypeRef

Available in Mac OS X v10.2 and later.

`typeCGContextRef`
CGContextRef

Available in Mac OS X v10.0 and later.

`typeHIPoint`
HIPoint

Available in Mac OS X v10.1 and later.

`typeHISize`
HISize

Available in Mac OS X v10.2 and later.

`typeHIRect`
HIRect

Available in Mac OS X v10.2 and later.

`typeHIShapeRef`
HIShapeRef

Available in Mac OS X v10.4 and later.

`typeVoidPtr`
Void

Available in Mac OS X v10.2 and later.

`typeGDHandle`
GDHandle

Available in Mac OS X v10.3 and later.

Discussion
You specify these parameters to obtain references to various objects such as windows, controls, graphics ports, and so on. See the various event kinds to determine the parameters available for each event. For example, Table 8 (page 149) in "Mouse Events" (page 148) lists the various parameters used in mouse events.

Apple Event Constants

AppleEvent Constant

Define a constant related to events from `kEventClassAppleEvent`.

```
enum {
    kEventAppleEvent = 1
};
```

Constants
`kEventAppleEvent`
An AppleEvent event was received.

Available in Mac OS X v10.0 and later.

Discussion
The standard application handler automatically calls the Apple Event Manager function `AEProcessAppleEvent` to handle Apple events.

Table 1 shows the parameter associated with AppleEvent events.

Table 1 Parameter names and types for AppleEvent kinds

Event kind	Parameter name	Parameter type
`kEventAppleEvent`	`kEventParamAEEventID`	`typeType`

Deprecated AppleEvent Event Constants

Define constants for older names for AppleEvent event constants.

```
enum {
    kEventClassEPPC = kEventClassAppleEvent,
    kEventHighLevelEvent = kEventAppleEvent
};
```

Constants

`kEventClassEPPC`
Equivalent to `kEventClassAppleEvent`.

Available in Mac OS X v10.0 and later.

`kEventHighLevelEvent`
Equivalent to `kEventAppleEvent`.

Available in Mac OS X v10.0 and later.

Appearance Manager Event Constants

Appearance Manager Events

Define a constant related to events from `kEventClassAppearance`.

```
enum {
    kEventAppearanceScrollBarVariantChanged = 1
};
```

Constants

`kEventAppearanceScrollBarVariantChanged`
The scroll bar variant has changed.

Available in Mac OS X v10.1 and later.

Appearance Manager Event Parameter

Define a constant for the parameter to Appearance Manager events.

```
enum {
    kEventParamNewScrollBarVariant = 'nsbv'
};
```

Constants

`kEventParamNewScrollBarVariant`
typeShortInteger

Available in Mac OS X v10.1 and later.

Application Event Constants

Application Event Constants

Define constants related to events from `kEventClassApplication`.

```
enum {
    kEventAppActivated = 1,
    kEventAppDeactivated = 2,
    kEventAppQuit = 3,
    kEventAppLaunchNotification = 4,
    kEventAppLaunched = 5,
    kEventAppTerminated = 6,
    kEventAppFrontSwitched = 7,
    kEventAppFocusMenuBar = 8,
    kEventAppFocusNextDocumentWindow = 9,
    kEventAppFocusNextFloatingWindow = 10,
    kEventAppFocusToolbar = 11,
    kEventAppFocusDrawer = 12,
    kEventAppGetDockTileMenu = 20,
    kEventAppIsEventInInstantMouser = 104,
    kEventAppHidden = 107,
    kEventAppShown = 108,
    kEventAppSystemUIModeChanged = 109,
    kEventAppAvailableWindowBoundsChanged = 110,
    kEventAppActiveWindowChanged = 111
};
```

Constants

`kEventAppActivated`
The application was activated (resumed, in old parlance).

Available in Mac OS X v10.0 and later.

`kEventAppDeactivated`
The application was deactivated (suspended, in old parlance).

Available in Mac OS X v10.0 and later.

`kEventAppQuit`
The application is quitting.

Available in Mac OS X v10.0 and later.

`kEventAppLaunchNotification`
Response to asynchronous application launch.

Available in Mac OS X v10.0 and later.

`kEventAppLaunched`
Some other application was launched. (CarbonLib 1.3 or later)

Available in Mac OS X v10.0 and later.

`kEventAppTerminated`
Some other application was terminated. (CarbonLib 1.3 or later)

Available in Mac OS X v10.0 and later.

`kEventAppFrontSwitched`
The frontmost application has changed. (CarbonLib 1.3 or later)

Available in Mac OS X v10.0 and later.

`kEventAppFocusMenuBar`
Request to switch the keyboard focus to the menu bar. The Carbon Event Manager handles this event by default.

Available in Mac OS X v10.2 and later.

`kEventAppFocusNextDocumentWindow`
Request to shift keyboard focus to the next or previous document window (depending on the state of the Shift key). If there are no more document windows in the current process, focus should shift to the document window in the next (or previous) process.

If something other than a document window currently has keyboard focus, you should shift focus to the frontmost document window without changing the ordering of the windows.

If the document window does not have a focused area, you should set the focus to the main control within the window.

The Carbon Event Manager handles this event by default; if you handle this event, you should only check if the user focus is somewhere other than a document window, and if so, set the focus on the active document window. If the focus is already on a document window, your handler should always return `eventNotHandledErr` so that the default handler can rotate to the next window across all processes.

Available in Mac OS X v10.2 and later.

`kEventAppFocusNextFloatingWindow`
Request to shift keyboard focus to the next or previous floating window (depending on the state of the Shift key).

If something other than a floating window currently has keyboard focus, you should shift focus to the frontmost floating window without changing the ordering of the windows.

If the floating window does not have a focused area, you should set the focus to the main control within the window.

The default behavior for this event is to send a `kEventCommandProcess` event containing `kHICommandRotateFloatingWindowsForward` or `kHICommandRotateFloatingWindowsBackward`.

Available in Mac OS X v10.2 and later.

`kEventAppFocusToolbar`
Request to shift keyboard focus to the toolbar.

The default behavior for this event is to move the keyboard focus to the first item in the toolbar (assuming you are using the standard toolbar).

Available in Mac OS X v10.2 and later.

`kEventAppFocusDrawer`
Request to shift keyboard focus to the drawer in the focused window.

The default behavior for this event is to move the focus to the first control in the drawer in the focused window if a drawer is present. If multiple drawers are present, focus is moved in clockwise order from one drawer to the next, starting with the top drawer, if any. If the modifiers parameter contains the shift key, focus is moved in reverse (counterclockwise) order.

(Available in Mac OS X v10.4 and later.)

Available in Mac OS X v10.4 and later.

`kEventAppGetDockTileMenu`
Request to display a pop-up menu by the application's dock tile. You should return the menu reference of the menu to display in the `kEventParamMenuRef` parameter. The sender of this event releases this menu after the Dock displays it, so if you supply a permanently allocated menu reference, you should call the Menu Manager function `RetainMenu` on it before returning from your handler.

The default behavior for this event is to return the menu (if any) supplied by the `SetApplicationDockTileMenu` function (described in the Dock Manager Reference). Note that for most functions, it's easier to set a menu using `SetApplicationDockTileMenu` rather than installing a handler for this event.

(Available in Mac OS X v10.1 and later.)

Available in Mac OS X v10.1 and later.

`kEventAppIsEventInInstantMouser`
The given event's global mouse location is over an "instant mousing" area. An instant mousing area is an area where a mouse down should not generate ink but should be interpreted as a click.

(Available in Mac OS X v10.2 and later.)

Available in Mac OS X v10.3 and later.

`kEventAppHidden`
The application was hidden. (Available in Mac OS X v10.2 and later.)

Available in Mac OS X v10.2 and later.

`kEventAppShown`
The application was shown. (Available in Mac OS X v10.2 and later.)

Available in Mac OS X v10.2 and later.

`kEventAppSystemUIModeChanged`
The system user interface mode of the frontmost application has changed. (Available in Mac OS X v10.2 and later.)

Available in Mac OS X v10.2 and later.

`kEventAppAvailableWindowBoundsChanged`

The available window positioning bounds have changed. This event is currently sent when the Dock has changed position or size and when the display configuration has changed. A separate copy of this event is sent to each affected GDevice. (Available in Mac OS X v10.3 and later.)

Available in Mac OS X v10.3 and later.

`kEventAppActiveWindowChanged`

The active window in the current process has changed.The Window Manager uses `ActiveNonFloatingWindow` to track the active window. When `SelectWindow` is called on a window, that window is made the new active window. At that time, the Window Manager also posts a `kEventAppActiveWindowChanged` event to the main event queue.

If more than one window is activated sequentially before the event loop is run, a single `kEventAppActiveWindowChanged` event is left in the event queue. Its `PreviousActiveWindow` parameter will be the window that was originally active, and its `CurrentActiveWindow` parameter will be the window that was finally active.

(Available in Mac OS X v10.3 and later.)

Available in Mac OS X v10.3 and later.

Discussion

You can pass any of these constants when registering an event handler. You can also pass these constants to the `CreateEvent` (page 24) function to specify the type of application event you want to create.

Table 2 shows the event parameters associated with application events.

Table 2 Parameter names and types for application event kinds

Event kind	Parameter name	Parameter type
`kEventAppActivated`	`kEventParamWindowRef`	`typeWindowRef`
`kEventAppDeactivated`	*None*	
`kEventAppQuit`	*None*	
`kEventAppLaunchNotification`	`kEventParamProcessID`	`typeProcessSerialNumber`
	`kEventParamLaunchRefCon`	`typeUInt32`
	`kEventParamLaunchErr`	`typeOSStatus`
`kEventAppLaunched`	`kEventParamProcessID`	`typeProcessSerialNumber`
`kEventAppTerminated`	`kEventParamProcessID`	`typeProcessSerialNumber`
`kEventAppFrontSwitched`	`kEventParamProcessID`	`typeProcessSerialNumber`
`kEventAppFocusMenuBar`	`kEventParamKeyModifiers`	`typeUInt32`
`kEventAppFocusNextDocumentWindow`	`kEventParamKeyModifiers`	`typeUInt32`
`kEventAppFocusNextFloatingWindow`	`kEventParamKeyModifiers`	`typeUInt32`

kEventAppFocusToolbar	kEventParamKeyModifiers	typeUInt32
kEventAppGetDockTileMenu	kEventParamMenuRef	typeMenuRef
kEventAppHidden	*None*	
kEventAppShown	*None*	
kEventAppSystemUIModeChanged	kEventParamSystemUIMode	typeUInt32

Application Event Parameters

Define constants for parameters to application events.

```
enum {
    kEventParamProcessID = 'psn ',
    kEventParamLaunchRefCon = 'lref',
    kEventParamLaunchErr = 'err ',
    kEventParamSystemUIMode = 'uimd'
};
```

Constants

`kEventParamProcessID`
typeProcessSerialNumber

Available in Mac OS X v10.0 and later.

`kEventParamLaunchRefCon`
typeWildcard

Available in Mac OS X v10.0 and later.

`kEventParamLaunchErr`
typeOSStatus

Available in Mac OS X v10.0 and later.

`kEventParamSystemUIMode`
typeUInt32

Available in Mac OS X v10.2 and later.

Command Events

Command Event Constants

Define constants related to events from `kEventClassCommand`.

```
enum {
    kEventProcessCommand = 1,
    kEventCommandProcess = 1,
    kEventCommandUpdateStatus = 2
};
```

Constants

`kEventProcessCommand`

A command has been invoked and the application should handle it. This event is sent when the user chooses a menu item or when a control with a command is pressed. Some senders of this event will also include the modifier keys that were pressed by the user when the command was invoked, but this parameter is optional.

Available in Mac OS X v10.0 and later.

`kEventCommandProcess`

Same as `kEventProcessCommand`.

Available in Mac OS X v10.0 and later.

`kEventCommandUpdateStatus`

Sent when updates related to the command event may be required. When you receive this event, you should update the necessary user interface elements in your application to reflect the current status of the command. For example, if the command has the `kHICommandFromMenu` bit set, you should update the menu item state, text, and so on, to reflect the current state of your application.

Note that the standard handler for `kEventMenuEnableItems` automatically sends this event to your menu commands. As this can cause a performance hit if you have many menu items, you can choose to bypass these updates by installing a no-op handler for `kEventMenuEnableItems` that simply returns `noErr`.

Available in Mac OS X v10.0 and later.

Discussion

You pass this constant to the `CreateEvent` (page 24) function to indicate the type of command event you want to create. Future releases of the Carbon Event Manager will provide additional command event types.

Table 3 shows the parameters associated with command events.

Table 3 Parameter names and types for command event kinds

Event kind	Parameter name	Parameter type
`kEventCommandProcess`	`kEventParamDirectObject`	`typeHICommand`
	`kEventParamKeyModifiers` (Optional)	`typeUInt32`
	`kEventParamMenuContext` (Optional)	`typeUInt32`
`kEventCommand-UpdateStatus`	`kEventParamDirectObject`	`typeHICommand`
	`kEventParamMenuContext` (Optional)	`typeUInt32`

Standard Command ID Constants

Define command IDs for common menu commands and controls.

```
enum {
```

```
    kHICommandOK = 'ok  ',
    kHICommandCancel = 'not!',
    kHICommandQuit = 'quit',
    kHICommandUndo = 'undo',
    kHICommandRedo = 'redo',
    kHICommandCut = 'cut ',
    kHICommandCopy = 'copy',
    kHICommandPaste = 'past',
    kHICommandClear = 'clea',
    kHICommandSelectAll = 'sall',
    kHICommandHide = 'hide',
    kHICommandHideOthers = 'hido',
    kHICommandShowAll = 'shal',
    kHICommandPreferences = 'pref',
    kHICommandZoomWindow = 'zoom',
    kHICommandMinimizeWindow = 'mini',
    kHICommandMinimizeAll = 'mina',
    kHICommandMaximizeWindow = 'maxi',
    kHICommandMaximizeAll = 'maxa',
    kHICommandArrangeInFront = 'frnt',
    kHICommandBringAllToFront = 'bfrt',
    kHICommandWindowListSeparator = 'wldv',
    kHICommandWindowListTerminator = 'wlst',
    kHICommandSelectWindow = 'swin',
    kHICommandRotateWindowsForward = 'rotw',
    kHICommandRotateWindowsBackward = 'rotb',
    kHICommandRotateFloatingWindowsForward = 'rtfw',
    kHICommandRotateFloatingWindowsBackward = 'rtfb',
    kHICommandAbout = 'abou',
    kHICommandNew = 'new ',
    kHICommandOpen = 'open',
    kHICommandClose = 'clos',
    kHICommandSave = 'save',
    kHICommandSaveAs = 'svas',
    kHICommandRevert = 'rvrt',
    kHICommandPrint = 'prnt',
    kHICommandPageSetup = 'page',
    kHICommandAppHelp = 'ahlp',
    kHICommandShowCharacterPalette = 'chrp',
    kHICommandShowSpellingPanel = 'shsp',
    kHICommandCheckSpelling = 'cksp',
    kHICommandChangeSpelling = 'chsp',
    kHICommandCheckSpellingAsYouType = 'chsp',
    kHICommandIgnoreSpelling = 'igsp',
    kHICommandLearnWord = 'lrwd'
};
```

Constants

`kHICommandOK`

The OK button in a dialog or alert.

Available in Mac OS X v10.0 and later.

`kHICommandCancel`

The Cancel button in a dialog or alert.

Available in Mac OS X v10.0 and later.

`kHICommandQuit`
The application should quit.

Available in Mac OS X v10.0 and later.

`kHICommandUndo`
The last editing operation should be undone.

Available in Mac OS X v10.0 and later.

`kHICommandRedo`
The last editing operation should be redone.

Available in Mac OS X v10.0 and later.

`kHICommandCut`
The selected items should be cut.

Available in Mac OS X v10.0 and later.

`kHICommandCopy`
The selected items should be copied.

Available in Mac OS X v10.0 and later.

`kHICommandPaste`
The contents of the clipboard should be pasted.

Available in Mac OS X v10.0 and later.

`kHICommandClear`
The selected items should be deleted.

Available in Mac OS X v10.0 and later.

`kHICommandSelectAll`
All items in the active window should be selected.

Available in Mac OS X v10.0 and later.

`kHICommandHide`
The application should be hidden. The Menu Manager responds to this command automatically; your application does not need to handle it.

Available in Mac OS X v10.0 and later.

`kHICommandHideOthers`
Other applications should be hidden. The Menu Manager responds to this command automatically; your application does not need to handle it.

Available in Mac OS X v10.1 and later.

`kHICommandShowAll`
All applications should become visible. The Menu Manager responds to this command automatically; your application does not need to handle it.

Available in Mac OS X v10.1 and later.

`kHICommandPreferences`
The Preferences menu item has been selected.

Available in Mac OS X v10.0 and later.

`kHICommandZoom`
The active window should be zoomed in or out. The default application handler responds to this event automatically. Your application does not need to handle this event, but you may want to install a Carbon event handler for `kEventWindowGetIdealSize` to return the ideal size for your document windows.

`kHICommandMinimizeWindow`
The active window should be minimized. The default application handler will respond to this event automatically; your application does not need to handle it.

Available in Mac OS X v10.0 and later.

`kHICommandMinimizeAll`
All collapsable windows should be minimized. The default application handler responds to this event automatically; your application does not need to handle it.

Available in Mac OS X v10.1 and later.

`kHICommandMaximizeWindow`
The active window should be maximized. Sent only on Mac OS 9. The default application handler will respond to this event automatically; your application does not need to handle it.

Available in Mac OS X v10.1 and later.

`kHICommandMaximizeAll`
All collapsible windows should be maximized. This event is not sent or handled on Mac OS X.

Available in Mac OS X v10.1 and later.

`kHICommandArrangeInFront`
All document-class windows should be arranged in a stack. The default application handler responds to this event automatically; your application does not need to handle it.

Available in Mac OS X v10.0 and later.

`kHICommandBringAllToFront`
All windows of this application should be brought in front of windows from other applications. Sent only on Mac OS X. The default application handler responds to this event automatically; your application does not need to handle it.

Available in Mac OS X v10.1 and later.

`kHICommandWindowListSeparator`
A placeholder to mark the separator item dividing the Zoom/Minimize/Maximize/Arrange menu items in the standard Window menu from the menu items listing the visible windows. If you need to add your own menu items to the standard Window menu before the window list section, you can use `GetIndMenuItemWithCommandID` (described in the Menu Manager Reference in the User Experience section of the Carbon documentation) to look for the menu item with this command ID and insert your menu items before the item with this ID.

Available in Mac OS X v10.1 and later.

`kHICommandWindowListTerminator`

Used as a placeholder to mark the end of the window list section of the standard Window menu. If you need to add your own menu items to the standard Window menu after the window list section, you can use `GetIndMenuItemWithCommandID` (described in the Menu Manager Reference in the User Experience section of the Carbon documentation) to look for the menu item with this command ID and insert your items after the item with this ID.

Available in Mac OS X v10.1 and later.

`kHICommandSelectWindow`

A window in the standard Window menu has been selected and should be activated. In Mac OS X v10.3, this command is also sent by the toolbox whenever it needs to activate a window in your application. For example, it is used when a window is selected from the application's Dock menu, and when a window that uses the standard window event handler is clicked. The default application handler responds to this event automatically; your application does not need to handle it.

Available in Mac OS X v10.1 and later.

`kHICommandRotateWindowsForward`

The Rotate Windows hot key (cmd-~ by default) has been pressed. Non-floating windows should be rotated so that the window after the active window is activated. The default application handler responds to this event automatically; your application does not need to handle it.

Available in Mac OS X v10.2 and later.

`kHICommandRotateWindowsBackward`

The Rotate Windows hot key (cmd-~ by default) has been pressed. Non-floating windows should be rotated so that the window before the active window is activated. The default application handler responds to this event automatically; your application does not need to handle it.

Available in Mac OS X v10.2 and later.

`kHICommandRotateFloatingWindowsForward`

The floating window focus hot key (ctl-F6 by default) has been pressed, and floating windows should be rotated so that the window after the focused window is activated. The default application handler responds to this event automatically; your application does not need to handle it.

Available in Mac OS X v10.2 and later.

`kHICommandRotateFloatingWindowsBackward`

The floating window focus hot key (ctl-F6 by default) has been pressed, and floating windows should be rotated so that the window before the focused window is activated. The default application handler responds to this event automatically; your application does not need to handle it.

Available in Mac OS X v10.2 and later.

`kHICommandAbout`

The About menu item has been selected. In Mac OS X v10.3 and later, `RunApplicationEventLoop` installs a handler for this command ID on the application target that handles this event automatically by calling `HIAboutBox`. Your application can install its own handler if you want to display a customized about box.

Available in Mac OS X v10.0 and later.

`kHICommandNew`
A new document or item should be created.

Available in Mac OS X v10.1 and later.

`kHICommandOpen`
The user wants to open an existing document.

Available in Mac OS X v10.1 and later.

`kHICommandClose`
The active window should be closed. This command is typically be generated by a Close menu item. In Mac OS X v10.3 and later, the default application handler responds to this command by sending a `kEventWindowClose` event; on earlier systems, only the standard window event handler responded to this event.

Available in Mac OS X v10.1 and later.

`kHICommandSave`
The active document should be saved.

Available in Mac OS X v10.1 and later.

`kHICommandSaveAs`
The user wants to save the active document under a new name.

Available in Mac OS X v10.1 and later.

`kHICommandRevert`
The contents of the active document should be reverted to the last saved version.

Available in Mac OS X v10.1 and later.

`kHICommandPrint`
The active window should be printed.

Available in Mac OS X v10.1 and later.

`kHICommandPageSetup`
The user wants to configure the current page margins, formatting, and print options.

Available in Mac OS X v10.1 and later.

`kHICommandAppHelp`
The application's help book should be displayed. The Help Manager installs a handler for this command ID on the Help menu returned by `HMGetHelpMenu` and responds to this event automatically. Your application does not need to handle it. In Mac OS X v10.4, the Help Manager installs a handler for this event on the application event target rather than on the Help menu.

Available in Mac OS X v10.1 and later.

`kHICommandShowCharacterPalette`
The character palette needs to be shown. Events with this command ID are only generated in Mac OS X v10.3 and later. The toolbox will respond to this event automatically; your application does not need to handle it.

Available in Mac OS X v10.3 and later.

`kHICommandShowSpellingPanel`
: Display the spelling panel if it is not already visible. Events with this command ID are only generated in Mac OS X v10.4 and later. If spell checking has been enabled in the Multilingual Text Engine (MLTE) or an HITextView, this command is handled automatically.

  Available in Mac OS X v10.4 and later.

`kHICommandCheckSpelling`
: Spell check the document now. Events with this command ID are only generated in Mac OS X v10.4 and later. If spell checking has been enabled in MLTE or an HITextView, this command is handled automatically.

  Available in Mac OS X v10.4 and later.

`kHICommandChangeSpelling`
: Change the spelling. Events with this command ID are only generated in Mac OS X v10.4 and later. If spell checking has been enabled in MLTE or an HITextView, this command is handled automatically.

  Available in Mac OS X v10.4 and later.

`kHICommandCheckSpellingAsYouType`
: Begin interactive spell checking. Events with this command ID are only generated in Mac OS X v10.4 and later. If spell checking has been enabled in MLTE or an HITextView, this command is handled automatically.

  Available in Mac OS X v10.4 and later.

`kHICommandIgnoreSpelling`
: Ignore this word while spell checking this text view. Events with this command ID are only generated in Mac OS X v10.4 and later. If spell checking has been enabled in MLTE or an HITextView, this command is handled automatically.

  Available in Mac OS X v10.4 and later.

`kHICommandLearnWord`
: Learn this spelling for all documents. Events with this command ID are generated only in Mac OS X v10.4 and later. If spell checking has been enabled in MLTE or an HITextView, this command is handled automatically.

  Available in Mac OS X v10.4 and later.

Discussion
You should use these values for standard menu and control commands rather than defining your own.

Command Event Source Constants

Define constants for the user interface element that produced an HICommand event.

```
enum {
    kHICommandFromMenu = (1L << 0),
    kHICommandFromControl = (1L << 1),
    kHICommandFromWindow = (1L << 2)
};
```

Constants

`kHICommandFromMenu`

This bit is set for commands generated from menu items in all versions of CarbonLib and Mac OS X.

Available in Mac OS X v10.0 and later.

`kHICommandFromControl`

The command event originated from a control. This bit was introduced in Mac OS X v10.2 and CarbonLib 1.6; it is never set in earlier versions of Mac OS X or CarbonLib.

Available in Mac OS X v10.2 and later.

`kHICommandFromWindow`

The command event originated from a window. This bit was introduced in Mac OS X v10.2 and CarbonLib 1.6; it is never set in earlier versions of Mac OS X or CarbonLib.

Available in Mac OS X v10.2 and later.

Control Events

Control Event Constants

Define constants related to events from `kEventClassControl`.

```
enum {
    kEventControlInitialize = 1000,
    kEventControlDispose = 1001,
    kEventControlGetOptimalBounds = 1003,
    kEventControlDefInitialize = kEventControlInitialize,
    kEventControlDefDispose = kEventControlDispose,
    kEventControlHit = 1,
    kEventControlSimulateHit = 2,
    kEventControlHitTest = 3,
    kEventControlDraw = 4,
    kEventControlApplyBackground = 5,
    kEventControlApplyTextColor = 6,
    kEventControlSetFocusPart = 7,
    kEventControlGetFocusPart = 8,
    kEventControlActivate = 9,
    kEventControlDeactivate = 10,
    kEventControlSetCursor = 11,
    kEventControlContextualMenuClick = 12,
    kEventControlClick = 13,
    kEventControlGetNextFocusCandidate = 14,
    kEventControlGetAutoToggleValue = 15,
    kEventControlInterceptSubviewClick = 16,
    kEventControlGetClickActivation = 17,
    kEventControlDragEnter = 18,
    kEventControlDragWithin = 19,
    kEventControlDragLeave = 20,
    kEventControlDragReceive = 21,
    kEventControlTrack = 51,
    kEventControlGetScrollToHereStartPoint = 52,
    kEventControlGetIndicatorDragConstraint = 53,
```

```
    kEventControlIndicatorMoved = 54,
    kEventControlGhostingFinished = 55,
    kEventControlGetActionProcPart = 56,
    kEventControlGetPartRegion = 101,
    kEventControlGetPartBounds = 102,
    kEventControlSetData = 103,
    kEventControlGetData = 104,
    kEventControlGetSizeConstraints = 105,
    kEventControlValueFieldChanged = 151,
    kEventControlAddedSubControl = 152,
    kEventControlRemovingSubControl = 153,
    kEventControlBoundsChanged = 154,
    kEventControlTitleChanged = 158,
    kEventControlOwningWindowChanged = 159,
    kEventControlHiliteChanged = 160,
    kEventControlEnabledStateChanged = 161,
    kEventControlArbitraryMessage = 201
};
```

Constants

`kEventControlInitialize`
Sent when a control is created. Allows the control to initialize private data.

Available in Mac OS X v10.0 and later.

`kEventControlDispose`
Sent when a control is disposed. Allows the control to dispose of private data.

Available in Mac OS X v10.0 and later.

`kEventControlGetOptimalBounds`
Allows the control to report its best size and its text baseline based on its current settings. You should set the `kEventParamControlOptimalBounds` parameter to an appropriate rectangle. You should also set the `kEventParamControlOptimalBaselineOffset` parameter to be the offset from the top of your optimal bounds of a text baseline, if any. (Mac OS X only)

Available in Mac OS X v10.0 and later.

`kEventControlDefInitialize`
Same as `kEventControlInitialize`. You can use this event when creating custom control definitions.

Available in Mac OS X v10.0 and later.

`kEventControlDefDispose`
Same as `kEventControlDispose`. You can use this event when creating custom control definitions.

Available in Mac OS X v10.0 and later.

`kEventControlHit`
Sent by the Control Manager functions `TrackControl` and `HandleControlClick` after handling a click in a control. If you do not handle this event, and the control has a command ID associated with it, then the Control Manager sends a `kEventCommandProcess` event to the control.

Available in Mac OS X v10.0 and later.

`kEventControlSimulateHit`
Sent when your control should simulate a click in response to some other action, such as a return key for a default button. The default behavior is to use the Control Manager function `HiliteControl` to highlight and unhighlight the part specified in the `kEventParamControlPart` parameter (simulating the hit) and then call the control's action callback function. (Mac OS X only)

Available in Mac OS X v10.0 and later.

`kEventControlHitTest`
Sent when someone wants to find out what part of your control is at a given point in local coordinates. You should set the `kEventParamControlPart` parameter to the appropriate part. (Mac OS X only)

Available in Mac OS X v10.0 and later.

`kEventControlDraw`
Sent when your control should draw itself. The event can optionally contain parameters indicating which port to draw into and which part to constrain drawing to. (Mac OS X only)

Available in Mac OS X v10.0 and later.

`kEventControlApplyBackground`
Sent when your control should apply its background color/pattern to the port specified so the subcontrol can properly erase. The port is optional; if it does not exist you should apply the background to the current port. Note that if you don't handle this event, the event is propagated to the control's parent. (Mac OS X only)

Available in Mac OS X v10.0 and later.

`kEventControlApplyTextColor`
Sent when your control should apply a color/pattern to the specified port and context so a subcontrol can draw text which looks appropriate for your control's background. The port is optional; if it does not exist, you should apply the text color to the current port. The context is also optional. (Mac OS X only)

Available in Mac OS X v10.0 and later.

`kEventControlSetFocusPart`
Sent when your control is gaining, losing, or changing the focus. Set the focus to the part indicated by the `kEventParamControlPart` parameter. (Mac OS X only)

Available in Mac OS X v10.0 and later.

`kEventControlGetFocusPart`
Sent when your the Control Manager wants to know what part of your control is currently focused. Set the `kEventParamControlPart` parameter to your currently focused part. If you don't handle this event, the Control Manager sets the part parameter to the last part that was focused (or no part if the control lost focus). (Mac OS X only)

Available in Mac OS X v10.0 and later.

`kEventControlActivate`
Sent when your control becomes active as a result of a call to `ActivateControl`. (Mac OS X only)

Available in Mac OS X v10.0 and later.

`kEventControlDeactivate`
Sent when your control becomes inactive as a result of a call to `DeactivateControl`. (Mac OS X only)

Available in Mac OS X v10.0 and later.

`kEventControlSetCursor`
Sent when your control is asked to change the cursor as a result of a call to the Control Manager function `HandleControlSetCursor`. (Mac OS X only)

Available in Mac OS X v10.0 and later.

`kEventControlContextualMenuClick`
Sent when your control is asked to display a contextual menu as a result of a call to the Control Manager function `HandleControlContextualMenuClick`. (Mac OS X only)

Available in Mac OS X v10.0 and later.

`kEventControlClick`
A mouse down occurred in a control. The standard window handler sets the keyboard focus to the control if it takes focus on clicks, and calls the Control Manager function `HandleControlClick`.

Available in Mac OS X and CarbonLib 1.3.1 and later.

`kEventControlGetNextFocusCandidate`
Sent to allow a control to customize the focus order of its subcontrols. The current subcontrol with focus is stored in the `kEventParamStartControl` parameter. The desired focus direction is indicated by the `kControlFocusNextPart` or `kControlFocusPrevPart` constants, passed to you in the `kEventParamControlPart` parameter. The handler should return the next subcontrol in the `kEventParamNextControl` parameter. If the `kEventParamStartControl` parameter is `NULL`, return the first subcontrol in the specified focus direction. If no next subcontrol exists in the desired focus direction, return `NULL` or omit the `kEventParamNextControl` parameter.

The default behavior is to return the “most appropriate” peer control, which currently means the previous control in the ordering scheme.

(Available in Mac OS X v10.2 and later.)

Available in Mac OS X v10.2 and later.

`kEventControlGetAutoToggleValue`
Sent when the system wants to auto-toggle a control. You can specify the value to use based on the current value of your control.

If the control has the `kControlAutoToggles` feature bit set, then the default behavior is as follows:

- If the control does not behave like a radio button (the `kControlHasRadioBehavior` feature bit is not set), and its value is 1, then the `kEventParamControlValue` parameter is set to 0.
- If the control's value is anything other than 1, the `kEventParamControlValue` parameter is set to 0.

Otherwise, there is no default behavior.

(Available in Mac OS X v10.2 and later.)

Available in Mac OS X v10.2 and later.

`kEventControlInterceptSubviewClick`
Sent when the `HIViewGetViewForMouseClick` function is called (typically by the Control Manager before it descends into subviews). A view can use this event to intercept mouse clicks that would normally be destined for one of its subviews. For example, the Toolbar control uses this event to intercept command-clicks so that it can handle dragging of its children. If the command key is down, the user wants to drag, so the handler returns `noErr` to indicate that this view (the Toolbar) should receive the click, not the child that was actually under the mouse.

(Available in Mac OS X v10.2 and later.)

Available in Mac OS X v10.2 and later.

`kEventControlGetClickActivation`
Sent when a mouse click occurs in a background (inactive) control. This event is essentially the control version of `kEventWindowGetClickActivation`. The only differences are that the mouse location is view-relative and no window part parameter is passed to you.

This event is sent only when the standard window handler is installed. The default behavior is to activate the view and absorb the mouse click (that is, the click is not passed on to the view).

(Available in Mac OS X v10.2 and later.)

Available in Mac OS X v10.2 and later.

`kEventControlDragEnter`
Sent when a drag item enters a view's bounds. If you want to respond to the drag, your drag entered handler must return `noErr` If you return `eventNotHandledErr` then you will not receive further drag events, nor will you be able to receive the drag item.

(Available in Mac OS X v10.2 and later.)

Available in Mac OS X v10.2 and later.

`kEventControlDragWithin`
Sent when a drag item has moved while in the view's bounds (but not within any of its subviews). If the drag subsequently enters a subview, all additional drag events are directed to that subview.

(Available in Mac OS X v10.2 and later.)

Available in Mac OS X v10.2 and later.

`kEventControlDragLeave`
Sent when a drag item leaves your view. You can use this event to unhighlight your view, and so on. (Available in Mac OS X v10.2 and later.)

Available in Mac OS X v10.2 and later.

`kEventControlDragReceive`
Sent when a drag item is dropped within your view. (Available in Mac OS X v10.2 and later.)

Available in Mac OS X v10.2 and later.

`kEventControlTrack`
Sent to allow your control to completely replace the normal tracking that is part of a call to the Control Manager functions `TrackControl` or `HandleControlClick`. Set the `kEventParamControlPart` to the part hit during tracking.

This event is sent only to controls that return a non-zero control part code from `kEventControlHitTest`. If you are implementing a custom HIView and you need to receive this event, you must also handle `kEventControlHitTest`. The hit-test handler must place a valid control part code into the `kEventParamControlPart` parameter and return `noErr`.

The default behavior is to implement indicator tracking (if the mouse is down in an indicator part, such as for a scroll bar) or one-part tracking (if the mouse is down in a button or similar part). If the tracking is successful, the Control Manager passes back the part that was hit.

(Available in Mac OS X only)

Available in Mac OS X v10.0 and later.

`kEventControlGetScrollToHereStartPoint`
Sent so your control can support "Scroll To Here" behavior during tracking. Set the `kEventParamMouseLocation` parameter to the mouse location in local coordinates which represents where a click would have needed to be to cause your indicator to be dragged to the incoming mouse location. (Mac OS X only)

Available in Mac OS X v10.0 and later.

`kEventControlGetIndicatorDragConstraint`
Sent so your control can constrain the movement of its indicator during tracking. Set the `kEventParamControlIndicatorDragConstraint` parameter to the appropriate constraint. (Mac OS X only)

Available in Mac OS X v10.0 and later.

`kEventControlIndicatorMoved`
Sent during live-tracking of the indicator so your control can update its value based on the new indicator position. During non-live tracking, this event lets you redraw the indicator ghost at the appropriate place. (Mac OS X only)

Available in Mac OS X v10.0 and later.

`kEventControlGhostingFinished`
Sent at the end of non-live indicator tracking so your control can update its value based on the final ghost location. (Mac OS X only)

Available in Mac OS X v10.0 and later.

`kEventControlGetActionProcPart`
Sent during tracking so your control can alter the part that is passed to its action callback based on modifier keys, etc. Set the `kEventParamControlPart` to the part you want to have sent. (Mac OS X only)

Available in Mac OS X v10.0 and later.

`kEventControlGetPartRegion`
Sent when a client wants to get a particular region of your control. See the `GetControlRegion` function in the Control Manager. The `kEventParamControlRegion` contains a region for you to modify.

If the requested part is `kControlStructureMetaPart`, the default behavior is to pass back a region equal to the control's bounds. Otherwise, there is no default behavior.

(Mac OS X only)

Available in Mac OS X v10.0 and later.

`kEventControlGetPartBounds`
Sent when a client wants to get a particular rectangle of your control when it may be more efficient than asking for a region. Set the `kEventParamControlPartBounds` parameter to the appropriate rectangle. (Mac OS X only)

Available in Mac OS X v10.0 and later.

`kEventControlSetData`
Sent when a client wants to change an arbitrary setting of your control. See the `SetControlData` function in the Control Manager. (Mac OS X only)

Available in Mac OS X v10.0 and later.

`kEventControlGetData`
Sent when a client wants to get an arbitrary setting of your control. See the Control Manager function `GetControlData`. (Mac OS X only)

Available in Mac OS X v10.0 and later.

`kEventControlGetSizeConstraints`
Sent when the `HIViewGetSizeContraints` function is called. You use this to let your custom view indicate its maximum and minimum size. A parent view can use this information to help it lay out subviews. (Available in Mac OS X v10.2 and later.)

Available in Mac OS X v10.2 and later.

`kEventControlValueFieldChanged`
Sent when your control's value, minimum, maximum, or view size has changed. Useful so other entities can watch for your control's value to change. If the window does not have compositing enabled, the default behavior is to redraw the control (but not its subcontrols). (Mac OS X only)

Available in Mac OS X v10.0 and later.

`kEventControlAddedSubControl`
Sent when a control is embedded within your control. (Mac OS X only)

Available in Mac OS X v10.0 and later.

`kEventControlRemovingSubControl`
Sent when one of your child controls will be removed from your control. (Mac OS X only)

Available in Mac OS X v10.0 and later.

`kEventControlBoundsChanged`
Sent when your control's bounding rectangle has changed. Note that the `kEventParamOriginalBounds` and `kEventParamPreviousBounds` parameters for this event contain the same value. (Mac OS X only)

Available in Mac OS X v10.0 and later.

`kEventControlTitleChanged`
Sent when your control's title changes. (Available in Mac OS X v10.2 and later.)

Available in Mac OS X v10.2 and later.

`kEventControlOwningWindowChanged`
Sent when one your control's owning window has changed. Useful to update any dependencies that your control has on its owning window. (Mac OS X only)

Available in Mac OS X v10.0 and later.

`kEventControlHiliteChanged`
Sent when a control's highlight state changes. (Available in Mac OS X v10.2 and later.)

Available in Mac OS X v10.2 and later.

`kEventControlEnabledStateChanged`
Sent when a control's enabled state changes (that is, when a control is enabled or disabled). (Available in Mac OS X v10.2 and later.)

Available in Mac OS X v10.2 and later.

`kEventControlArbitraryMessage`
Sent when someone tries to send an old-style CDEF message to your control. In most cases, you should implement Carbon event replacements for CDEF messages instead. If you do handle this event, but do not explicitly handle a particular CDEF message, you should propagate this event up the handler chain (either explicitly by calling `CallNextEventHandler` (page 20) or implicitly by returning `eventNotHandledErr`), as some default behavior may be implemented for compatibility purposes. (Mac OS X only)

Available in Mac OS X v10.0 and later.

Discussion
You can specify any of these events when installing an event handler. You can also pass these constants to the `CreateControlEvent` function to specify the type of control event you want to create.

Note that many control events are not sent as a request for you to take action; rather they provide a way for the application to override default behavior. Because this is the case, most control events do not have a standard handler associated with them. Their default behavior occurs whether or not you have the standard window handler installed.

Table 4 shows the parameters available for control events.

Table 4 Parameter names and types for common control event kinds

Event kind	Parameter name	Parameter type
kEventControl-Initialize	kEventParamDirectObject	typeControlRef
	kEventParamInitCollection	typeCollection
	kEventParamControlFeatures	typeUInt32
kEventControlDispose	kEventParamDirectObject	typeControlRef
kEventControlGet-Optimal Bounds	kEventParamDirectObject	typeControlRef
	kEventParamControlOptimal Bounds	typeQDRectangle
	kEventParamControlOptimal BaselineOffset (Optional)	typeShortInteger
kEventControlHit	kEventParamDirectObject	typeControlRef
	kEventParamControlPart	typeControlPartCode
	kEventParamKeyModifiers	typeUInt32
kEventControl-SimulateHit	kEventParamDirectObject	typeControlRef
	kEventParamControlPart	typeControlPartCode
	kEventParamKeyModifiers	typeUInt32
kEventControlHitTest	kEventParamDirectObject	typeControlRef
	kEventParamMouseLocation	typeQDPoint
	kEventParamControlPart	typeControlPartCode
kEventControlDraw	kEventParamDirectObject	typeControlRef
	kEventParamControlPart (Optional)	typeControlPartCode
	kEventParamGrafPort (Optional)	typeGrafPtr
	kEventParamRgnHandle (Optional)	typeQDRgnHandle
	kEventParamCGContextRef (Optional)	typeCGContextRef
kEventControlApply-Background	kEventParamDirectObject	typeControlRef
	kEventParamControlSubControl	typeControlRef
	kEventParamControlDrawDepth	typeShortInteger

	kEventParamGrafPort (Optional)	typeGrafPtr
kEventControlApply-TextColor	kEventParamDirectObject	typeControlRef
	kEventParamControlSubControl	typeControlRef
	kEventParamControlDrawDepth	typeShortInteger
	kEventParamControlDrawIn Color	typeBoolean
	kEventParamGrafPort (Optional)	typeGrafPtr
	kEventParamCGContextRef (Optional)	typeCGContextRef
kEventControlGetNext-Focus Candidate	kEventParamNextControl	typeControlRef
	kEventParamControlPart	typeControlPartCode
kEventControlGetAuto-Toggle Value	kEventParamDirectObject	typeControlRef
	kEventParamControlPart	typeControlPartCode
	kEventParamControlValue	typeLongInteger
kEventControl-Intercept Subview-Click	kEventParamEventRef	typeEventRef
kEventControlGet-Click Activation	kEventParamClickActivation	typeClickActivationResult
kEventControl-DragEnter	kEventParamDirectObject	typeControlRef
	kEventParamDragRef	typeDragRef
kEventControl-DragWithin	kEventParamDirectObject	typeControlRef
	kEventParamDragRef	typeDragRef
kEventControl-DragLeave	kEventParamDirectObject	typeControlRef
	kEventParamDragRef	typeDragRef
kEventControl-DragReceive	kEventParamDirectObject	typeControlRef
	kEventParamDragRef	typeDragRef

kEventControl-SetFocusPart	kEventParamDirectObject	typeControlRef
	kEventParamStartControl	typeControlRef
	kEventParamControlPart	typeControlPartCode
	kEventParamControlFocus Everything (Optional)	typeBoolean
kEventControl-GetFocusPart	kEventParamDirectObject	typeControlRef
	kEventParamControlPart	typeControlPartCode
kEventControl-Activate	kEventParamDirectObject	typeControlRef
kEventControl-Deactivate	kEventParamDirectObject	typeControlRef
kEventControl-SetCursor	kEventParamDirectObject	typeControlRef
	kEventParamMouseLocation	typeQDPoint
	kEventParamKeyModifiers	typeUInt32
kEventControl-ContextualMenu Click	kEventParamDirectObject	typeControlRef
	kEventParamMouseLocation	typeQDPoint
kEventControlTrack	kEventParamDirectObject	typeControlRef
	kEventParamMouseLocation	typeQDPoint
	kEventParamKeyModifiers	typeUInt32
	kEventParamControlAction	typeControlActionUPP
	kEventParamControlPart	typeControlPartCode
kEventControlGet-ScrollToHere Start-Point	kEventParamDirectObject	typeControlRef
	kEventParamMouseLocation	typeQDPoint
	kEventParamKeyModifiers	typeUInt32
kEventControlGet-Indicator Drag-Constraint	kEventParamDirectObject	typeControlRef

	kEventParamMouseLocation	typeQDPoint
	kEventParamKeyModifiers	typeUInt32
	kEventParamControlIndicator Drag-Constraint	typeIndicatorDrag-Constraint
kEventControl-IndicatorMoved	kEventParamDirectObject	typeControlRef
	kEventParamControlIndicator Region	typeQDRgnHandle
	kEventParamControlIsGhosting	typeBoolean
kEventControl-Ghosting Finished	kEventParamDirectObject	typeControlRef
	kEventParamControlIndicator Offset	typeQDPoint
kEventControlGet-ActionProc Part	kEventParamDirectObject	typeControlRef
	kEventParamKeyModifiers	typeUInt32
	kEventParamControlPart	typeControlPartCode
kEventControlGet-PartRegion	kEventParamDirectObject	typeControlRef
	kEventParamControlPart	typeControlPartCode
	kEventParamControlRegion	typeQDRgnHandle
kEventControlGet-PartBounds	kEventParamDirectObject	typeControlRef
	kEventParamControlPart	typeControlPartCode
	kEventParamControlBounds	typeQDRectangle
kEventControlSetData	kEventParamDirectObject	typeControlRef
	kEventParamControlPart	typeControlPartCode
	kEventParamControlDataTag	typeEnumeration
	kEventParamControlDataBuffer	typePtr
	kEventParamControlDataBufferSize	typeLongInteger
kEventControlGetData	kEventParamDirectObject	typeControlRef
	kEventParamControlPart	typeControlPartCode
	kEventParamControlDataTag	typeEnumeration

	kEventParamControlDataBuffer	typePtr
	kEventParamControlDataBuffer Size	typeLongInteger
kEventControlGetSize Constraints	kEventParamDirectObject	typeControlRef
	kEventParamMinimumSize	typeHISize
	kEventParamMaximumSize	typeHISize
kEventControlValue-Field Changed	kEventParamDirectObject	typeControlRef
kEventControlAdded-SubControl	kEventParamDirectObject	typeControlRef
	kEventParamControlSubControl	typeControlRef
kEventControl-Removing SubControl	kEventParamDirectObject	typeControlRef
	kEventParamControlSubControl	typeControlRef
kEventControlBounds-Changed	kEventParamDirectObject	typeControlRef
	kEventParamAttributes	typeUInt32
	kEventParamOriginalBounds	typeQDRectangle
	kEventParamPreviousBounds	typeQDRectangle
	kEventParamCurrentBounds	typeQDRectangle
kEventControlOwning-Window Changed	kEventParamDirectObject	typeControlRef
	kEventParamAttributes	typeUInt32
	kEventParamControlOriginal OwningWindow	typeWindowRef
	kEventParamControlCurrent OwningWindow	typeWindowRef
kEventControlHilite-State Changed	kEventParamDirectObject	typeControlRef
	kEventParamPreviousPart	typeControlPartCode
	kEventParamCurrentPart	typeControlPartCode
kEventControlEnabled-State Changed	kEventParamDirectObject	typeControlRef

kEventControl- Arbitrary Message	kEventParamDirectObject	typeControlRef
	kEventParamControlMessage	typeShortInteger
	kEventParamControlParam	typeLongInteger
	kEventParamControlResult	typeLongInteger

Control Bounds Constants

Define control bounds change-event attributes.

```
enum {
    kControlBoundsChangeSizeChanged = (1 << 2),
    kControlBoundsChangePositionChanged = (1 << 3)
};
```

Constants

`kControlBoundsChangeSizeChanged`

The dimensions of the control (width and height) changed.

Available in Mac OS X v10.0 and later.

`kControlBoundsChangePositionChanged`

The position of the control changed (that is, the top-left corner moved).

Available in Mac OS X v10.0 and later.

Discussion

When the system sends out a `kEventControlBoundsChanged` event, it also sends along a parameter containing attributes of the event. These attributes can be used to determine what aspect of the control changed (position, size, or both).

Control Event Parameters

Define parameters related to control events.

```
enum {
    kEventParamControlPart = 'cprt',
    kEventParamInitCollection = 'icol',
    kEventParamControlMessage = 'cmsg',
    kEventParamControlParam = 'cprm',
    kEventParamControlResult = 'crsl',
    kEventParamControlRegion = 'crgn',
    kEventParamControlAction = 'caup',
    kEventParamControlIndicatorDragConstraint = 'cidc',
    kEventParamControlIndicatorRegion = 'cirn',
    kEventParamControlIsGhosting = 'cgst',
    kEventParamControlIndicatorOffset = 'ciof',
    kEventParamControlClickActivationResult = 'ccar',
    kEventParamControlSubControl = 'csub',
    kEventParamControlOptimalBounds = 'cobn',
    kEventParamControlOptimalBaselineOffset = 'cobo',
    kEventParamControlDataTag = 'cdtg',
```

```
    kEventParamControlDataBuffer = 'cdbf',
    kEventParamControlDataBufferSize = 'cdbs',
    kEventParamControlDrawDepth = 'cddp',
    kEventParamControlDrawInColor = 'cdic',
    kEventParamControlFeatures = 'cftr',
    kEventParamControlPartBounds = 'cpbd',
    kEventParamControlOriginalOwningWindow = 'coow',
    kEventParamControlCurrentOwningWindow = 'ccow',
    kEventParamControlFocusEverything = 'cfev',
    kEventParamNextControl = 'cnxc',
    kEventParamStartControl = 'cstc',
    kEventParamControlSubview = 'csvw',
    kEventParamControlPreviousPart = 'copc',
    kEventParamControlCurrentPart = 'cnpc',
    kEventParamControlInvalRgn = 'civr',
    kEventParamControlValue = 'cval',
    kEventParamControlHit = 'chit',
    kEventParamControlPartAutoRepeats = 'caur',
    kEventParamControlFrameMetrics = 'cfmt',
    kEventParamControlWouldAcceptDrop = 'cldg',
    kEventParamControlPrefersShape = 'cpsh',
    typeControlActionUPP = 'caup',
    typeIndicatorDragConstraint = 'cidc',
    typeControlPartCode = 'cprt',
    typeControlFrameMetrics = 'cins'
};
```

Constants

`kEventParamControlPart`
typeControlPartCode

Available in Mac OS X v10.0 and later.

`kEventParamInitCollection`
typeCollection

Available in Mac OS X v10.0 and later.

`kEventParamControlMessage`
typeShortInteger

Available in Mac OS X v10.0 and later.

`kEventParamControlParam`
typeLongInteger

Available in Mac OS X v10.0 and later.

`kEventParamControlResult`
typeLongInteger

Available in Mac OS X v10.0 and later.

`kEventParamControlRegion`
typeQDRgnHandle

Available in Mac OS X v10.0 and later.

`kEventParamControlAction`
typeControlActionUPP

Available in Mac OS X v10.0 and later.

`kEventParamControlIndicatorDragConstraint`
typeIndicatorDragConstraint

Available in Mac OS X v10.0 and later.

`kEventParamControlIndicatorRegion`
typeQDRgnHandle

Available in Mac OS X v10.0 and later.

`kEventParamControlIsGhosting`
typeBoolean

Available in Mac OS X v10.0 and later.

`kEventParamControlIndicatorOffset`
typeQDPoint

Available in Mac OS X v10.0 and later.

`kEventParamControlClickActivationResult`
typeClickActivationResult

Available in Mac OS X v10.0 and later.

`kEventParamControlSubControl`
typeControlRef

Available in Mac OS X v10.0 and later.

`kEventParamControlOptimalBounds`
typeQDRectangle

Available in Mac OS X v10.0 and later.

`kEventParamControlOptimalBaselineOffset`
typeShortInteger

Available in Mac OS X v10.0 and later.

`kEventParamControlDataTag`
typeEnumeration

Available in Mac OS X v10.0 and later.

`kEventParamControlDataBuffer`
typePtr

Available in Mac OS X v10.0 and later.

`kEventParamControlDataBufferSize`
typeLongInteger

Available in Mac OS X v10.0 and later.

`kEventParamControlDrawDepth`
typeShortInteger

Available in Mac OS X v10.0 and later.

`kEventParamControlDrawInColor`
typeBoolean

Available in Mac OS X v10.0 and later.

`kEventParamControlFeatures`
typeUInt32

Available in Mac OS X v10.0 and later.

`kEventParamControlPartBounds`
typeQDRectangle

Available in Mac OS X v10.0 and later.

`kEventParamControlOriginalOwningWindow`
typeWindowRef

Available in Mac OS X v10.0 and later.

`kEventParamControlCurrentOwningWindow`
typeWindowRef

Available in Mac OS X v10.0 and later.

`kEventParamControlFocusEverything`
typeBoolean

Available in Mac OS X v10.2 and later.

`kEventParamNextControl`
typeControlRef

Available in Mac OS X v10.2 and later.

`kEventParamStartControl`
typeControlRef

Available in Mac OS X v10.2 and later.

`kEventParamControlSubview`
typeControlRef

Available in Mac OS X v10.2 and later.

`kEventParamControlPreviousPart`
typeControlPartCode

Available in Mac OS X v10.2 and later.

`kEventParamControlCurrentPart`
typeControlPartCode

Available in Mac OS X v10.2 and later.

`kEventParamControlInvalRgn`
typeQDRgnHandle

Available in Mac OS X v10.2 and later.

`kEventParamControlValue`
typeLongInteger

Available in Mac OS X v10.2 and later.

`kEventParamControlHit`
typeBoolean

Available in Mac OS X v10.3 and later.

`kEventParamControlPartAutoRepeats`
typeBoolean

Available in Mac OS X v10.3 and later.

`kEventParamControlFrameMetrics`
typeControlFrameMetrics

Available in Mac OS X v10.3 and later.

`kEventParamControlWouldAcceptDrop`
typeBoolean

Available in Mac OS X v10.3 and later.

`kEventParamControlPrefersShape`
typeBoolean

Available in Mac OS X v10.4 and later.

`typeControlActionUPP`
ControlActionUPP

Available in Mac OS X v10.0 and later.

`typeIndicatorDragConstraint`
IndicatorDragConstraint

Available in Mac OS X v10.0 and later.

`typeControlPartCode`
ControlPartCode

Available in Mac OS X v10.0 and later.

`typeControlFrameMetrics`
HIViewFrameMetrics

Available in Mac OS X v10.3 and later.

Ink Events

Ink Event Constants

Define constants related to events from `kEventClassInk`.

```
enum {
    kEventInkPoint = 10,
    kEventInkGesture = 11,
    kEventInkText = 12
};
```

Constants

`kEventInkPoint`

A mouse event will be handled as an ink point and used for recognition. The Ink Manager has determined that the mouse event in `kEventParamEventRef` should be used for recognition. If the application handles the event and returns `noErr`, the Ink Manager does nothing further with the mouse event. If the application returns any other value (including `eventNotHandledErr`), the Ink Manager processes the point normally.

Available in Mac OS X v10.3 and later.

`kEventInkGesture`

The Ink Manager recognizes the current ink phrase as one of the known system gestures. Applications can install a handler for these events to provide targeted gestures and support for context-dependent (tentative) gestures. Applications should return `noErr` if they handled the gesture. If the gesture was context dependent and does not apply to the current situation, applications should return `eventNotHandledErr`.

Available in Mac OS X v10.3 and later.

`kEventInkText`

The Ink Manager recognizes a word. The `kEventParamInkTextRef` parameter contains the ink text reference with all the information about the word. For more information, see `Ink.h`.

Available in Mac OS X v10.3 and later.

Discussion

Table 6 shows the parameters associated with ink events

Table 5 Parameter names and types for ink event kinds

Event kind	Parameter name	Parameter type
`kEventInkPoint`	`kEventParamEventRef`	`typeEventRef`
`kEventInkGesture`	`kEventParamInkGesture`	`typeHIRect`
	`kEventParamInkGestureBounds`	`typeHIRect`
	`kEventParamInkGestureHotspot`	`typeHIPoint`
`kEventInkText`	`kEventParamInkTextRef`	`typePtr`
	`kEventParamInkKeyboardShortcut`	`typeBoolean`

Availability
Available in Mac OS X v10.3 and later.

Ink Event Parameters

Define constants for parameters to ink events.

```
enum {
    kEventParamInkTextRef = 'iwrd',
    kEventParamInkKeyboardShortcut = 'ikbd',
    kEventParamInkGestureKind = 'gknd',
    kEventParamInkGestureBounds = 'gbnd',
    kEventParamInkGestureHotspot = 'ghot'
};
```

Constants

`kEventParamInkTextRef`
The ink text reference containing the data for the word the Ink Manager recognized. (`typePtr`)

Available in Mac OS X v10.3 and later.

`kEventParamInkKeyboardShortcut`
A Boolean whose value indicates whether the word the Ink Manager recognized is a keyboard shortcut. That is, the Command or Control key was pressed and the top-choice alternate text is a single character. (`typeBoolean`)

Available in Mac OS X v10.3 and later.

`kEventParamInkGestureKind`
Kind of gesture. (`typeUInt32`)

Available in Mac OS X v10.3 and later.

`kEventParamInkGestureBounds`
Bounds of the gesture in global coordinates. (`typeHIRect`)

Available in Mac OS X v10.3 and later.

`kEventParamInkGestureHotspot`
Hotspot, in global coordinates, for the gesture. (`typeHIPoint`)

Available in Mac OS X v10.3 and later.

Availability
Available in Mac OS X v10.3 and later.

Keyboard Events

Keyboard Event Constants

Define constants related to events from `kEventClassKeyboard`.

```
enum {
    kEventRawKeyDown = 1,
```

```
    kEventRawKeyRepeat = 2,
    kEventRawKeyUp = 3,
    kEventRawKeyModifiersChanged = 4,
    kEventHotKeyPressed = 5,
    kEventHotKeyReleased = 6
};
```

Constants

`kEventRawKeyDown`
A key was pressed.

Available in Mac OS X v10.0 and later.

`kEventRawKeyRepeat`
Sent periodically as a key is held down by the user.

Available in Mac OS X v10.0 and later.

`kEventRawKeyUp`
A key was released.

Available in Mac OS X v10.0 and later.

`kEventRawKeyModifiersChanged`
The keyboard modifiers have changed.

Available in Mac OS X v10.0 and later.

`kEventHotKeyPressed`
A registered hot key was pressed.

Available in Mac OS X v10.0 and later.

`kEventHotKeyReleased`
A registered hot key was released.

Available in Mac OS X v10.0 and later.

Discussion

These events are the lowest-level keyboard events.

Table 6 shows the parameters associated with keyboard events.

Table 6 Parameter names and types for keyboard event kinds

Event kind	Parameter name	Parameter type
`kEventRawKeyDown`	`kEventParamKeyMacCharCodes`	`typeChar`
	`kEventParamKeyCode`	`typeUInt32`
	`kEventParamKeyModifiers`	`typeUInt32`
	`kEventParamKeyboardType`	`typeUInt32`
`kEventRawKeyRepeat`	`kEventParamKeyMacCharCodes`	`typeChar`
	`kEventParamKeyCode`	`typeUInt32`

	kEventParamKeyModifiers	typeUInt32
	kEventParamKeyboardType	typeUInt32
kEventRawKeyUp	kEventParamKeyMacCharCodes	typeChar
	kEventParamKeyCode	typeUInt32
	kEventParamKeyModifiers	typeUInt32
	kEventParamKeyboardType	typeUInt32
kEventRawKey-ModifiersChanged	kEventParamKeyModifiers	typeUInt32
kEventHotKeyPressed	kEventParamDirectObject	typeEventHotKeyID
kEventHotKeyReleased	kEventParamDirectObject	typeEventHotKeyID

Key Modifier Event Masks

Define values used to determine whether additional modifier keys are down for a keyboard or mouse event.

```
enum {
    kEventKeyModifierNumLockMask = 1L << kEventKeyModifierNumLockBit,
    kEventKeyModifierFnMask = 1L << kEventKeyModifierFnBit
};
```

Constants

`kEventKeyModifierNumLockMask`
A bit mask containing `kEventKeyModifierNumLockBit`. (Mac OS X only)

Available in Mac OS X v10.0 and later.

`kEventKeyModifierFnMask`
A bit mask containing `kEventKeyModifierFnBit`. (Mac OS X only)

Available in Mac OS X v10.0 and later.

Key Modifier Event Bits

Define key modifier change event bits.

```
enum {
    kEventKeyModifierNumLockBit = 16,
    kEventKeyModifierFnBit = 17
};
```

Constants

`kEventKeyModifierNumLockBit`

This keyboard event was generated either on the numeric keypad or in the numeric section of an iBook or PowerBook keyboard with the Num Lock key pressed. This state bit does not provide an indication of the Num Lock key on non-portable keyboards. This bit is set on Mac OS X only.

Available in Mac OS X v10.0 and later.

`kEventKeyModifierFnBit`

The Fn key was pressed when this keyboard event was generated. This bit is set on Mac OS X only.

Available in Mac OS X v10.0 and later.

Discussion

Note that bits 8 through 15 (`cmdKeyBit` to `rightControlKeyBit`) are compatible with the Classic Event Manager modifiers.

Keyboard Event Parameters and Types

Define constants for parameters to raw keyboard events.

```
enum {
    kEventParamKeyCode = 'kcod',
    kEventParamKeyMacCharCodes = 'kchr',
    kEventParamKeyModifiers = 'kmod',
    kEventParamKeyUnicodes = 'kuni',
    kEventParamKeyboardType = 'kbdt',
    typeEventHotKeyID = 'hkid'
};
```

Constants

`kEventParamKeyCode`

typeUInt32

Available in Mac OS X v10.0 and later.

`kEventParamKeyMacCharCodes`

typeChar

Available in Mac OS X v10.0 and later.

`kEventParamKeyModifiers`

typeUInt32

Available in Mac OS X v10.0 and later.

`kEventParamKeyUnicodes`

typeUnicodeText

Available in Mac OS X v10.0 and later.

`typeEventHotKeyID`

EventHotKeyID

Available in Mac OS X v10.0 and later.

Symbolic Hot Key Definitions

Define CFDictionaryRef keys returned by `CopySymbolicHotKeys`.

```
#define kHISymbolicHotKeyCode CFSTR("kHISymbolicHotKeyCode")
#define kHISymbolicHotKeyModifiers CFSTR("kHISymbolicHotKeyModifiers")
#define kHISymbolicHotKeyEnabled CFSTR("kHISymbolicHotKeyEnabled")
```

Constants

`kHISymbolicHotKeyCode`
The virtual key code of the hot key, represented as a CFNumber.

Available in Mac OS X v10.3 and later.

`kHISymbolicHotKeyModifiers`
The hot key's keyboard modifiers, represented as a CFNumber.

Available in Mac OS X v10.3 and later.

`kHISymbolicHotKeyEnabled`
The enable state of the hot key, represented as a CFBoolean.

Available in Mac OS X v10.3 and later.

Availability

Available in Mac OS X v10.3 and later.

Hot Key Constants

Define hot key states used by `PushSymbolicHotKeyMode`.

```
enum {
    kHIHotKeyModeAllEnabled = 0,
    kHIHotKeyModeAllDisabled = (1 << 0),
    kHIHotKeyModeAllDisabledExceptUniversalAccess = (1 <<  1)
};
```

Constants

`kHIHotKeyModeAllEnabled`
All hot keys are enabled.

Available in Mac OS X v10.4 and later.

`kHIHotKeyModeAllDisabled`
All hot keys are disabled.

Available in Mac OS X v10.4 and later.

`kHIHotKeyModeAllDisabledExceptUniversalAccess`
All hot keys are disabled except for the Universal Access hot keys (that is, zooming, white-on-black, and enhanced contrast).

Available in Mac OS X v10.4 and later.

Menu Events

Menu Event Constants

Define constants related to events from `kEventClassMenu`.

```
enum {
    kEventMenuBeginTracking = 1,
    kEventMenuEndTracking = 2,
    kEventMenuChangeTrackingMode = 3,
    kEventMenuOpening = 4,
    kEventMenuClosed = 5,
    kEventMenuTargetItem = 6,
    kEventMenuMatchKey = 7,
    kEventMenuEnableItems = 8,
    kEventMenuPopulate = 9,
    kEventMenuMeasureItemWidth = 100,
    kEventMenuMeasureItemHeight = 101,
    kEventMenuDrawItem = 102,
    kEventMenuDrawItemContent = 103,
    kEventMenuDispose = 1001,
    kEventMenuCalculateSize = 1004,
    kEventMenuCreateFrameView = 1005,     kEventMenuGetFrameBounds = 1006,
kEventMenuBecomeScrollable = 1007,     kEventMenuCeaseToBeScrollable = 1008,
    kEventMenuBarShown = 2000,     kEventMenuBarHidden = 2001
};
```

Constants

`kEventMenuBeginTracking`

The user has begun tracking the menubar or a pop-up menu. The direct object parameter is a valid menu reference if tracking a pop-up menu, or `NULL` if tracking the menubar. The `kEventParamCurrentMenuTrackingMode` parameter indicates whether the user is tracking the menus using the mouse or the keyboard. The handler may return `userCanceledErr` to stop menu tracking. (Available in CarbonLib 1.1 and Mac OS X v10.0 and later.)

Available in Mac OS X v10.0 and later.

`kEventMenuEndTracking`

The user has finished tracking the menubar or a pop-up menu. In Mac OS X v10.3 and later, when a menu tracking session ends, the Menu Manager sends `kEventMenuEndTracking` to every menu that was opened during the session, in addition to the root menu. This is done to allow menus with dynamic content to remove that content at the end of menu tracking; for example, a menu containing many IconRefs might want to load the IconRefs dynamically in response to the `kEventMenuMenuPopulate` event and remove them in response to the `kEventMenuEndTracking` event to avoid the memory overhead of keeping the IconRef data in memory while the menu is not being displayed. (Available in CarbonLib 1.1 and Mac OS X v10.0 and later.)

Available in Mac OS X v10.0 and later.

`kEventMenuChangeTrackingMode`
The user has switched from selecting a menu with the mouse to selecting with the keyboard, or from selecting with the keyboard to selecting with the mouse. (Available in Mac OS X v10.1 and later.)

Available in Mac OS X v10.0 and later.

`kEventMenuOpening`
A menu is opening. This event is sent each time that the menu is opened (that is, more than once during a given tracking session if the user opens the menu multiple times). It is sent before the menu is actually drawn, so you can update the menu contents (including making changes that will alter the menu size) and the new contents will be drawn correctly. The `kEventParamMenuFirstOpen` parameter indicates whether this is the first time this menu has been opened during this menu tracking session. The handler may return `userCanceledErr` to prevent this menu from opening. Note that for most applications, you should handle the `kEventMenuPopulate` event instead. (Available in CarbonLib 1.1 and Mac OS X v10.0 and later.)

Available in Mac OS X v10.0 and later.

`kEventMenuClosed`
A menu has been closed. Sent after the menu is hidden. (Available in CarbonLib 1.1 and later and Mac OS X v10.0 and later.)

Available in Mac OS X v10.0 and later.

`kEventMenuTargetItem`
The mouse is moving over a particular menu item. This event is sent for both enabled and disabled items. (Available in CarbonLib 1.1 and later and Mac OS X v10.0 and later.)

Available in Mac OS X v10.0 and later.

`kEventMenuMatchKey`

A menu is about to be examined for items that match a command key event. A handler for this event may perform its own command key matching and override the Menu Manager's default matching algorithms. Returning `noErr` from your handler indicates that you have found a match. The handler for this event should not examine submenus of this menu for a match; a separate event will be sent for each submenu.

When called from `IsMenuKeyEvent`, the `kEventParamEventRef` parameter contains the event reference that was passed to `IsMenuKeyEvent`, for your handler to examine; when called from `MenuKey` or `MenuEvent`, the `EventRef` parameter contains an event created from the information passed to `MenuKey` or `MenuEvent`. Note that in the `MenuKey` case, no virtual key code (`kEventParamKeyCode`) or key modifiers (`kEventParamKeyModifiers`) will be available.

The `kEventParamMenuEventOptions` parameter contains a copy of the options that were passed to `IsMenuKeyEvent`, or `0` if called from `MenuKey` or `MenuEvent`. The only option that your handler will need to obey is `kMenuEventIncludeDisabledItems`.

If your handler finds a match, it should set the `kEventParamMenuItemIndex` parameter to contain the item index of the matching item, and return `noErr`. If it does not find a match, it should return `menuItemNotFoundErr`. Any other return value will cause the Menu Manager to use its default command key matching algorithm for this menu.

This event is sent after `kEventMenuEnableItems`.

In CarbonLib and Mac OS X through version 10.3, the Menu Manager sends a `kEventMenuEnableItems` event to the menu before sending `kEventMenuMatchKey`. In Mac OS X 10.4 and later, the Menu Manager no longer sends `kEventMenuEnableItems` (or the resulting `kEventCommandUpdateStatus` events) to the menu; a handler for `kEventMenuMatchKey` is expected to determine on its own whether a matching menu item is enabled.

Available in Mac OS X v10.0 and later.

`kEventMenuEnableItems`

A request that the items in the menu be properly enabled or disabled according to the current state of the application. This event is sent from inside `MenuKey`, `MenuEvent`, and `IsMenuKeyEvent` before those functions examine the menu for an item that matches a keyboard event. It is also sent during menu tracking before a menu is first made visible; it is sent right after `kEventMenuOpening`, once per menu per menu tracking session.

If you install an event handler for `kEventProcessCommand`, you should also install a handler for `kEventMenuEnableItems`. This is necessary because the Carbon event system will attempt to match command keys against the available menus before returning the keyboard event to your application via `WaitNextEvent`. If you have menu command event handlers installed for your menu items, your handlers will be called without you ever receiving the keyboard event or calling `MenuKey/MenuEvent/IsMenuKeyEvent` yourself. Therefore, you have no opportunity to enable your menu items properly other than from a `kEventMenuEnableItems` handler.

It is not necessary to handle this event if you do not install `kEventProcessCommand` handlers for your menu items; in that case, the command key event will be returned from `WaitNextEvent` or `ReceiveNextEvent` as normal, and you can set up your menus before calling `MenuKey/MenuEvent/IsMenuKeyEvent`.

The `kEventParamEnableMenuForKeyEvent` parameter indicates whether this menu should be enabled for key event matching (`true`) or because the menu itself is about to become visible (`false`). If true, only the item enable state, command key, command key modifiers, and (optionally) the command key glyph need to be correct. If false, the entire menu item contents must be correct. This may be useful if you have custom menu content that is expensive to prepare.

Note that the standard application handler for `kEventMenuEnableItems` automatically sends `kEventCommandUpdateStatus` to your menu commands. As this can cause a performance hit if you have many menu items, you can choose to bypass these updates by installing a no-op handler for `kEventMenuEnableItems` that simply returns `noErr`.

(Available in CarbonLib 1.1 and later, and Mac OS X v10.0 and later.)

Available in Mac OS X v10.0 and later.

`kEventMenuPopulate`

Sent when an application should dynamically create a menu. You should use this event instead of `kEventMenuOpening` as the Menu Manager sends it before it searches a menu for a matching command key sequence; therefore you can use this event to dynamically add menu items that have command-key equivalents, making them selectable even if the menu itself is never displayed. The `kEventMenuPopulate` event is sent just once during a menu tracking session, even if the menu is opened and closed multiple times; the `kEventWindowOpening` event is sent each time the menu opens.

To determine whether this event was triggered by a command-key sequence, you should examine the `kEventParamMenuContext` parameter for the `kMenuContextKeyMatching` flag. If the event corresponds to the actual display of a menu, the `kEventContextMenuBarTracking` or `kEventContextPopUpTracking` flags are set.

(Available in CarbonLib 1.5 and later, and Mac OS X v10.1 and later. Note that in CarbonLib 1.6 and Mac OS X v10.2 and later, the Menu Manager sends this event before it searches a menu for a matching command ID. To determine if this event was triggered by a command ID, check for the `kMenuContextCommandIDSearch` flag in the `kEventParamMenuContext` parameter. If that flag is set, the command ID that triggered this event is contained in the `kEventParamMenuCommand` parameter.)

Available in Mac OS X v10.1 and later.

`kEventMenuMeasureItemWidth`

Sent by the standard window definition if a menu item has the `kMenuItemAttrCustomDraw` attribute set. You should install your handler directly on the menu. Your handler should return a customized width for the menu item in the `kEventParamMenuItemWidth` parameter.

The default behavior (if you are using the standard menu definition) is to return the standard width for the item.

(Available in CarbonLib 1.5 and later, and Mac OS X v10.1 and later.)

Available in Mac OS X v10.1 and later.

`kEventMenuMeasureItemHeight`

Sent by the standard window definition if a menu item has the `kMenuItemAttrCustomDraw` attribute set. You should install your handler directly on the menu. Your handler should return a customized height for the menu item in the `kEventParamMenuItemHeight` parameter.

The default behavior (if you are using the standard menu definition) is to return the standard height for the item.

(Available in CarbonLib 1.5 and later, and Mac OS X v10.1 and later.)

Available in Mac OS X v10.1 and later.

`kEventMenuDrawItem`

Sent by the standard window definition if a menu item has the `kMenuItemAttrCustomDraw` attribute set. You should install your handler directly on the menu. Use your handler to override the drawing of the menu item and background.

The default behavior (if you are using the standard menu definition) is to call the Appearance Manager function `DrawThemeMenuItem` to draw the menu item's background and content.

If you have the standard event handler installed, the event also contains additional parameters indicating the bounds of the various portions of the menu item content, as well as the baseline of the menu item text. Using these parameters, you can call `CallNextEventHandler` to let the system handlers draw the standard menu content and then your handler can draw custom content on top.

(Available in CarbonLib 1.5 and later, and Mac OS X v10.1 and later.)

Available in Mac OS X v10.1 and later.

`kEventMenuDrawItemContent`

Sent by the standard window definition if a menu item has the `kMenuItemAttrCustomDraw` attribute set. You should install your handler directly on the menu. You use your handler to override the drawing of one or more parts of the menu item's content (that is the mark character, icon, text, and command key information).

When you receive this event, the background and highlighting (if applicable) has already been drawn using the standard system appearance.

The default behavior (if you are using the standard menu definition) is to draw the standard menu item content.

If you have the standard event handler installed, the event also contains additional parameters indicating the bounds of the various portions of the menu item content, as well as the baseline of the menu item text. Using these parameters, you can call `CallNextEventHandler` to let the system handlers draw the standard menu content and then your handler can draw custom content on top.

(Available in CarbonLib 1.5 and later, and Mac OS X v10.1 and later.)

Available in Mac OS X v10.1 and later.

`kEventMenuDispose`

Sent when a menu is being disposed. (Available in CarbonLib 1.1 and later and Mac OS X v10.0 and later.)

Available in Mac OS X v10.0 and later.

`kEventMenuCalculateSize`

Sent by `CalcMenuSize` to request that the menu calculate its size. The Menu Manager provides a default handler for all menus that calls the menu's MDEF or menu content view to determine the menu size. Applications typically do not need to handle this event. A custom menu definition or menu content view should use `kMenuSizeMsg` or `kEventControlGetOptimalBounds` to calculate its size. Note that if the menu uses an MDEF, the MDEF sets the menu's width and height in response to `kMenuSizeMsg`. The default handler for this event saves the old width and height before calling the MDEF and restores them afterward. `CalcMenuSize` sets the final menu width and height based on the dimensions returned from this event; applications may override this event to customize the width or height of a menu by modifying the `kEventParamDimensions` parameter. This event is sent only to the menu and is not propagated past the menu. (Available in Mac OS X v10.3 and later.)

Available in Mac OS X v10.3 and later.

`kEventMenuCreateFrameView`

Requests that a menu content view create the HIView that will be used to draw the menu window frame. The HIMenuView class provides a default handler for this event that creates an instance of the standard menu window frame view. This event is sent only to the menu content view and is not propagated past the view. (Available in Mac OS X v10.3 and later.)

Available in Mac OS X v10.3 and later.

`kEventMenuGetFrameBounds`

Requests that a menu content view calculate the bounding rect, in global coordinates, of the menu window frame that should contain the menu. This event is sent by the Menu Manager before displaying pull-down, popup, and hierarchical menus. It provides an opportunity for the menu content view to determine the position of the menu frame based on the position of the menu title, parent menu item, or popup menu location. The HIMenuView class provides a default handler for this event that calculates an appropriate location based on the bounds of the menu, the available screen space, and the frame metrics of the menu window content view. This event is sent only to the menu content view and is not propagated past the view. (Available in Mac OS X v10.3 and later.)

Available in Mac OS X v10.3 and later.

`kEventMenuBecomeScrollable`

Requests that a menu content view prepare to be scrollable, which it does by installing the appropriate event handlers, timers, and the like. This event is sent by the Menu Manager when a menu becomes the most recently opened menu in the menu hierarchy. It is an indication that this menu content view is now a candidate for scrolling. The Menu Manager provides a default handler for this event that installs event handlers to provide automatic scrolling behavior for HIView-based menus. If a menu content view does not wish to use the Menu Manager's default scrolling support, it can override this event and return `noErr` to prevent the event from being propagated to the Menu Manager's default handler. This event is sent only to the menu content view and is not propagated past the view. (Available in Mac OS X v10.3 and later.)

Available in Mac OS X v10.3 and later.

`kEventMenuCeaseToBeScrollable`
Requests that a menu content view cease to be scrollable. This event is sent by the Menu Manager when a menu ceases to be the most recently opened menu. This occurs when the menu is closed, or when a submenu of the most recently opened menu is opened. It is an indication that this menu content view is no longer a candidate for scrolling. The Menu Manager provides a default handler for this event that removes event handlers installed in response to `kEventMenuBecomeScrollable`. This event is sent only to the menu content view and is not propagated past the view. (Available in Mac OS X v10.3 and later.)

Available in Mac OS X v10.3 and later.

`kEventMenuBarShown`
Sent to all handlers registered for this event when the front process shows its menubar. This event is sent only to the application target. (Available in Mac OS X v10.3 and later.)

Available in Mac OS X v10.3 and later.

`kEventMenuBarHidden`
Sent to all handlers registered for this event when the front process hides its menubar. This event is sent only to the application target.(Available in Mac OS X v10.3 and later.)

Available in Mac OS X v10.3 and later.

Discussion
Some menu events are sent or handled by the standard menu definition, which is a collection of handlers that define the default menu behavior. If you have specified a custom menu definition, you will not get the behavior provided by the standard definition.

Table 7 shows the parameters associated with menu events.

Table 7 Parameter names and types for menu event kinds

Event kind	Parameter name	Parameter type
`kEventMenuBegin-Tracking`	`kEventParamDirectObject`	`typeMenuRef`
	`kEventParamCurrentMenuTrackingMode`	`typeMenuTrackingMode`
	`kEventParamMenuContext` (Mac OS X v10.1 and later and CarbonLib 1.5 and later.)	`typeUInt32`
`kEventMenuEnd-Tracking`	`kEventParamDirectObject`	`typeMenuRef`
	`kEventParamMenuContext` (Mac OS X v10.1 and later and CarbonLib 1.5 and later.)	`typeUInt32`
	`kEventParamMenuDismissed` (Mac OS X v10.3 and later.)	`typeUInt32`
`kEventMenuChange-Tracking Mode`	`kEventParamDirectObject`	`typeMenuRef`
	`kEventParamCurrentMenuTrackingMode`	`typeMenuTrackingMode`

	kEventParamNewMenuTrackingMode	typeMenuTrackingMode
	kEventParamMenuContext	typeUInt32
kEventMenuOpening	kEventParamDirectObject	typeMenuRef
	kEventParamMenuFirstOpen	typeBoolean
	kEventParamMenuContext (Mac OS X v 10.0 and later and CarbonLib 1.5 and later.)	typeUInt32
kEventMenuClosed	kEventParamDirectObject	typeMenuRef
	kEventParamMenuContext (Mac OS X v10.1 and later and CarbonLib 1.5 an later.)	typeUInt32
kEventMenuTargetItem	kEventParamDirectObject	typeMenuRef
	kEventParamMenuItemIndex	typeMenuItemIndex
	kEventParamMenuCommand	typeMenuCommand
	kEventParamMenuContext (Mac OS X v10.1 and later and CarbonLib 1.5 and later.)	typeUInt32
kEventMenuMatchKey	kEventParamDirectObject	typeMenuRef
	kEventParamEventRef	typeEventRef
	kEventParamMenuEventOptions	typeMenuEventOptions
	kEventParamMenuContext (Mac OS X v10.1 and later, and CarbonLib 1.5 and later.)	typeUInt32
	kEventParamMenuItemIndex	typeMenuItemIndex
kEventMenuEnable-Items	kEventParamDirectObject	typeMenuRef
	kEventParamEnableMenuForKeyEvent	typeBoolean
	kEventParamMenuContext (Mac OS X v10.0 and later, and CarbonLib 1.1 and later.)	typeUInt32
kEventMenuPopulate	kEventParamDirectObject	typeMenuRef
	kEventParamMenuContext	typeUInt32
	kEventParamMenuCommand (Mac OS X v10.2 and later, and CarbonLib 1.6 and later.)	typeMenuCommand
kEventMenuMeasure-ItemWidth	kEventParamDirectObject	typeMenuRef
	kEventParamMenuItemIndex	typeMenuItemIndex

	kEventParamMenuItemWidth	typeShortInteger
kEventMenuMeasure-ItemHeight	kEventParamDirectObject	typeMenuRef
	kEventParamMenuItemIndex	typeMenuItemIndex
	kEventParamMenuItemHeight	typeShortInteger
kEventMenuDrawItem	kEventParamDirectObject	typeMenuRef
	kEventParamCurrentBounds	typeQDRectangle
	kEventParamMenuItemIndex	typeMenuItemIndex
	kEventParamMenuItemBounds	typeQDRectangle
	kEventParamMenuVirtualTop	typeLongInteger
	kEventParamMenuVirtualBottom	typeLongInteger
	kEventParamMenuDrawState	typeThemeMenuState
	kEventParamMenuItemType	typeThemeMenuItemType
	kEventParamCGContextRef	typeCGContextRef
	kEventParamMenuMarkBounds	typeQDRectangle
	kEventParamMenuIconBounds	typeQDRectangle
	kEventParamMenuTextBounds	typeQDRectangle
	kEventParamMenuTextBaseline	typeShortInteger
	kEventParamMenuKeyCommandKeyBounds	typeQDRectangle
kEventMenuDraw-ItemContent	kEventParamDirectObject	typeMenuRef
	kEventParamMenuItemIndex	typeMenuItemIndex
	kEventParamMenuItemBounds	typeQDRectangle
	kEventParamDeviceDepth	typeShortInteger
	kEventParamDeviceColor	typeBoolean
	kEventParamCGContextRef	typeCGContextRef
	kEventParamMenuMarkBounds	typeQDRectangle
	kEventParamMenuIconBounds	typeQDRectangle
	kEventParamMenuTextBounds	typeQDRectangle

	kEventParamMenuTextBaseline	typeShortInteger
	kEventParamMenuKeyCommandKeyBounds	typeQDRectangle
kEventMenuDispose	kEventParamDirectObject	typeMenuRef
kEventMenuCalculate-Size	kEventParamDirectObject	typeMenuRef
	kEventParamControlRef	typeControlRef
	kEventParamGDevice	typeGDHandle
	kEventParamAvailableBounds	typeQDRectangle
	kEventParamDimensions	typeHISize
kEventMenuCreate-FrameView	kEventParamEventRef	typeEventRef
	kEventParamMenuType	typeThemeMenuType
	kEventParamMenuFrameView	typeControlRef
kEventMenuGet-FrameBounds	kEventParamMenuType	typeThemeMenuType
	kEventParamMenuIsPopup	typeBoolean
	kEventParamMenuFrameView	typeControlRef
	kEventParamMenuDirection	typeMenuDirection
	kEventParamMenuItemBounds	typeHIRect
	kEventParamGDevice	typeGDHandle
	kEventParamAvailableBounds	typeHIRect
	kEventParamParentMenu	typeMenuRef
	kEventParamParentMenuItem	typeMenuItemIndex
	kEventParamMenuPopupItem	typeMenuItemIndex
	kEventParamBounds	typeHIRect
	kEventParamOrigin	typeHIPoint
kEventMenuBecome-Scrollable	*None*	
kEventMenuCease-ToBeScrollable	*None*	
kEventMenuBarShown	*None*	

kEventMenuBarHidden	*None*	

Menu Context Constants

Define constants that describe the usage or context of a menu event.

```
enum {
    kMenuContextMenuBar = 1 << 0,
    kMenuContextPullDown = 1 << 8,
    kMenuContextPopUp = 1 << 9,
    kMenuContextSubmenu = 1 << 10,
    kMenuContextMenuBarTracking = 1 << 16,
    kMenuContextPopUpTracking = 1 << 17,
    kMenuContextKeyMatching = 1 << 18,
    kMenuContextMenuEnabling = 1 << 19,
    kMenuContextCommandIDSearch = 1 << 20
};
```

Constants

`kMenuContextMenuBar`
The menu associated with this event is in the menu bar or is a submenu of a menu in the menubar.

Available in Mac OS X v10.1 and later.

`kMenuContextPulldown`
The menu associated with this event is a pulldown menu located in the menu bar.

`kMenuContextPopUp`
The menu associated with this event is a popup menu displayed by the Menu Manager function `PopUpMenuSelect`.

Available in Mac OS X v10.1 and later.

`kMenuContextSubmenu`
The menu associated with this event is a submenu of a pulldown or popup menu.

Available in Mac OS X v10.1 and later.

`kMenuContextMenuBarTracking`
This event is being sent while a menu is being tracked in the menu bar.

Available in Mac OS X v10.1 and later.

`kMenuContextPopUpTracking`
This event is being sent while a popup menu is being tracked.

Available in Mac OS X v10.1 and later.

`kMenuContextKeyMatching`
This event is being sent while trying to match a command-key equivalent to a menu item.

Available in Mac OS X v10.1 and later.

`kMenuContextMenuEnabling`
Sent at idle time to update the enabled state of the menus. (Available in CarbonLib 1.5 and later, and Mac OS X v10.1 and later; on earlier releases the `kMenuContextKeyMatching` flag is set when an event is sent during menu enabling.)

Available in Mac OS X v10.1 and later.

`kMenuContextCommandIDSearch`
Sent while trying to match a command ID using the Menu Manager function `CountMenuItemsWithCommandID` or `GetIndMenuItemWithCommandID`. (Both functions are described in the Menu Manager Reference in the User Experience section of the Carbon documentation.) (Available in CarbonLib 1.6 and Mac OS X v10.2 and later.)

Available in Mac OS X v10.2 and later.

Discussion
The bits corresponding to these constants are set in the `kEventParamMenuContext` parameter of the menu event.

Menu Event Parameters

Define constants for parameters to menu events.

```
enum {
    kEventParamCurrentMenuTrackingMode = 'cmtm',
    kEventParamNewMenuTrackingMode = 'nmtm',
    kEventParamMenuFirstOpen = '1sto',
    kEventParamMenuItemIndex = 'item',
    kEventParamMenuCommand = 'mcmd',
    kEventParamEnableMenuForKeyEvent = 'fork',
    kEventParamMenuEventOptions = 'meop',
    kEventParamMenuContext = 'mctx',
    kEventParamMenuItemBounds = 'mitb',
    kEventParamMenuMarkBounds = 'mmkb',
    kEventParamMenuIconBounds = 'micb',
    kEventParamMenuTextBounds = 'mtxb',
    kEventParamMenuTextBaseline = 'mtbl',
    kEventParamMenuCommandKeyBounds = 'mcmb',
    kEventParamMenuVirtualTop = 'mvrt',
    kEventParamMenuVirtualBottom = 'mvrb',
    kEventParamMenuDrawState = 'mdrs',
    kEventParamMenuItemType = 'mitp',
    kEventParamMenuItemWidth = 'mitw',
    kEventParamMenuItemHeight = 'mith',
    typeMenuItemIndex = 'midx',
    typeMenuCommand = 'mcmd',
    typeMenuTrackingMode = 'mtmd',
    typeMenuEventOptions = 'meop',
    typeThemeMenuState = 'tmns',
    typeThemeMenuItemType = 'tmit'
};
```

Constants
`kEventParamCurrentMenuTrackingMode`
typeMenuTrackingMode

Available in Mac OS X v10.0 and later.

`kEventParamNewMenuTrackingMode`
typeMenuTrackingMode

Available in Mac OS X v10.0 and later.

`kEventParamMenuFirstOpen`
typeBoolean

Available in Mac OS X v10.0 and later.

`kEventParamMenuItemIndex`
typeMenuItemIndex

Available in Mac OS X v10.0 and later.

`kEventParamMenuCommand`
typeMenuCommand

Available in Mac OS X v10.0 and later.

`kEventParamEnableMenuForKeyEvent`
typeBoolean

Available in Mac OS X v10.0 and later.

`kEventParamMenuEventOptions`
typeMenuEventOptions

Available in Mac OS X v10.0 and later.

`typeMenuItemIndex`
MenuItemIndex

Available in Mac OS X v10.0 and later.

`typeMenuCommand`
MenuCommand

Available in Mac OS X v10.0 and later.

`typeMenuTrackingMode`
MenuTrackingMode

Available in Mac OS X v10.0 and later.

`typeMenuEventOptions`
MenuEventOptions

Available in Mac OS X v10.0 and later.

Services Menu Command Keys

Define CFDictionaryRef keys returned by `CopyServicesMenuCommandKeys`.

```
#define kHIServicesMenuProviderName CFSTR("kHIServicesMenuProviderName")
#define kHIServicesMenuItemName CFSTR("kHIServicesMenuItemName")
#define kHIServicesMenuCharCode CFSTR("kHIServicesMenuCharCode")
#define kHIServicesMenuKeyModifiers CFSTR("kHIServicesMenuKeyModifiers")
```

Constants

`kHIServicesMenuProviderName`
The name of the service provider.

`kHIServicesMenuItemName`
The name of the menu item.

`kHIServicesMenuCharCode`
The character code of the menu item shortcut.

`kHIServicesMenuKeyModifiers`
The keyboard modifiers of the menu item shortcut in Menu Manager modifiers format.

Availability
Available in Mac OS X v10.4 and later.

Mouse Events

Mouse Events

Define constants related to events from `kEventClassMouse`.

```
enum {
    kEventMouseDown = 1,
    kEventMouseUp = 2,
    kEventMouseMoved = 5,
    kEventMouseDragged = 6,
    kEventMouseEntered = 8,
    kEventMouseExited = 9,
    kEventMouseWheelMoved = 10
};
```

Constants

`kEventMouseDown`
A mouse button was pressed. Note that if you install a handler for this event on a window, you must allow the event to propagate (either by calling `CallNextEventHandler` or returning `eventNotHandledErr`) so that the window can be activated.

Available in Mac OS X v10.0 and later.

`kEventMouseUp`
A mouse button was released.

Available in Mac OS X v10.0 and later.

`kEventMouseMoved`
The mouse moved.

Available in Mac OS X v10.0 and later.

`kEventMouseDragged`
The mouse moved, and a button was down.

Available in Mac OS X v10.0 and later.

`kEventMouseEntered`
The mouse entered a tracking region. Used with mouse tracking regions. See `CreateMouseTrackingRegion` (page 208) for more information.

Available in Mac OS X v10.2 and later.

`kEventMouseExited`
The mouse left a tracking region. Used with mouse tracking regions. See `CreateMouseTrackingRegion` (page 208) for more information.

Available in Mac OS X v10.2 and later.

`kEventMouseWheelMoved`
The mouse wheel moved. (Mac OS X only)

Available in Mac OS X v10.0 and later.

Discussion
Table 8 shows the parameters related to mouse events.

Table 8 Parameter names and types for mouse event kinds

Event kind	Parameter name	Parameter type
`kEventMouseDown`	`kEventParamMouseLocation`	`typeQDPoint`
	`kEventParamKeyModifiers`	`typeUInt32`
	`kEventParamMouseButton`	`typeMouseButton`
	`kEventParamClickCount`	`typeUInt32`
	`kEventParamWindowRef`	`typeWindowRef`
	`kEventParamWindowMouseLocation`	`typeHIPoint`
	`kEventParamMouseChord`	`typeUInt32`
	`kEventParamTabletEventType`	`typeUInt32`
	`kEventParamTabletPointRec`	`typeTabletPointRec`
	`kEventParamTabletProximityRec`	`typeTabletProximityRec`
`kEventMouseUp`	`kEventParamMouseLocation`	`typeQDPoint`
	`kEventParamKeyModifiers`	`typeUInt32`
	`kEventParamMouseButton`	`typeMouseButton`
	`kEventParamWindowRef`	`typeWindowRef`
	`kEventParamClickCount`	`typeUInt32`
	`kEventParamMouseChord`	`typeUInt32`
	`kEventParamTabletEventType`	`typeUInt32`

	kEventParamTabletPointRec	typeTabletPointRec
	kEventParamTabletProximityRec	typeTabletProximityRec
kEventMouseMoved	kEventParamMouseLocation	typeQDPoint
	kEventParamKeyModifiers	typeUInt32
	kEventParamMouseDelta	typeHIPoint
	kEventParamWindowRef	typeWindowRef
	kEventParamWindowMouseLocation	typeHIPoint
	kEventParamTabletEventType	typeUInt32
	kEventParamTabletPointRec	typeTabletPointRec
	kEventParamTabletProximityRec	typeTabletProximityRec
kEventMouseDragged	kEventParamMouseLocation	typeQDPoint
	kEventParamKeyModifiers	typeUInt32
	kEventParamMouseButton	typeMouseButton
	kEventParamMouseChord	typeUInt32
	kEventParamMouseDelta	typeHIPoint
	kEventParamRef	typeWindowRef
	kEventParamWindowMouseLocation	typeHIPoint
	kEventParamTabletEventType	typeUInt32
	kEventParamTabletPointRec	typeTabletPointRec
	kEventParamTabletProximityRec	typeTabletProximityRec
kEventMouseEntered	kEventMouseTrackingRef	typeMouseTrackingRef
	kEventParamMouseLocation	typeQDPoint
	kEventParamKeyModifiers	typeUInt32
	kEventParamWindowRef	typeWindowRef
	kEventParamWindowMouseLocation	typeHIPoint
kEventMouseExited	kEventMouseTrackingRef	typeMouseTrackingRef
	kEventParamMouseLocation	typeQDPoint
	kEventParamKeyModifiers	typeUInt32

	kEventParamWindowRef	typeWindowRef
	kEventParamWindowMouseLocation	typeHIPoint
kEventMouse-WheelMoved	kEventParamMouseLocation	kEventParamWindowMouse Location
	kEventParamKeyModifiers	kEventParamTabletEvent Type
	kEventParamWindowRef	typeWindowRef
	kEventParamWindowMouseLocation	typeHIPoint
	kEventParamMouseWheelAxis	kEventParamTabletPointRec
	kEventParamMouseWheelDelta	kEventParamTablet ProximityRec

Mouse Button Constants

Define mouse button constants.

```
typedef UInt16 EventMouseButton;
enum {
    kEventMouseButtonPrimary = 1,
    kEventMouseButtonSecondary = 2,
    kEventMouseButtonTertiary = 3
};
```

Constants

`kEventMouseButtonPrimary`
The primary mouse button (default for one-button mice, typically the left button for multi-button mice).

Available in Mac OS X v10.0 and later.

`kEventMouseButtonSecondary`
The “right-click” mouse button.

Available in Mac OS X v10.0 and later.

`kEventMouseButtonTertiary`
The tertiary mouse button.

Available in Mac OS X v10.0 and later.

Mouse Wheel Constants

Define mouse scroll-wheel-axis constants.

```
typedef UInt16 EventMouseWheelAxis;
enum {
    kEventMouseWheelAxisX = 0,
    kEventMouseWheelAxisY = 1
};
```

Constants

`kEventMouseWheelAxisX`
The x-axis (left-right movement).

Available in Mac OS X v10.0 and later.

`kEventMouseWheelAxisY`
The y-axis (up-down movement).

Available in Mac OS X v10.0 and later.

Mouse Tracking Region Options

Define constants used by the `CreateMouseTrackingRegion` function.

```
typedef UInt32 MouseTrackingOptions;
enum {
    kMouseTrackingOptionsLocalClip = 0,
    kMouseTrackingOptionsGlobalClip = 1,
    kMouseTrackingOptionsStandard = kMouseTrackingOptionsLocalClip
};
```

Constants

`kMouseTrackingOptionsLocalClip`
The region passed to `CreateMouseTrackingRegion` (page 208) is defined in local coordinates, and that the region is clipped to the owning window's content region.

Available in Mac OS X v10.2 and later.

`kMouseTrackingOptionsGlobalClip`
The region passed to `CreateMouseTrackingRegion` (page 208) is defined in global coordinates and that the region is clipped to the owning window's structure region.

Available in Mac OS X v10.2 and later.

`kMouseTrackingOptionsStandard`
Same as `kMouseTrackingOptionsLocalClip`.

Available in Mac OS X v10.2 and later.

Alternate Mouse Tracking Result Constants

Define constants for alternate mouse tracking results.

```
enum {
    kMouseTrackingMousePressed = kMouseTrackingMouseDown,
    kMouseTrackingMouseReleased = kMouseTrackingMouseUp
};
```

Constants

`kMouseTrackingMousePressed`
The user pressed any mouse button.

Available in Mac OS X v10.0 and later.

`kMouseTrackingMouseReleased`
The user released the mouse button.

Available in Mac OS X v10.0 and later.

Mouse Event Parameters

Define constants for parameters to mouse events.

```
enum {
    kEventParamMouseLocation = 'mloc',
    kEventParamWindowMouseLocation = 'wmou',
    kEventParamMouseButton = 'mbtn',
    kEventParamClickCount = 'ccnt',
    kEventParamMouseWheelAxis = 'mwax',
    kEventParamMouseWheelDelta = 'mwdl',
    kEventParamMouseDelta = 'mdta',
    kEventParamMouseChord = 'chor',
    kEventParamTabletEventType = 'tblt',
    kEventParamMouseTrackingRef = 'mtrf',
    typeMouseButton = 'mbtn',
    typeMouseWheelAxis = 'mwax',
    typeMouseTrackingRef = 'mtrf'
};
```

Constants

`kEventParamMouseLocation`
typeQDPoint

Available in Mac OS X v10.0 and later.

`kEventParamMouseButton`
typeMouseButton

Available in Mac OS X v10.0 and later.

`kEventParamClickCount`
typeUInt32

Available in Mac OS X v10.0 and later.

`kEventParamMouseWheelAxis`
typeMouseWheelAxis

Available in Mac OS X v10.0 and later.

`kEventParamMouseWheelDelta`
typeSInt32

Available in Mac OS X v10.0 and later.

`kEventParamMouseDelta`
typeQDPoint

Available in Mac OS X v10.0 and later.

`kEventParamMouseChord`
typeUInt32

Available in Mac OS X v10.1 and later.

`typeMouseButton`
EventMouseButton

Available in Mac OS X v10.0 and later.

`typeMouseWheelAxis`
EventMouseWheelAxis

Available in Mac OS X v10.0 and later.

Mouse Tracking Option Constant

Define options for the `TrackMouseLocationWithOptions` function.

```
enum {
    kTrackMouseLocationOptionDontConsumeMouseUp = (1 << 0)
};
```

Constants

`kTrackMouseLocationOptionDontConsumeMouseUp`
Leave mouse-up events in the event queue (the default is to pull them.)

Available in Mac OS X v10.0 and later.

Discussion

This constant can be passed to `TrackMouseLocationWithOptions` (page 71) in the `inOptions` parameter.

Mouse Tracking Constants

Define constants for mouse tracking.

```
typedef UInt16 MouseTrackingResult;
enum {
    kMouseTrackingMouseDown = 1,
    kMouseTrackingMouseUp = 2,
    kMouseTrackingMouseExited = 3,
    kMouseTrackingMouseEntered = 4,
    kMouseTrackingMouseDragged = 5,
    kMouseTrackingKeyModifiersChanged = 6,
    kMouseTrackingUserCancelled = 7,
    kMouseTrackingTimedOut = 8,
    kMouseTrackingMouseMoved = 9
};
```

Constants

`kMouseTrackingMouseDown`
The user pressed any mouse button.

Available in Mac OS X v10.1 and later.

`kMouseTrackingMouseUp`
The user released the mouse button.

Available in Mac OS X v10.1 and later.

`kMouseTrackingMouseExited`
The mouse exited the specified region.

Available in Mac OS X v10.0 and later.

`kMouseTrackingMouseEntered`
The mouse entered the specified region.

Available in Mac OS X v10.0 and later.

`kMouseTrackingMouseDragged`
The mouse moved while the mouse button was down.

Available in Mac OS X v10.1 and later.

`kMouseTrackingKeyModifiersChanged`
One or more keyboard modifiers (option, control, and so on) for the mouse changed.

Available in Mac OS X v10.0 and later.

`kMouseTrackingMouseMoved`
Prior to Mac OS X v10.2, this constant was equivalent to `kMouseTrackingMouseDragged`. In Mac OS X v10.2 and later, `kMouseTrackingMouseMoved` indicates that the mouse moved while the mouse button was up.

Available in Mac OS X v10.0 and later.

Discussion

These constants are returned by `TrackMouseLocation` (page 70) and `TrackMouseRegion` (page 72), which are designed as replacements to calls such as `StillDown` and `WaitMouseUp`. The advantage over those routines is that `TrackMouseLocation` and `TrackMouseRegion` block if the user is not moving the mouse, whereas mouse tracking loops based on `StillDown` and `WaitMouseUp` spin, chewing up valuable CPU time that could be better spent elsewhere. It is highly recommended that any tracking loops in your application stop using `StillDown` and `WaitMouseUp` and start using `TrackMouseLocation` or `TrackMouseRegion`. See the notes on those functions for more information.

Mouse Tracking Selectors

Define a constant for "sticky" mode used by the `HIMouseTrackingGetParameters` function.

```
enum {
    kMouseParamsSticky = 'stic'
};
```

Constants

`kMouseParamsSticky`
Requests the time and distance for determining "sticky" mouse tracking. When the mouse is clicked on a menu title, the toolbox enters a sticky mouse-tracking mode that varies according to the time and distance between the mouse-down event and the mouse-up event. In this mode, the menu is tracked even though the mouse has already been released.

Available in Mac OS X v10.3 and later.

Services Manager Constants

Services Manager Events

Define constants related to Services Manager events.

```
enum {
    kEventServiceCopy = 1,
    kEventServicePaste = 2,
    kEventServiceGetTypes = 3,
    kEventServicePerform = 4
};
```

Constants

`kEventServiceCopy`

The user wants to invoke a service that requires your application to provide data. Your application must update the scrap reference in the `kEventParamScrapRef` parameter to indicate the appropriate data from the current selection or user focus. See the Discussion section for additional information about this parameter for Mac OS X v10.3 and later.(Available in Mac OS X v10.1 and later.)

Available in Mac OS X v10.1 and later.

`kEventServicePaste`

The user has invoked a service that requires your application to receive data. Your application must update the current user focus with the data provided by the `kEventParamScrapRef` parameter. See the Discussion section for additional information about this parameter for Mac OS X v10.3 and later.

(Available in Mac OS X v10.1 and later.)

Available in Mac OS X v10.1 and later.

`kEventServiceGetTypes`

Sent when the Services Manager needs to know what types of data it can cut-and-paste into the scrap. The Services Manager uses this information to update the Services menu, indicating which services are available for the current selection. This event passes two `CFMutableArray` references to you in the `kEventParamServiceCopyTypes` and `kEventParamServicePasteTypes` parameters. You should fill out these arrays with Core Foundation strings indicating which types your application can copy and paste. Note that you can use the `CreateTypeStringWithOSType` (page 25) to create a `CFStringRef` from an `OSType`.(Available in Mac OS X v10.1 and later.)

Available in Mac OS X v10.1 and later.

`kEventServicePerform`

Sent when your application must perform a service. The `kEventParamScrapRef` parameter holds the scrap information, and the `kEventParamServiceMessageName` parameter contains a Core Foundation string indicating what service was requested. Only applications that can provide services receive this event. See the Discussion section for additional information about this parameter for Mac OS X v10.3 and later.(Available in Mac OS X v10.1 and later.)

Available in Mac OS X v10.1 and later.

Discussion
In Mac OS X 10.3 and later, the `kEventServiceCopy`, `kEventServicePaste`, and `kEventServicePerform` events include a PasteboardRef and a ScrapRef. A handler for this event should provide its data using either Pasteboard or Scrap Manager APIs, and the corresponding pasteboard or scrap reference, depending on which is more convenient or appropriate. Data only needs to be placed on one of the pasteboard or scrap; it does not need to be placed on both. Data written to the pasteboard is also be available on the scrap, and vice versa.

Table 9 Parameter names and types for Service class events

Event kind	Parameter name	Parameter type
`kEventServiceCopy`	`kEventParamPasteboardRef` (Mac OS X v10.3 and later.)	`typePasteboardRef`
	`kEventParamScrapRef` (Mac OS X v10.1 and later.)	`typeScrapRef`
`kEventServicePaste`	`kEventParamPasteboardRef` (Mac OS X v10.3 and later.)	`typePasteboardRef`
	`kEventParamScrapRef` (Mac OS X v10.1 and later.)	`typeScrapRef`
`kEventService-GetTypes`	`kEventParamServiceCopyTypes`	`typeCFMutableArrayRef`
	`kEventParamServicePasteTypes`	`typeCFMutableArrayRef`
`kEventServicePerform`	`kEventParamPasteboardRef` (Mac OS X v10.3 and later.)	`typePasteboardRef`
	`kEventParamScrapRef` (Mac OS X v10.1 and later.)	`typeScrapRef`
	`kEventParamServiceMessageName`	`typeCFStringRef`
	`kEventParamServiceUserData`	`typeCFStringRef`

Services Manager Event Parameters

Define constants for parameters to Service Manager events.

```
enum {
    kEventParamScrapRef = 'scrp',
    kEventParamServiceCopyTypes = 'svsd',
    kEventParamServicePasteTypes = 'svpt',
    kEventParamServiceMessageName = 'svmg',
    kEventParamServiceUserData = 'svud',
    typeScrapRef = 'scrp',
    typeCFMutableArrayRef = 'cfma'
};
```

Constants

`kEventParamScrapRef`
When provided as a parameter to the `kEventServicePaste` event, the current selection should be replaced by data from this scrap. When provided as a parameter to `kEventServicePerform`, the scrap that should be used to send and receive data from the requester. When provided as a parameter to the `kEventServiceCopy` event, data from the current selection should be placed into this scrap. (Available in Mac OS X v10.1 and later.)

Available in Mac OS X v10.1 and later.

`kEventParamServiceCopyTypes`
When provided as a parameter to the `kEventServiceGetTypes` event, add CFString references to this array to report the types that can be pasted from the current selection. These strings will be released automatically after the event is handled. (Available in Mac OS X v10.1 and later.)

Available in Mac OS X v10.1 and later.

`kEventParamServicePasteTypes`
When provided as a parameter to the `kEventServiceGetTypes` event, add CFString references to this array to report the types that can be copied from the current selection. These strings will be released automatically after the event is handled. (Available in Mac OS X v10.1 and later.)

Available in Mac OS X v10.1 and later.

`kEventParamServiceMessageName`
When provided as a parameter to the `kEventServicePerform` event, contains the name of the advertised service that was invoked. (Available in Mac OS X v10.1 and later.)

Available in Mac OS X v10.1 and later.

`kEventParamServiceUserData`
When provided as a parameter to the `kEventServicePerform` event, contains extra data provided by the requestor. (Available in Mac OS X v10.1 and later.)

Available in Mac OS X v10.1 and later.

Tablet Event Constants

Tablet Events

Define constants for events related to drawing tablets.

```
enum {
    kEventTabletPoint = 1,
    kEventTabletProximity = 2,
    kEventTabletPointer = 1
};
```

Constants

`kEventTabletPoint`
Indicates that the pen has moved on a tablet. (Mac OS X only)

Available in Mac OS X v10.1 and later.

`kEventTabletProximity`
Indicates that the pen has entered the proximity region of the tablet. (Mac OS X only)

Available in Mac OS X v10.0 and later.

`kEventTabletPointer`
Same as `kEventTabletPoint`. This deprecated constant is here for compatibility only.

Available in Mac OS X v10.0 and later.

Tablet Event Parameters

Define constants for parameters to table events.

```
enum {
    kEventParamTabletPointRec = 'tbrc',
    kEventParamTabletProximityRec = 'tbpx',
    typeTabletPointRec = 'tbrc',
    typeTabletProximityRec = 'tbpx',
    kEventParamTabletPointerRec = 'tbrc',
    typeTabletPointerRec = 'tbrc'
};
```

Constants

`kEventParamTabletPointRec`
typeTabletPointRec

Available in Mac OS X v10.1 and later.

`kEventParamTabletProximityRec`
typeTabletProximityRec

Available in Mac OS X v10.0 and later.

`typeTabletPointRec`
kEventParamTabletPointRec

Available in Mac OS X v10.1 and later.

`typeTabletProximityRec`
kEventParamTabletProximityRec

Available in Mac OS X v10.0 and later.

`kEventParamTabletPointerRec`
typeTabletPointerRec

Available in Mac OS X v10.0 and later.

`typeTabletPointerRec`
kEventParamTabletPointerRec

Available in Mac OS X v10.0 and later.

Text Input Events

Text Input Event Constants

Define constants related to events from `kEventClassTextInput`.

```
enum {
    kEventTextInputUpdateActiveInputArea = 1,
    kEventTextInputUnicodeForKeyEvent = 2,
    kEventTextInputOffsetToPos = 3,
    kEventTextInputPosToOffset = 4,
    kEventTextInputShowHideBottomWindow = 5,
    kEventTextInputGetSelectedText = 6,
    kEventTextInputUnicodeText = 7,
    kEventTextInputFilterText = 14
};
```

Constants

`kEventTextInputUpdateActiveInputArea`

Tells the application text engine to initiate or terminate or manage the content of inline input session.

Available in Mac OS X v10.0 and later.

`kEventTextInputUnicodeForKeyEvent`

Provides Unicode text resulting from a key event (in which TSM originates the event) or from a `kEventTextInputUnicodeText` event produced by an input method, such as the Character Palette class input method, or a Handwriting input method. A client need not be fully TSM-aware to process or receive this event, which has become the standard way of getting Unicode text from key events. You can also get Mac encoding characters from the raw keyboard event contained in this event. If no `UnicodeForKeyEvent` handler is installed, and no `kUnicodeNotFromInputMethod` AppleEvent handler is installed (or the application has not created a Unicode TSMDocument), the Mac encoding charCodes (if these can be converted from the Unicodes) are provided to `WaitNextEvent`. This event is generated automatically by TSM when a `kEventRawKeyDown` event is sent to the application event target. The typical keyboard event flow begins with a `kEventRawKeyDown` event posted to the event queue. This event is dequeued during `WaitNextEvent` or `RunApplicationEventLoop`, and sent to the event dispatcher target. If the key down event reaches the application target, it is handled by TSM, which generates a `kEventTextInputUnicodeForKeyEvent` event and sends it to the event dispatcher target. The event dispatcher resends the event to the user focus target, which sends it to the focused window.

Available in Mac OS X v10.0 and later.

`kEventTextInputOffsetToPos`

Convert from inline session text offset to global QD Point. This event is typically produced by an input method so that it can best position a palette “near” the text being operated on by the user.

Available in Mac OS X v10.0 and later.

`kEventTextInputPosToOffset`
Convert from global QD point to inline session text offset. This event is typically produced by an input method to perform proper cursor management as the cursor moves over various subranges, or clauses of text (or the boundaries between these) in the inline input session.

Available in Mac OS X v10.0 and later.

`kEventTextInputShowHideBottomWindow`
Show/Hide the bottom line input window. This event is produced by Input Methods to control the Text Services Manager bottom-line input window, and is not normally handled by an application.

Available in Mac OS X v10.0 and later.

`kEventTextInputGetSelectedText`
Get the selected text (or the character before or after the insertion point, based on the value of the `leadingEdge` parameter) from the application's text engine.)

Available in Mac OS X v10.0 and later.

`kEventTextInputUnicodeText`
Produced only by input methods or other text services and is delivered to the Text Services Manager by `SendTextInputEvent`. The Text Services Manager does not dispatch this event to the user focus, so application handlers should not install handlers for this event. Instead, the Text Services Manager chains this event into any active keyboard input method in order to prevent interference with existing inline input sessions. The keyboard input method can either insert the text into the inline session or confirm its session and return the `UnicodeText` event to the Text Services Manager unhandled, in which case the Text Services Manager converts the event into a `UnicodeForKey` event (converting the Unicodes to Mac character codes and synthesizing information where needed) and finally dispatch the resulting event to the user focus as usual. (Available in Mac OS X v10.2 and later.)

Available in Mac OS X v10.2 and later.

`kEventTextInputFilterText`
Sent before any final-form text is sent to the user focus. Final form text includes text produced by a keyboard layout, Ink input method, Character palette or any other Text Services Manager text service, and any text being "confirmed" (or committed) from an inline input session. In the case of text confirmed from an inline input session, the Text Services Manager takes the resulting text buffer filtered by the event handler and adjusts all parameters in the `UpdateActiveInputArea` event produced by the input method. The text filtering action is thus transparent to both the application's `UpdateActiveInputArea` handler and the input method confirming the text.

Available in Mac OS X v10.4 and later.

Discussion
The following text input events reimplement the Apple events defined in the *Text Services Manager Reference*, and provide the benefits of Carbon event targeting, dispatching and propagation to applications that have formerly handled the Text Services Manager suite of Apple events. You can install text input handlers on controls, windows, or the application event target (which is equivalent to Apple-event-based handling). In all cases, if a given text input handler is not installed, the Text Services Manager converts the event into an AppleEvent and redispatches it via `AESend` to the current process, making adoption as gradual as is desired.

Table 10 shows the parameters related to text input events.

Table 10 Required parameter names and types for text input event kinds

Event kind	Parameter name	Parameter type
kEventTextInput-UpdateActive Input-Area	kEventParamTextInputSendComponent Instance	typeComponentInstance
	kEventParamTextInputSendRefCon	typeLongInteger
	kEventParamTextInputSendSLRec	typeIntlWritingCode
	kEventParamTextInputSendFixLen	typeLongInteger
	kEventParamTextInputSendText	typeUnicodeText for a Unicode document; typeChar otherwise
kEventTextInput-UnicodeFor KeyEvent	kEventParamTextInputSendComponent Instance	typeComponentInstance
	kEventParamTextInputSendRefCon	typeLongInteger
	kEventParamTextInputSendSLRec	typeIntlWritingCode
	kEventParamTextInputSendText	typeUnicodeText
	kEventParamTextInputSendKeyboard Event	typeEventRef
kEventTextInput-OffsetToPos	kEventParamTextInputSendComponent Instance	typeComponentInstance
	kEventParamTextInputSendRefCon	typeLongInteger
	kEventParamTextInputSendText Offset	typeLongInteger
	kEventParamTextInputReplyPoint	typeQDPoint
	kEventParamTextInputSendSLRec (Optional)	typeIntlWritingCode
	kEventParamTextInputSendLeading Edge (Optional)	typeBoolean
	kEventParamTextInputSendReplySL Rec (Optional)	typeIntlWritingCode
	kEventParamTextInputSendReplyFont (Optional)	typeLongInteger
	kEventParamTextInputSendReplyFM Font (Optional)	typeUInt32

	kEventParamTextInputSendReply PointSize (Optional)	typeFixed
	kEventParamTextInputSendReplyLine Height (Optional)	typeShortInteger
	kEventParamTextInputSendReplyLine Ascent (Optional)	typeShortInteger
	kEventParamTextInputSendReplyText Angle (Optional)	typeFixed
kEventTextInput-PosToOffset	kEventParamTextInputSendComponent Instance	typeComponentInstance
	kEventParamTextInputSendRefCon	typeLongInteger
	kEventParamTextInputSendCurrent Point	typeQDPoint
	kEventParamTextInputReplyRegion Class	typeLongInteger
	kEventParamTextInputReplyText Offset	typeLongInteger
	kEventParamTextInputSendDragging Mode (Optional)	typeBoolean
	kEventParamTextInputReplyLeading Edge (Optional)	typeBoolean
	kEventParamTextInputSendReplySL Rec (Optional)	typeIntlWritingCode
kEventTextInputShow-Hide BottomWindow	kEventParamTextInputSendComponent Instance	typeComponentInstance
	kEventParamTextInputSendRefCon	typeLongInteger
	kEventParamTextInputSendShowHide (Optional)	typeBoolean
	kEventParamTextInputReplyShowHide (Optional)	typeBoolean
kEventTextInputGet-Selected Text	kEventParamTextInputSendComponent Instance	typeComponentInstance
	kEventParamTextInputSendRefCon	typeLongInteger
	kEventParamTextInputSendLeading Edge (Optional)	typeBoolean
	kEventParamTextInputSendText ServiceEncoding (Optional)	TypeUInt32

	kEventParamTextInputSendText ServiceMacEncoding (Optional)	TypeUInt32
	kEventParamTextInputReplyText (Optional)	typeUnicodeText or typeChar depending on the TSMDocument type.
	kEventParamTextInputSendReplySL Rec (Optional)	typeIntlWritingCode
	kEventParamTextInputGlyphInfo Array (Optional)	TypeGlyphInfoArray
kEventTextInput-UnicodeText	kEventParamTextInputSendComponent Instance	typeComponentInstance
	kEventParamTextInputSendSLRec	typeIntlWritingCode
	kEventParamTextInputSendText	typeUnicodeText
	kEventParamTextInputSendText ServiceEncoding (Optional)	TypeUInt32
	kEventParamTextInputSendText ServiceMacEncoding (Optional)	TypeUInt32
	kEventParamTextInputGlyphInfo Array (Optional)	TypeGlyphInfoArray
kEventTextInput-FilterText	kEventParamTextInputSendRefCon	typeLongInteger
	kEventParamTextInputSendText	typeUnicodeText
	kEventParamTextInputReplyText	typeUnicodeText

Deprecated Text Input Constants

Define deprecated text input events.

```
enum {
    kEventUpdateActiveInputArea = kEventTextInputUpdateActiveInputArea,
    kEventUnicodeForKeyEvent = kEventTextInputUnicodeForKeyEvent,
    kEventOffsetToPos = kEventTextInputOffsetToPos,
    kEventPosToOffset = kEventTextInputPosToOffset,
    kEventShowHideBottomWindow = kEventTextInputShowHideBottomWindow,
    kEventGetSelectedText = kEventTextInputGetSelectedText
};
```

Constants

`kEventUpdateActiveInputArea`

Equivalent to `kEventTextInputUpdateActiveInputArea`.

Available in Mac OS X v10.0 and later.

`kEventUnicodeForKeyEvent`
Equivalent to `kEventTextInputUnicodeForKeyEvent`.

Available in Mac OS X v10.0 and later.

`kEventOffsetToPos`
Equivalent to `kEventTextInputOffsetToPos`.

Available in Mac OS X v10.0 and later.

`kEventPosToOffset`
Equivalent to `kEventTextInputPosToOffset`.

Available in Mac OS X v10.0 and later.

`kEventShowHideBottomWindow`
Equivalent to `kEventTextInputShowHideBottomWindow`.

Available in Mac OS X v10.0 and later.

`kEventGetSelectedText`
Equivalent to `kEventTextInputGetSelectedText`.

Available in Mac OS X v10.0 and later.

Discussion
These constants are superseded by constants described in "Text Input Event Constants" (page 160) and are included for backwards compatibility.

Text Input Event Parameters

Define constants for parameters to text input events.

```
enum {
    kEventParamTextInputSendRefCon = 'tsrc',
    kEventParamTextInputSendComponentInstance = 'tsci',
    kEventParamTextInputSendSLRec = 'tssl',
    kEventParamTextInputReplySLRec = 'trsl',
    kEventParamTextInputSendText = 'tstx',
    kEventParamTextInputReplyText = 'trtx',
    kEventParamTextInputSendUpdateRng = 'tsup',
    kEventParamTextInputSendHiliteRng = 'tshi',
    kEventParamTextInputSendClauseRng = 'tscl',
    kEventParamTextInputSendPinRng = 'tspn',
    kEventParamTextInputSendFixLen = 'tsfx',
    kEventParamTextInputSendLeadingEdge = 'tsle',
    kEventParamTextInputReplyLeadingEdge = 'trle',
    kEventParamTextInputSendTextOffset = 'tsto',
    kEventParamTextInputReplyTextOffset = 'trto',
    kEventParamTextInputReplyRegionClass = 'trrg',
    kEventParamTextInputSendCurrentPoint = 'tscp',
    kEventParamTextInputSendDraggingMode = 'tsdm',
    kEventParamTextInputReplyPoint = 'trpt',
    kEventParamTextInputReplyFont = 'trft',
    kEventParamTextInputReplyFMFont = 'trfm',
    kEventParamTextInputReplyPointSize = 'trpz',
    kEventParamTextInputReplyLineHeight = 'trlh',
    kEventParamTextInputReplyLineAscent = 'trla',
```

```
    kEventParamTextInputReplyTextAngle = 'trta',
    kEventParamTextInputSendShowHide = 'tssh',
    kEventParamTextInputReplyShowHide = 'trsh',
    kEventParamTextInputSendKeyboardEvent = 'tske',
    kEventParamTextInputSendTextServiceEncoding = 'tsse',
    kEventParamTextInputSendTextServiceMacEncoding = 'tssm',
    kEventParamTextInputGlyphInfoArray = 'glph',
    kEventParamTextInputSendGlyphInfoArray =  kEventParamTextInputGlyphInfoArray,
    kEventParamTextInputReplyGlyphInfoArray = 'rgph',
    kEventParamTextInputSendReplaceRange = 'tsrp'
};
```

Constants

`kEventParamTextInputSendRefCon`
typeLongInteger

Available in Mac OS X v10.0 and later.

`kEventParamTextInputSendComponentInstance`
typeComponentInstance

Available in Mac OS X v10.0 and later.

`kEventParamTextInputSendSLRec`
typeIntlWritingCode

Available in Mac OS X v10.0 and later.

`kEventParamTextInputReplySLRec`
typeIntlWritingCode

Available in Mac OS X v10.0 and later.

`kEventParamTextInputSendText`
typeUnicodeText (if TSMDocument is Unicode), otherwise typeChar

Available in Mac OS X v10.0 and later.

`kEventParamTextInputReplyText`
typeUnicodeText (if TSMDocument is Unicode), otherwise typeChar

Available in Mac OS X v10.0 and later.

`kEventParamTextInputSendUpdateRng`
typeTextRangeArray

Available in Mac OS X v10.0 and later.

`kEventParamTextInputSendHiliteRng`
typeTextRangeArray

Available in Mac OS X v10.0 and later.

`kEventParamTextInputSendClauseRng`
typeOffsetArray

Available in Mac OS X v10.0 and later.

`kEventParamTextInputSendPinRng`
typeTextRange

Available in Mac OS X v10.0 and later.

`kEventParamTextInputSendFixLen`
typeLongInteger

Available in Mac OS X v10.0 and later.

`kEventParamTextInputSendLeadingEdge`
typeBoolean

Available in Mac OS X v10.0 and later.

`kEventParamTextInputReplyLeadingEdge`
typeBoolean

Available in Mac OS X v10.0 and later.

`kEventParamTextInputSendTextOffset`
typeLongInteger

Available in Mac OS X v10.0 and later.

`kEventParamTextInputReplyTextOffset`
typeLongInteger

Available in Mac OS X v10.0 and later.

`kEventParamTextInputReplyRegionClass`
typeLongInteger

Available in Mac OS X v10.0 and later.

`kEventParamTextInputSendCurrentPoint`
typeQDPoint

Available in Mac OS X v10.0 and later.

`kEventParamTextInputSendDraggingMode`
typeBoolean

Available in Mac OS X v10.0 and later.

`kEventParamTextInputReplyPoint`
typeQDPoint

Available in Mac OS X v10.0 and later.

`kEventParamTextInputReplyFont`
typeLongInteger

Available in Mac OS X v10.0 and later.

`kEventParamTextInputReplyFMFont`
typeUInt32

Available in Mac OS X v10.1 and later.

`kEventParamTextInputReplyPointSize`
typeFixed

Available in Mac OS X v10.0 and later.

`kEventParamTextInputReplyLineHeight`
typeShortInteger

Available in Mac OS X v10.0 and later.

`kEventParamTextInputReplyLineAscent`
typeShortInteger

Available in Mac OS X v10.0 and later.

`kEventParamTextInputReplyTextAngle`
typeFixed

Available in Mac OS X v10.0 and later.

`kEventParamTextInputSendShowHide`
typeBoolean

Available in Mac OS X v10.0 and later.

`kEventParamTextInputReplyShowHide`
typeBoolean

Available in Mac OS X v10.0 and later.

`kEventParamTextInputSendKeyboardEvent`
typeEventRef

Available in Mac OS X v10.0 and later.

`kEventParamTextInputSendTextServiceEncoding`
typeUInt32

Available in Mac OS X v10.0 and later.

`kEventParamTextInputSendTextServiceMacEncoding`
typeUInt32

Available in Mac OS X v10.0 and later.

`kEventParamTextInputReplyGlyphInfoArray`
typeGlyphInfoArray

Available in Mac OS X v10.3 and later.

`kEventParamTextInputSendReplaceRange`
typeCFRange

Available in Mac OS X v10.3 and later.

Text Service Manager Document Events

Text Service Manager Document Event Parameters

Define constants for Text Service Manager Document event parameters and types.

```
enum {
    kEventParamTSMDocAccessSendRefCon = kEventParamTSMSendRefCon,
    kEventParamTSMDocAccessSendComponentInstance =
kEventParamTSMSendComponentInstance,
    kEventParamTSMDocAccessCharacterCount = 'tdct',
    kEventParamTSMDocAccessReplyCharacterRange = 'tdrr',
    kEventParamTSMDocAccessReplyCharactersPtr = 'tdrp',
    kEventParamTSMDocAccessSendCharacterIndex = 'tdsi',
    kEventParamTSMDocAccessSendCharacterRange = 'tdsr',
    kEventParamTSMDocAccessSendCharactersPtr = 'tdsp',
    kEventParamTSMDocAccessRequestedCharacterAttributes = 'tdca',
    kEventParamTSMDocAccessReplyATSFont = 'tdaf',
    kEventParamTSMDocAccessReplyFontSize = 'tdrs',
    kEventParamTSMDocAccessEffectiveRange = 'tder',
    kEventParamTSMDocAccessReplyATSUGlyphSelector = 'tdrg',
    kEventParamTSMDocAccessLockCount = 'tdlc',
    kEventParamTSMDocAccessLineBounds = 'tdlb',
    typeATSFontRef = 'atsf',
    typeGlyphSelector = 'glfs'
};
```

Constants

`kEventParamTSMDocAccessSendRefCon`
typeLongInteger

Available in Mac OS X v10.3 and later.

`kEventParamTSMDocAccessSendComponentInstance`
typeComponentInstance

Available in Mac OS X v10.3 and later.

`kEventParamTSMDocAccessCharacterCount`
typeCFIndex

Available in Mac OS X v10.3 and later.

`kEventParamTSMDocAccessReplyCharacterRange`
typeCFRange

Available in Mac OS X v10.3 and later.

`kEventParamTSMDocAccessReplyCharactersPtr`
typePtr

Available in Mac OS X v10.3 and later.

`kEventParamTSMDocAccessSendCharacterIndex`
typeCFIndex

Available in Mac OS X v10.3 and later.

`kEventParamTSMDocAccessSendCharactersPtr`
typePtr

Available in Mac OS X v10.3 and later.

`kEventParamTSMDocAccessRequestedCharacterAttributes`
typeUInt32

Available in Mac OS X v10.3 and later.

`kEventParamTSMDocAccessReplyATSFont`
typeATSFontRef

Available in Mac OS X v10.3 and later.

`kEventParamTSMDocAccessReplyFontSize`
typeFloat

Available in Mac OS X v10.3 and later.

`kEventParamTSMDocAccessEffectiveRange`
typeRange

Available in Mac OS X v10.3 and later.

`kEventParamTSMDocAccessReplyATSUGlyphSelector`
typeGlyphSelector

Available in Mac OS X v10.3 and later.

`kEventParamTSMDocAccessLockCount`
typeCFIndex

Available in Mac OS X v10.3 and later.

`kEventParamTSMDocAccessLineBounds`
typeCFMutableArrayRef

Available in Mac OS X v10.4 and later.

`typeATSFontRef`
ATSFontRef

Available in Mac OS X v10.3 and later.

`typeGlyphSelector`
ATSUGlyphSelector

Available in Mac OS X v10.3 and later.

Timer Constants

Idle Timer Event Constants

Define constants used for timer events.

```
enum {
```

```
    kEventLoopIdleTimerStarted = 1,
    kEventLoopIdleTimerIdling = 2,
    kEventLoopIdleTimerStopped = 3
};
```

Constants

`kEventLoopIdleTimerStarted`
The idle period has just begun (and this is the first time your callback is being called for this idle period).

Available in Mac OS X v10.2 and later.

`kEventLoopIdleTimerIdling`
The idle period is continuing.

Available in Mac OS X v10.2 and later.

`kEventLoopIdleTimerStopped`
The idle period has just stopped (a user event occurred). Your callback should do any necessary cleanup of the idle process now that a user event has occurred.

Available in Mac OS X v10.2 and later.

Discussion

These constants are passed to your idle timer callback function. For more information, see the `InstallEventLoopIdleTimer` (page 44) function.

Toolbar Events

Toolbar Event Parameters

Define constants for parameters to toolbar events.

```
enum {
    kEventParamToolbar = 'tbar',
    kEventParamToolbarItem = 'tbit',
    kEventParamToolbarItemIdentifier = 'tbii',
    kEventParamToolbarItemConfigData = 'tbid',
    typeHIToolbarRef = 'tbar',
    typeHIToolbarItemRef = 'tbit'
};
```

Constants

`kEventParamToolbar`
typeHIToolbarRef

Available in Mac OS X v10.2 and later.

`kEventParamToolbarItem`
typeHIToolbarItemRef

Available in Mac OS X v10.2 and later.

`kEventParamToolbarItemIdentifier`
typeHIToolbarRef

Available in Mac OS X v10.2 and later.

`kEventParamToolbarItemConfigData`
typeCFStringRef

Available in Mac OS X v10.2 and later.

Discussion
For details about toolbar events and event parameters, see HIToolbar Reference in the User Experience section of the Carbon documentation.

Volume Events

Volume Event Constants

Define constants related to events from `kEventClassVolume`.

```
enum {
    kEventVolumeMounted = 1,
    kEventVolumeUnmounted = 2
};
```

Constants
`kEventVolumeMounted`
New volume (hard drive or removable media) mounted.

Available in Mac OS X v10.0 and later.

`kEventVolumeUnmounted`
Volume has been ejected or unmounted.

Available in Mac OS X v10.0 and later.

Volume Reference Constant

Define the type of a volume reference.

```
enum {
    typeFSVolumeRefNum = 'voln'
};
```

Constants
`typeFSVolumeRefNum`
An `FSVolumeRefNum` identifying the volume that was mounted or unmounted.

Available in Mac OS X v10.0 and later.

Window Events

Window Action Event Constants

Define constants related to events from `kEventClassWindow`.

```
enum {
    kEventWindowCollapse = 66,
    kEventWindowCollapseAll = 68,
    kEventWindowExpand = 69,
    kEventWindowExpandAll = 71,
    kEventWindowClose = 72,
    kEventWindowCloseAll = 74,
    kEventWindowZoom = 75,
    kEventWindowZoomAll = 77,
    kEventWindowContextualMenuSelect = 78,
    kEventWindowPathSelect = 79,
    kEventWindowGetIdealSize = 80,
    kEventWindowGetMinimumSize = 81,
    kEventWindowGetMaximumSize = 82,
    kEventWindowConstrain = 83,
    kEventWindowHandleContentClick = 85,
    kEventWindowTransitionStarted = 88,
    kEventWindowTransitionCompleted = 89,
    kEventWindowGetDockTileMenu = 90,
    kEventWindowGetDockTileMenu = 90,
    kEventWindowProxyBeginDrag = 128,
    kEventWindowProxyEndDrag = 129,
    kEventWindowToolbarSwitchMode = 150
};
```

Constants

`kEventWindowCollapse`

If the window is not collapsed, this event is sent by the standard window handler after it has received `kEventWindowClickCollapseRgn` and received `true` from a call to `TrackBox`. The default behavior is to call `CollapseWindow` and then send `kEventWindowCollapsed` if no error is received from `CollapseWindow`. (Available in Mac OS X and CarbonLib 1.1 and later.)

Available in Mac OS X v10.0 and later.

`kEventWindowCollapseAll`

Sent by the standard window handler (when the option key is down) after it has received `kEventWindowClickCollapseRgn` and then received true from a call to `TrackBox`. The default response is to send each window of the same class as the clicked window a `kEventWindowCollapse` event. (Available in Mac OS X and CarbonLib 1.1 and later.)

Available in Mac OS X v10.0 and later.

`kEventWindowExpand`

If the window is collapsed, this event is sent by the standard window handler after it has received `kEventWindowClickCollapseRgn` and received true from a call to `TrackBox`. The default response is to call `CollapseWindow`, then send `kEventWindowExpanded`. Note that you will not receive this event before a window is expanded from the dock, since minimized windows in the dock don't uses collapse boxes to unminimize. (Available in Mac OS X and CarbonLib 1.1 and later.)

Available in Mac OS X v10.0 and later.

`kEventWindowExpandAll`

Sent by the standard window handler (when the option key is down) after it has received `kEventWindowClickCollapseRgn` and then received true from a call to `TrackBox`. The default response is to send each window of the same class as the clicked window a `kEventWindowExpand` event. (Available in Mac OS X and CarbonLib 1.1 and later.)

Available in Mac OS X v10.0 and later.

`kEventWindowClose`

Sent by the standard window handler after it has received `kEventWindowClickCloseRgn` and successfully called `TrackBox`. Your application might intercept this event to check if the document is dirty, and display a Save/Don'tSave/Cancel alert.

The default response is to call the Window Manager function `DisposeWindow`. (Available in Mac OS X and CarbonLib 1.1 and later.)

Available in Mac OS X v10.0 and later.

`kEventWindowCloseAll`

Sent by the standard window handler (when the option key is down) after it has received `kEventWindowClickCloseRgn` and received true from a call to `TrackGoAway`. The standard window handler's response is to send each window with the same class as the clicked window a `kEventWindowClose` event. (Available in Mac OS X and CarbonLib 1.1 and later.)

Available in Mac OS X v10.0 and later.

`kEventWindowZoom`

Sent by the standard window handler upon receiving `kEventWindowClickZoomRgn` and then receiving true from a call to `TrackBox`. The default behavior is to zoom the window using `ZoomWindowIdeal` then, if successful, send a `kEventWindowZoomed` event. (Available in Mac OS X and CarbonLib 1.1 and later.)

Available in Mac OS X v10.0 and later.

`kEventWindowZoomAll`

Sent by the standard window handler (when the option key is down) after it has received `kEventObjectClickZoomRgn` and received true from a call to `TrackBox`. The standard window handler's response is to send each window with the same class as the clicked window a `kEventObjectZoom` event and then to reposition all zoomed windows using the `kWindowCascadeOnParentWindowScreen` positioning method. For more details, see the *Window Manager Reference* for more details. (Available in Mac OS X and CarbonLib 1.1 and later.)

Available in Mac OS X v10.0 and later.

`kEventWindowContextualMenuSelect`
Sent when either the right mouse button is pressed, or the control key is held down and the left mouse button is pressed, or the left mouse button is held down for more than 1/4th of a second (and nothing else is handling the generated mouse tracking events). The standard window handler ignores this event. Note that this event supports `kEventParamMouseLocation` and other parameters associated with the `kEventMouseDown` event. (Available in Mac OS X and CarbonLib 1.1 and later.)

Available in Mac OS X v10.0 and later.

`kEventWindowPathSelect`
Sent when the Window Manager function `IsWindowPathSelectClick` would return true. The standard window handler sends this event while handling `kEventWindowClickDragRgn` if the click occurs in the proxy icon. Set the menu reference parameter (`kEventParamMenuRef`) in the event if you wish to customize the menu passed to the Window Manager function `WindowPathSelect`. (Available in Mac OS X and CarbonLib 1.1 and later.)

Available in Mac OS X v10.0 and later.

`kEventWindowGetIdealSize`
Sent by the standard window handler to determine the standard state for zooming. The standard window handler ignores this event. (Available in Mac OS X and CarbonLib 1.1 and later.)

Available in Mac OS X v10.0 and later.

`kEventWindowGetMinimumSize`
Sent by the standard window handler to determine the minimum size of the window (used during window resizing). (Available in Mac OS X and CarbonLib 1.1 and later.)

In Mac OS X v10.2 and later, the default behavior is to call the Window Manager function `GetWindowResizeLimits` and return the size obtained in the `kEventParamDimensions` parameter. There is no default behavior before Mac OS X v10.2.

Available in Mac OS X v10.0 and later.

`kEventWindowGetMaximumSize`
Sent by the standard window handler to determine the maximum size of the window (used during window resizing). (Available in Mac OS X and CarbonLib 1.1 and later.) On CarbonLib 1.6 and Mac OS X v10.2 and later, this event is also sent by the Window Manager functions `ResizeWindow` and `GrowWindow` if the `sizeContraints` parameter was set to `NULL`.

In Mac OS X v10.2 and later, the default behavior is to call the Window Manager function `GetWindowResizeLimits` and return the size obtained in the `kEventParamDimensions` parameter. There is no default behavior before Mac OS X v10.2.

Available in Mac OS X v10.0 and later.

`kEventWindowConstrain`

Sent by the standard window handler to warn of a change in the available window positioning bounds on the window (for example, due to a change in screen resolution or Dock size). (Available in Mac OS X and CarbonLib 1.5 and later.)

In Mac OS v10.0.x the default behavior is to call the Window Manager function `ConstrainWindowToScreen` on the window with the `kWindowConstrainMoveRegardlessOfFit` attribute set and a window region code of `kWindowDragRgn`. The window is constrained to the bounds returned by the Window Manager function `GetAvailableWindowPositioningBounds` for that display device.

In Mac OS X v10.1 and later the default behavior is to call `ConstrainWindowToScreen` on the window with the `kWindowConstrainMoveRegardlessOfFit` and `kWindowConstrainAllowPartial` attributes set, and a window region code of `kWindowDragRgn`. Instead of accepting the normal device bounds, you can also modify the `kEventParamAvailableBounds` for this event, and the default handler constrains the window to those bounds.

In Mac OS X v10.2 and later, you can set the following optional parameters:

- `kEventParamAttributes`: You can set the constraint attributes to pass to `ConstrainWindowToScreen` by specifying them in this parameter.
- `kEventParamWindowRegionCode`: If you set this parameter (which must be of type `WindowRegionCode`), the standard window handler passes this value to `ConstrainWindowToScreen` instead of `kWindowDragRgn`.

In addition, the following optional parameters may exist in Mac OS X v10.2 and later:

- `kEventParamRgnHandle`: Contains the gray region before a configuration change in the available graphics devices (screens). This parameter exists only if the constrain event occurred because the user changed the screen configuration. You can call the Window Manager function `GetGrayRgn` to obtain the current gray region.
- `kEventParamCurrentDockRect`: Holds the current bounds of the dock. This parameter and `kEventParamPreviousDockRect` exist only if the constrain event resulted from a change in the Dock size or position.
- `kEventParamPreviousDockRect`: Holds the previous bounds of the dock.

Available in Mac OS X v10.0 and later.

`kEventWindowHandleContentClick`

Sent by the standard window handler in response to `kEventWindowClickContentRgn` when a mouse click is in the content region but is not a contextual menu invocation or a click on a control. Note that this event supports `kEventParamMouseLocation` and other parameters associated with the `kEventMouseDown` event.

The standard handler ignores this event.

(Available in Mac OS X and CarbonLib 1.3.1 and later.)

Available in Mac OS X v10.0 and later.

`kEventWindowTransitionStarted`
Sent to all handlers registered for it. It is sent by the `TransitionWindow`, `TransitionWindowAndParent`, and `TransitionWindowWithOptions` APIs just before the first frame of the transition animation.

(Available in Mac OS X v10.3 and later.)

Available in Mac OS X v10.3 and later.

`kEventWindowTransitionCompleted`
Sent to all handlers registered for it. It is sent by the `TransitionWindow`, `TransitionWindowAndParent`, and `TransitionWindowWithOptions` APIs just after the last frame of the transition animation.

(Available in Mac OS X v10.3 and later.)

Available in Mac OS X v10.3 and later.

`kEventWindowGetDockTileMenu`
Sent when a dock tile wants to display a menu. The sender of this event releases the menu after the Dock has displayed it, so if you want to keep the menu, you must call `RetainMenu` on it before returning from the event handler.

If you do not handle this event, the default behavior is to call the Window Manager function `GetWindowDockTileMenu` and return the menu obtained in the `kEventParamMenuRef` parameter. If no menu is specified, the default handler returns `eventNotHandledErr`.

Note that in most cases it is simpler to call the `SetWindowDockTileMenu` function directly rather than register for this event.

(Available in Mac OS X v10.2 and later.)

Available in Mac OS X v10.2 and later.

`kEventWindowProxyBeginDrag`
Sent before a proxy icon drag; you can attach data to the `DragRef` in the event. The standard handler ignores this event. (Available in Mac OS X v10.0 and later, and CarbonLib 1.1 and later.)

Available in Mac OS X v10.0 and later.

`kEventWindowProxyEndDrag`
Sent after the proxy icon drag is complete, whether successful or not. The standard handler ignores this event. (Available in Mac OS X v10.0 and later, and CarbonLib 1.1 and later.)

Available in Mac OS X v10.0 and later.

`kEventWindowToolbarSwitchMode`

The toolbar button (that is, the oblong button used to show and hide the toolbar) was successfully clicked. The standard window handler sends this event when receiving a `true` return value from `TrackBox` during the handling of the `kEventWindowClickToobarButtonRgn` event. Note, however, that you do not have to have the standard window handler installed to receive this event; any window that has a toolbar receives this event when its toolbar button is successfully clicked. Note that if you handle this event, your application is responsible for keeping track of the toolbar's mode (visible or hidden).

The default response is to toggle the toolbar (that is, show it if it is hidden, and vice-versa) when the toolbar button is clicked. If the option key is held down during the click, all toolbars in the windows of the current process are toggled. If the command key is down during the click, the toolbar mode is cycled between icons and text, icons alone, and text alone. If both the command and option keys are held during the click, the system displays the toolbar configuration sheet.

(

Available in Mac OS X only.)

Discussion

These events indicate that certain changes have been made to a window. These events have greater semantic meaning than the low-level window click events and are usually preferred for overriding.

Table 11 shows the parameters related to window action events.

Table 11 Parameter names and types for window action event kinds

Event kind	Parameter name	Parameter type
`kEventWindowCollapse`	`kEventParamDirectObject`	`typeWindowRef`
`kEventWindowCollapsed`	`kEventParamDirectObject`	`typeWindowRef`
`kEventWindowCollapseAll`	`kEventParamDirectObject`	`typeWindowRef`
`kEventWindowExpand`	`kEventParamDirectObject`	`typeWindowRef`
`kEventWindowExpanded`	`kEventParamDirectObject`	`typeWindowRef`
`kEventWindowExpandAll`	`kEventParamDirectObject`	`typeWindowRef`
`kEventWindowClose`	`kEventParamDirectObject`	`typeWindowRef`
`kEventWindowClosed`	`kEventParamDirectObject`	`typeWindowRef`
`kEventWindowCloseAll`	`kEventParamDirectObject`	`typeWindowRef`
`kEventWindowZoom`	`kEventParamDirectObject`	`typeWindowRef`
`kEventWindowZoomed`	`kEventParamDirectObject`	`typeWindowRef`
`kEventWindowZoomAll`	`kEventParamDirectObject`	`typeWindowRef`
`kEventWindow-ContextualMenuSelect`	`kEventParamDirectObject`	`typeWindowRef`

	Other parameters from `kEventMouseDown`	
`kEventWindowPathSelect`	`kEventParamDirectObject`	`typeWindowRef`
	`kEventParamMenuRef`	`typeMenuRef`
`kEventWindowGetIdealSize`	`kEventParamDirectObject`	`typeWindowRef`
	`kEventParamDimensions`	`typeQDPoint`
`kEventWindowGetMinimumSize`	`kEventParamDirectObject`	`typeWindowRef`
	`kEventParamDimensions`	`typeQDPoint`
`kEventWindowGetMaximumSize`	`kEventParamDirectObject`	`typeWindowRef`
	`kEventParamDimensions`	`typeQDPoint`
`kEventWindowConstrain`	`kEventParamDirectObject`	`typeWindowRef`
	`kEventParamAvailableBounds`	`typeQDRectangle`
	`kEventParamAttributes`	`typeUInt32`
	`kEventParamWindowRegionCode`	`typeQDRgnHandle`
	`kEventParamPreviousDockRect`	`typeHIRect`
	`kEventParamCurrentDockRect`	`typeHIRect`
`kEventWindowHandle-WindowContent Click`	`kEventParamDirectObject`	`typeWindowRef`
	Other parameters from `kEventMouseDown`	
`kEventWindowProxyBeginDrag`	`kEventParamDirectObject`	`typeWindowRef`
`kEventWindowProxyEndDrag`	`kEventParamDirectObject`	`typeWindowRef`
`kEventWindowToolbar-SwitchMode`	`kEventParamDirectObject`	`typeWindowRef`

Window Activation Event Constants

Define constants related to events from `kEventClassWindow` that specify whether a window is activated or deactivated.

```
enum {
    kEventWindowActivated = 5,
    kEventWindowDeactivated = 6,
    kEventWindowHandleActivate = 91,
    kEventWindowHandleDeactivate = 92
    kEventWindowGetClickActivation = 7,
    kEventWindowGetClickModality = 8
};
```

Constants

`kEventWindowActivated`

The window is active now. Sent to any window that is activated, regardless of whether the window has the standard window handler installed.

In Mac OS X v10.3 and later, the standard window handler responds to this event by sending a `kEventWindowHandleActivate` event to the window. On earlier releases of Mac OS X and CarbonLib, the standard window handler calls `ActivateControl` on the window's root control.

Note that this event is sent directly to the window target. If no handler takes the event (that is, they all return `eventNotHandledErr`), then the Window Manager posts this event to the event queue. Doing so allows the Carbon Event Manager to convert the event into an old-style event record (`EventRecord`), to be returned from `WaitNextEvent`.

Available in Mac OS X v10.0 and later.

`kEventWindowDeactivated`

The window is inactive now. Sent to any window that is deactivated, regardless of whether the window has the standard window handler installed.

In Mac OS X v10.3 and later, the standard window handler responds to this event by sending a `kEventWindowHandleDeactivate` event to the window. On earlier releases of Mac OS X and CarbonLib, the standard window handler calls `DeactivateControl` on the window's root control.

Note that this event is sent directly to the window target. If no handler takes the event (that is, they all return `eventNotHandledErr`), then the Window Manager posts this event to the event queue. Doing so allows the Carbon Event Manager to convert the event into an old-style event record (`EventRecord`), to be returned from `WaitNextEvent`.

Available in Mac OS X v10.0 and later.

`kEventWindowHandleActivate`

The window has received a `kEventWindowActivated` event, and its contents should become active. This event is generated by the standard window handler in response to a `kEventWindowActivated` event. You can handle this event by activating the window's content appropriately. The standard window handler responds to this event by calling `ActivateControl` on the window's root control.

(Available in Mac OS X v10.3 and later.)

Available in Mac OS X v10.4 and later.

`kEventWindowHandleDeactivate`

The window has received a `kEventWindowDeactivated` event, and its contents should become inactive. This event is generated by the standard window handler in response to a `kEventWindowDeactivated` event. You can handle this event by deactivating the window's content appropriately. The standard window handler responds to this event by calling `DeactivateControl` on the window's root control.

(Available in Mac OS X v10.3 and later.)

Available in Mac OS X v10.4 and later.

`kEventWindowGetClickActivation`
Sent when a click occurs in a background window that has the standard window handler installed.

The default behavior is to bring the window forward and absorb the click (that is, the mouse click is not passed on to the window). In addition, the appropriate click activation part code is returned in the `kEventParamClickActivationResult` parameter. You can use this event to override this behavior and implement click-through.

Available in Mac OS X v10.0 and later.

`kEventWindowGetClickModality`
Sent to a window by the event dispatcher target before dispatching a mouse-down or mouse-up event to the clicked window. A handler for this event may examine application state to determine whether this click should be allowed. This event may also be sent in circumstances other than a mouse event:

- In `SelectWindow`
- When handling the cmd-~ key sequence
- When restoring a collapsed window from the Dock
- When handling the `kHICommandSelectWindow` command
- When activating a clicked window during application activation,

In each case, the result of this event is used to determine whether z-ordering, activation, and highlighting of the window should be allowed.

This event contains an optional EventRef parameter that is the original mouse event. If the parameter is not present, the handler should generally assume that the click was a single click.

A default handler for this event is installed on the application target. The default handler determines whether this is a modal click by examining the modality of the visible, uncollapsed windows in front of the clicked window, the location of the click, and the keyboard modifiers. A custom handler may, of course, entirely ignore window z-order or modality, and determine modality in any way it deems necessary.

(Available in Mac OS X v10.4 and later.)

Available in Mac OS X v10.4 and later.

Discussion
Events related to activating and deactivating a window.

Table 12 shows the parameters related to window activation events.

Table 12 Parameter names and types for window activation event kinds

Event kind	Parameter name	Parameter type
`kEventWindow-Activated`	`kEventParamDirectObject`	`typeWindowRef`
`kEventWindow-Deactivated`	`kEventParamDirectObject`	`typeWindowRef`

kEventWindowHandle-Activate	kEventParamDirectObject	typeWindowRef
kEventWindowHandle-Deactivate	kEventParamDirectObject	typeWindowRef
kEventWindowGetClick-Activation	kEventParamDirectObject	typeWindowRef
	kEventParamMouseLocation	typeQDPoint
	kEventParamKeyModifiers	typeUInt32
	kEventParamWindowDefPart	typeWindowDefPartCode
	kEventParamControlRef	typeControlRef
	kEventParamClickActivation	typeClickActivation-Result
kEventWindowGet-ClickModality	kEventParamDirectObject	typeWindowRef
	kEventParamWindowPartCode	typeWindowPartCode
	kEventParamKeyModifiers	typeUInt32
	kEventParamEventRef (Optional)	typeEventRef
	kEventParamModalClickResult	typeModalClickResult
	kEventParamModalWindow (Required only if kEventParamModalClickResult is kHIModalClickIsModal.)	typeWindowRef
	kEventParamWindowModality (Required only if kEventParamModalClickResult is kHIModalClickIsModal.)	typeWindowModality

Window Click Event Constants

Define constants related to events from `kEventClassWindow` occurring in the standard window controls or regions (close, resize, drag, and so on).

```
enum {
    kEventWindowClickDragRgn = 32,
    kEventWindowClickResizeRgn = 33,
    kEventWindowClickCollapseRgn = 34,
    kEventWindowClickCloseRgn = 35,
    kEventWindowClickZoomRgn = 36,
    kEventWindowClickContentRgn = 37,
    kEventWindowClickProxyIconRgn = 38,
    kEventWindowClickToolbarButtonRgn = 41,
    kEventWindowClickStructureRgn = 42
```

```
};
```

Constants

`kEventWindowClickDragRgn`
Sent when the mouse is down in the drag region. The standard window handler calls `DragWindow`.

Available in Mac OS X v10.0 and later.

`kEventWindowClickResizeRgn`
Sent when the mouse is down in the resize area. The standard window handler calls `ResizeWindow`.

Available in Mac OS X v10.0 and later.

`kEventWindowClickCollapseRgn`
Sent when the mouse is down in the collapse widget. The default behavior is to call `CollapseWindow`, and then generate `kEventWindowExpand` or `kEventWindowCollapse` (whichever is the opposite of the window's original collapse state).

Available in Mac OS X v10.0 and later.

`kEventWindowClickCloseRgn`
Sent when the mouse is down in the close widget. The standard window handler calls `TrackGoAway`, and then generates `kEventWindowClose`.

Available in Mac OS X v10.0 and later.

`kEventWindowClickZoomRgn`
Sent when the mouse is down in the zoom widget. The standard window handler calls `TrackBox`, and then generates `kEventWindowZoom`.

Available in Mac OS X v10.0 and later.

`kEventWindowClickContentRgn`
Sent when the mouse is down in the content region. The standard window handler checks for contextual menu clicks and clicks on controls, and sends `kEventWindowContextualMenuSelect`, `kEventControlClick`, and `kEventWindowHandleContentClick` events as appropriate.

Available in Mac OS X v10.0 and later.

`kEventWindowClickProxyIconRgn`
Sent when the mouse is down in the proxy icon. The standard window handler handles proxy icon dragging, and generates proxy icon events.

Available in Mac OS X v10.0 and later.

`kEventWindowClickToolbarButtonRgn`
Sent when the mouse is down in the toolbar button. The default behavior is to call `TrackBox` and then generate a `kEventWindowToolbarSwitchMode` event.

Available in Mac OS X v10.1 and later.

Discussion

Low-level events which generate higher-level "action" events described in "Window Action Event Constants" (page 173). These events are generated only for windows with the standard window handler installed. Most clients should allow the standard window handler to implement these events.

Window State Event Constants

Define constants related to events from `kEventClassWindow` that notify of a change in the window's state.

```
enum {
    kEventWindowShowing = 22,
    kEventWindowHiding = 23,
    kEventWindowShown = 24,
    kEventWindowHidden = 25,
    kEventWindowCollapsing = 86,
    kEventWindowCollapsed = 67,
    kEventWindowExpanding = 87,
    kEventWindowExpanded = 70,
    kEventWindowZoomed = 76,
    kEventWindowBoundsChanging = 26,
    kEventWindowBoundsChanged = 27,
    kEventWindowResizeStarted = 28,
    kEventWindowResizeCompleted = 29,
    kEventWindowDragStarted = 30,
    kEventWindowDragCompleted = 31,
    kEventWindowClosed = 73
};
```

Constants

`kEventWindowShowing`

A window is being shown. This is sent inside `ShowHide`. This event is propagated to all handlers that registered for the event in the event target's handler chain, regardless of return value.

The standard window handler ignores this event.

Available in Mac OS X v10.0 and later.

`kEventWindowHiding`

A window is being hidden. This is sent inside `ShowHide`. This event is propagated to all handlers that registered for the event in the event target's handler chain, regardless of return value.

The standard window handler ignores this event.

Available in Mac OS X v10.0 and later.

`kEventWindowShown`

The window has been shown. This event is propagated to all handlers that registered for the event in the event target's handler chain, regardless of return value.

The standard window handler ignores this event.

Available in Mac OS X v10.0 and later.

`kEventWindowHidden`

The window has been hidden. This event is propagated to all handlers that registered for the event in the event target's handler chain, regardless of return value.

The standard window handler ignores this event.

Available in Mac OS X v10.0 and later.

`kEventWindowCollapsing`
The window is collapsing. This event is propagated to all handlers that registered for the event in the event target's handler chain, regardless of return value.

The standard window handler ignores this event.

(Available in Mac OS X v10.1 and later.)

Available in Mac OS X v10.2 and later.

`kEventWindowCollapsed`
The object has successfully collapsed. This event is propagated to all handlers that registered for the event in the event target's handler chain, regardless of return value.

The standard window handler ignores this event.

Available in Mac OS X v10.0 and later.

`kEventWindowExpanding`
The window is expanding. This event is propagated to all handlers that registered for the event in the event target's handler chain, regardless of return value.

The standard window handler ignores this event.

(Available in Mac OS X v10.1 and later.)

Available in Mac OS X v10.2 and later.

`kEventWindowExpanded`
The window has successfully expanded. This event is propagated to all handlers that registered for the event in the event target's handler chain, regardless of return value.

The standard window handler ignores this event.

Available in Mac OS X v10.0 and later.

`kEventWindowZoomed`
The window has been successfully zoomed. This event is propagated to all handlers that registered for the event in the event target's handler chain, regardless of return value.

In CarbonLib 1.1 through 1.4 and Mac OS X prior to v10.2, this event is sent only by the standard window handler after handling `kEventWindowZoom`. In CarbonLib 1.5 and later and Mac OS X v10.2 and later, this event is sent by the Window Manager functions `ZoomWindow` and `ZoomWindowIdeal`.

The standard window handler ignores this event.

Available in Mac OS X v10.0 and later.

`kEventWindowBoundsChanging`
Sent during `DragWindow` or `ResizeWindow`, before the window is actually moved or resized. You can alter the current bounds in the event (the `kEventParamCurrentBounds` parameter) to change the eventual size and location of the window. Do not call the Window Manager functions `SizeWindow` or `SetWindowBounds` from inside a handler for this event.

In Mac OS X v10.1 and later, this event is sent before all changes to a window's bounds, whether initiated by a user or by a Window Manager call. If the event was sent in response to a user action, the `kWindowBoundsChangeUserDrag` or `kWindowBoundsChangeUserResize` attribute will be set in the `kEventParamAttributes` parameter.

The standard window handler ignores this event.

Available in Mac OS X v10.0 and later.

`kEventWindowBoundsChanged`
The window has been moved or resized (or both). Do not call the Window Manager functions `SizeWindow` or `SetWindowBounds` from inside a handler for this event. If you want to enforce certain window bounds, you should do so from a `kEventWindowBoundsChanging` event handler.

This event is propagated to all handlers that registered for the event in the event target's handler chain, regardless of return value.

In Mac OS X v10.2 and later, the standard window handler can take this event under the following conditions:

- the window uses live resizing (the `kWindowLiveResizeAttribute` attribute is set.
- the user is the one resizing the window
- an update event for the window exists in the event queue

If these conditions are met, the standard window handler removes the update event from the event queue and sends it to the event dispatcher target. Doing so simplifies redrawing window content during live resizing.

Available in Mac OS X v10.0 and later.

`kEventWindowResizeStarted`
The user has just started to resize a window. This event is propagated to all handlers that registered for the event in the event target's handler chain, regardless of return value.

The standard window handler ignores this event.

Available in Mac OS X v10.0 and later.

`kEventWindowResizeCompleted`
The user has just finished resizing a window. This event is propagated to all handlers that registered for the event in the event target's handler chain, regardless of return value.

The standard window handler ignores this event.

Available in Mac OS X v10.0 and later.

`kEventWindowDragStarted`
The user has just started to drag a window. This event is propagated to all handlers that registered for the event in the event target's handler chain, regardless of return value.

The standard window handler ignores this event.

Available in Mac OS X v10.0 and later.

`kEventWindowDragCompleted`
The user has completed a window drag. This event is propagated to all handlers that registered for the event in the event target's handler chain, regardless of return value.

The standard window handler ignores this event.

Available in Mac OS X v10.0 and later.

`kEventWindowClosed`
Dispatched by `DisposeWindow` before the window is disposed. This event is propagated to all handlers that registered for the event in the event target's handler chain, regardless of return value.

In CarbonLib 1.5 and earlier and Mac OS X prior to v10.2, if a visible window is destroyed, `kEventWindowClosedEvent` is sent before the `kEventWindowHidden` event. In CarbonLib 1.6 and Mac OS X v10.2 and later, the `kEventWindowClosed` event is sent after `kEventWindowHidden`.

The standard window handler ignores this event.

Available in Mac OS X v10.0 and later.

Discussion
Table 13 shows the parameters related to window state change events.

Table 13 Parameter names and types for window state change event kinds

Event kind	Parameter name	Parameter type
`kEventWindowBoundsChanging`	`kEventParamDirectObject`	`typeWindowRef`
	`kEventParamAttributes`	`typeUInt32`
	`kEventParamOriginalBounds`	`typeQDRectangle`
	`kEventParamPreviousBounds`	`typeQDRectangle`
	`kEventParamCurrentBounds`	`typeQDRectangle`
`kEventWindowBoundsChanged`	`kEventParamDirectObject`	`typeWindowRef`
	`kEventParamAttributes`	`typeUInt32`
	`kEventParamOriginalBounds`	`typeQDRectangle`
	`kEventParamPreviousBounds`	`typeQDRectangle`
	`kEventParamCurrentBounds`	`typeQDRectangle`
`kEventWindowShown`	`kEventParamDirectObject`	`typeWindowRef`

kEventWindowShowing	kEventParamDirectObject	typeWindowRef
kEventWindowHidden	kEventParamDirectObject	typeWindowRef
kEventWindowHiding	kEventParamDirectObject	typeWindowRef
kEventWindowResizeStarted	kEventParamDirectObject	typeWindowRef
kEventWindowResizeCompleted	kEventParamDirectObject	typeWindowRef
kEventWindowDragStarted	kEventParamDirectObject	typeWindowRef
kEventWindowDragCompleted	kEventParamDirectObject	typeWindowRef

Window Refresh Event Constants

Define constants related to window refresh events from `kEventClassWindow`.

```
enum {
    kEventWindowUpdate = 1,
    kEventWindowDrawContent = 2
};
```

Constants

`kEventWindowUpdate`

Low-level update event. Sent to any window that needs updating regardless of whether the window has the standard window handler installed. You must call the Window Manager function `BeginUpdate`, and the QuickDraw function `SetPort` before drawing your window content, then call `EndUpdate` when you are finished.

The standard window handler for this event calls `BeginUpdate` and `SetPort`, sends a `kEventWindowDrawContent` event to the window, and then calls `EndUpdate`.

Note that this event is sent directly to the window target. If no handler takes the event (that is, they all return `eventNotHandledErr`), then the Window Manager posts this event to the event queue. Doing so allows the Carbon Event Manager to convert the event into an old-style event record (`EventRecord`), to be returned from `WaitNextEvent`.

Available in Mac OS X v10.0 and later.

`kEventWindowDrawContent`

Higher-level update event sent only if you have the standard window handler installed. Functions exactly as `kEventWindowUpdate`, except that the Carbon Event Manager calls `Begin/EndUpdate` and `SetPort` for you. All you need to do is draw the window content.

The standard window handler calls `DrawControls` for this window.

Available in Mac OS X v10.0 and later.

Discussion

Events related to drawing a window's content.

Table 14 shows the parameters related to window refresh events.

Table 14 Parameter names and types for window refresh event kinds

Event kind	Parameter name	Parameter type
kEventWindowUpdate	kEventParamDirectObject	typeWindowRef
kEventWindowDrawContent	kEventParamDirectObject	typeWindowRef

Window Cursor Change Event Constant

Define a constant related to events from `kEventClassWindow` that specify that the cursor must change.

```
enum {
    kEventWindowCursorChange = 40
};
```

Constants

`kEventWindowCursorChange`

Sent when the mouse is moving over the content region. This event is used to manage ownership of the cursor. You should only change the cursor if you receive this event; otherwise, someone else needed to adjust the cursor and handled the event (for example, a TSM Input Method when the mouse is over an inline input region).

The standard handler ignores this event.

Available in Mac OS X v10.0 and later.

Discussion

Table 15 shows the parameters related to window cursor change events.

Table 15 Parameter names and types for window cursor change event kinds

Event kind	Parameter name	Parameter type
kEventWindowCursorChange	kEventParamDirectObject	typeWindowRef
	kEventParamMouseLocation	typeQDPoint
	kEventParamKeyModifiers	typeUInt32

Window Focus Event Constants

Define constants related to events from `kEventClassWindow` that describe changes in the user focus.

```
enum {
    kEventWindowFocusAcquired = 200,
    kEventWindowFocusRelinquish = 201,
    kEventWindowFocusContent = 202,
    kEventWindowFocusToolbar = 203,
    kEventWindowFocusDrawer = 204
};
```

Constants

`kEventWindowFocusAcquired`

The user (or some other action) has caused the focus to shift to your window. In response to this, you should focus any control that might need to be focused.

The standard window handler calls the Control Manager function `SetKeyboardFocus` to highlight the first control in the window.

Available in Mac OS X v10.0 and later.

`kEventWindowFocusRelinquish`

The user has shifted the focus to another window. You should take the necessary steps to unhighlight the focus and so on.

The default behavior is to clear the current keyboard focus.

Available in Mac OS X v10.0 and later.

`kEventWindowFocusContent`

Focus should be shifted to the main content area of your window. You should set the focus to the content view of your window; if that area already has focus, then do nothing.

If the content area of the window already has focus, the standard handler does nothing. Otherwise, it calls the HIView function `HIViewAdvanceFocus` to move the focus to the first control in the content area.

(Available in Mac OS X v10.2 and later.)

Available in Mac OS X v10.2 and later.

`kEventWindowFocusToolbar`

Focus should be shifted to the window's toolbar. You should set the focus to the first item in the toolbar; if the toolbar already has focus, then do nothing.

If the toolbar already has focus, the default behavior is to do nothing. Otherwise, it calls the HIView function `HIViewAdvanceFocus` to move the focus to the first control in the toolbar.

If the window does not have a toolbar, this event is not handled.

(Available in Mac OS X v10.2 and later.)

Available in Mac OS X v10.2 and later.

`kEventWindowFocusDrawer`

Focus should be shifted to the drawer of a window. You should set the focus to the first item in the drawer. If the drawer already has focus, you should move the focus to the next or previous drawer, if any, depending on whether the modifiers parameter contains the shift key modifier.If the focus is not already contained within the drawer, the basic window handler responds to this event by calling `SetUserFocusWindow` and sending a `kEventWindowFocusContent` event to the appropriate drawer.

(Available in Mac OS X v10.4 and later.)

Available in Mac OS X v10.4 and later.

Discussion
These events are related to focus changes between windows. They are generated by `SetUserFocusWindow` (page 68). Because that function is called by default only by the standard window handler, these events are normally sent only to windows with the standard window handler installed.

Table 16 shows the parameters related to window focus events.

Table 16 Parameter names and types for window focus event kinds

Event kind	Parameter name	Parameter type
`kEventWindowFocusAcquire`	`kEventParamDirectObject`	`typeWindowRef`
`kEventWindowFocusRelinquish`	`kEventParamDirectObject`	`typeWindowRef`
`kEventWindowFocusContent`	`kEventParamDirectObject`	`typeWindowRef`
`kEventWindowFocusToolbar`	`kEventParamDirectObject`	`typeWindowRef`
`kEventWindowFocusDrawer`	`kEventParamDirectObject`	`typeWindowRef`
	`kEventParamKeyModifiers`	`typeUInt32`

Window Sheet Event Constants

Define constants for events from `kEventClassWindow` that describe window sheet changes.

```
enum {
    kEventWindowSheetOpening = 210,
    kEventWindowSheetOpened = 211,
    kEventWindowSheetClosing = 212,
    kEventWindowSheetClosed = 213
};
```

Constants

`kEventWindowSheetOpening`
A sheet is opening. This event is sent to the sheet, its parent window, and the application target before the sheet begins to open. An event handler for this event may return `userCanceledErr` if the sheet should not be opened. Any other return value is ignored, and the sheet is allowed to open.

(Available in Mac OS X v10.4 and later.)

Available in Mac OS X v10.4 and later.

`kEventWindowSheetOpened`
A sheet has opened. This event is sent to the sheet, its parent window, and the application target after the sheet is fully open.

(Available in Mac OS X v10.4 and later.)

Available in Mac OS X v10.4 and later.

`kEventWindowSheetClosing`
A sheet is closing. This event is sent to the sheet, its parent window, and the application target before the sheet begins to close. An event handler for this event may return `userCanceledErr` if the sheet should not close. Any other return value is ignored, and the sheet is allowed to close.

(Available in Mac OS X v10.4 and later.)

Available in Mac OS X v10.4 and later.

`kEventWindowSheetClosed`
A sheet has closed. This event is sent to the sheet, its parent window, and the application target after the sheet is fully closed.

(Available in Mac OS X v10.4 and later.)

Available in Mac OS X v10.4 and later.

Discussion
These events are related to sheet changes. Table 17 shows the parameters related to window sheet events.

Table 17 Parameter names and types for window sheet event kinds

Event kind	Parameter name	Parameter type
`kEventWindowSheetOpening`	`kEventParamDirectObject`	`typeWindowRef`
`kEventWindowSheetOpened`	`kEventParamDirectObject`	`typeWindowRef`
`kEventWindowSheetClosing`	`kEventParamDirectObject`	`typeWindowRef`
`kEventWindowSheetClosed`	`kEventParamDirectObject`	`typeWindowRef`

Window Drawer Event Constants

Define constants related to events from `kEventClassWindow` describing window drawer changes.

```
enum {
    kEventWindowDrawerOpening = 220,
    kEventWindowDrawerOpened = 221,
    kEventWindowDrawerClosing = 222,
    kEventWindowDrawerClosed = 223
};
```

Constants
`kEventWindowDrawerOpening`
Sent to the drawer and its parent window when the drawer is opening. If you don't want the drawer to open, your handler should return `userCanceledErr`.

Available in Mac OS X v10.2 and later.

`kEventWindowDrawerOpened`
Sent to the drawer and its parent window when the drawer has fully opened.

Available in Mac OS X v10.2 and later.

`kEventWindowDrawerClosing`
Sent to the drawer and its parent window when the drawer is closing. If you don't want the drawer to close, your handler should return `userCanceledErr`.

Available in Mac OS X v10.2 and later.

`kEventWindowDrawerClosed`
Sent to the drawer and its parent when the drawer has fully closed.

Available in Mac OS X v10.2 and later.

Discussion
Table 18 shows parameters related to window drawer events.

Table 18 Parameter names and types for window drawer event kinds

Event kind	Parameter name	Parameter type
`kEventWindowDrawerOpening`	`kEventParamDirectObject`	`typeWindowRef`
`kEventWindowDrawerOpened`	`kEventParamDirectObject`	`typeWindowRef`
`kEventWindowDrawerClosing`	`kEventParamDirectObject`	`typeWindowRef`
`kEventWindowDrawerClosed`	`kEventParamDirectObject`	`typeWindowRef`

Window Definition Message Constants

Define constants for events that correspond to classic WDEF messages.

```
enum {
    kEventWindowDrawFrame = 1000,
    kEventWindowDrawPart = 1001,
    kEventWindowGetRegion = 1002,
    kEventWindowHitTest = 1003,
    kEventWindowInit = 1004,
    kEventWindowDispose = 1005,
    kEventWindowDragHilite = 1006,
    kEventWindowModified = 1007,
    kEventWindowSetupProxyDragImage = 1008,
    kEventWindowStateChanged = 1009,
    kEventWindowMeasureTitle = 1010,
    kEventWindowDrawGrowBox = 1011,
    kEventWindowGetGrowImageRegion = 1012,
    kEventWindowPaint = 1013
};
```

Constants
`kEventWindowDrawFrame`
Sent by the Window Manager when it's time to draw a window's structure. This is the replacement to the old `wDraw` defProc message (though it is a special case of the 0 part code indicating to draw the entire window frame).

Available in Mac OS X v10.0 and later.

`kEventWindowDrawPart`
Sent by the Window Manager when it's time to draw a specific part of a window's structure, such as the close box. This is typically sent during window tracking.

Available in Mac OS X v10.0 and later.

`kEventWindowGetRegion`
Sent by the Window Manager when it needs to get a specific region from a window, or when the `GetWindowRegion` function is called. The region you should modify is sent in the `kEventParamRgnHandle` parameter.

Available in Mac OS X v10.0 and later.

`kEventWindowHitTest`
Sent when the Window Manager needs to determine what part of a window would be "hit" with a given mouse location in global coordinates. If you handle this event, you should set the `kEventParamWindowDefPart` parameter to reflect the part code hit.

Available in Mac OS X v10.0 and later.

`kEventWindowInit`
Sent by the Window Manager when the window is being created. This is a hook to allow you to do any initialization you might need to do. Note that if the window definition changes, you may receive this event more than once. See the `kEventWindowDispose` event for more information.

Available in Mac OS X v10.0 and later.

`kEventWindowDispose`
Sent by the Window Manager when the window is being disposed. You should dispose of any private data structures associated with the window. Note, however, that receiving this event does not necessarily mean that the window is being destroyed. Sometimes the Window Manager may need to change the window definition (such as when `ChangeWindowAttributes` is used to change the appearance of the window). In such cases, the window receives a `kEventWindowDispose` event followed by a `kEventWindowInit` event to disconnect the old window definition and connect the new one.

If you want to know when your window is actually being destroyed, you should register for the `kEventWindowClosed` event.

Available in Mac OS X v10.0 and later.

`kEventWindowDragHilite`
Sent by the Window Manager when it is time to draw/erase any drag highlight in the window structure. This is typically sent from within the `HiliteWindowFrameForDrag` function.

Available in Mac OS X v10.0 and later.

`kEventWindowModified`
Sent by the Window Manager when it is time to redraw window structure to account for a change in the document modified state. This is typically sent from within the `SetWindowModified` function.

Available in Mac OS X v10.0 and later.

`kEventWindowSetupProxyDragImage`
Sent by the Window Manager when it is time to generate a drag image for the window proxy. This is typically sent from within the `BeginWindowProxyDrag` function.

Available in Mac OS X v10.0 and later.

`kEventWindowStateChanged`
Sent by the Window Manager when a particular window state changes. See the state-change flags in `MacWindows.h`.

Available in Mac OS X v10.0 and later.

`kEventWindowMeasureTitle`
Sent when the Window Manager needs to know how much space the window's title area takes up.

Available in Mac OS X v10.0 and later.

`kEventWindowDrawGrowBox`
This is a compatibility event used before Mac OS 8 and not useful now. When the `DrawGrowIcon` function is called, this event is sent to the window to tell it to draw the grow box. This is really needed only for windows that do not have the grow box integrated into the window frame. Scroll bar delimiter lines are also drawn.

Available in Mac OS X v10.0 and later.

`kEventWindowGetGrowImageRegion`
This is a special way for a window to override the standard resize outline for windows that do not do live resizing. As the user resizes the window, this event is sent with the current size the user has chosen expressed as a rectangle. You should calculate your window outline and modify the `kEventParamRgnHandle` parameter to reflect your desired outline.

Available in Mac OS X v10.0 and later.

`kEventWindowPaint`
Sent when it is time to draw the entire window (such as when the window is first displayed). This is a convenience event that gives you a chance to draw all the window elements at once.

If you do not handle this event, the Window Manager sends the `kEventWindowDrawFrame` event to your window and erases the content region to its background color.

(

Available in Mac OS X only.)

Discussion
These events. which correspond to WDEF messages, are sent to all windows, regardless of whether they have the standard window handler installed.

Table 19 shows the parameters related to window definition events.

Table 19 Parameter names and types for window definition event kinds

Event kind	Parameter name	Parameter type
`kEventWindowDrawFrame`	`kEventParamDirectObject`	`typeWindowRef`
`kEventWindowDrawPart`	`kEventParamDirectObject`	`typeWindowRef`

	kEventParamWindowDefPart	typeWindowDefPartCode
kEventWindowGetRegion	kEventParamDirectObject	typeWindowRef
	kEventParamWindowRegionCode	typeWindowRegionCode
	kEventParamRgnHandle	typeQDRgnHandle
kEventWindowHitTest	kEventParamDirectObject	typeWindowRef
	kEventParamMouseLocation	typeQDPoint
	kEventParamWindowDefPart	typeWindowDefPartCode
kEventWindowInit	kEventParamDirectObject	typeWindowRef
	kEventParamWindowFeatures	typeUInt32
kEventWindowDispose	kEventParamDirectObject	typeWindowRef
kEventWindowDragHilite	kEventParamDirectObject	typeWindowRef
	kEventParamWindowDragHiliteFlag	typeBoolean
kEventWindowModified	kEventParamDirectObject	typeWindowRef
	kEventParamWindowModifiedFlag	typeBoolean
kEventWindowSetup-ProxyDrag Image	kEventParamDirectObject	typeWindowRef
	kEventParamWindowProxyImageRgn	typeQDRgnHandle
	kEventParamWindowProxyOutlineRgn	typeQDRgnHandle
	kEventParamWindowProxyGWorlPtr	typeGWorldPtr
kEventWindowState-Changed	kEventParamDirectObject	typeWindowRef
	kEventParamWindowStateChangedFlags	typeUInt32
kEventWindowMeasure-Title	kEventParamDirectObject	typeWindowRef
	kEventParamWindowTitleFullWidth	typeSInt16
	kEventParamWindowTitleTextWidth	typeSInt16
kEventWindow-DrawGrowBox	kEventParamDirectObject	typeWindowRef
kEventWindowGetGrow-Image Region	kEventParamDirectObject	typeWindowRef
	kEventParamWindowGrowRect	typeQDRectangle

	kEventParamRgnHandle	typeQDRectangle
kEventWindowPaint	kEventParamDirectObject	typeWindowRef

Alternate Window Definition Event Constants

Define alternate names for window definition events.

```
enum {
    kEventWindowDefDrawFrame = kEventWindowDrawFrame,
    kEventWindowDefDrawPart = kEventWindowDrawPart,
    kEventWindowDefGetRegion = kEventWindowGetRegion,
    kEventWindowDefHitTest = kEventWindowHitTest,
    kEventWindowDefInit = kEventWindowInit,
    kEventWindowDefDispose = kEventWindowDispose,
    kEventWindowDefDragHilite = kEventWindowDragHilite,
    kEventWindowDefModified = kEventWindowModified,
    kEventWindowDefSetupProxyDragImage = kEventWindowSetupProxyDragImage,
    kEventWindowDefStateChanged = kEventWindowStateChanged,
    kEventWindowDefMeasureTitle = kEventWindowMeasureTitle,
    kEventWindowDefDrawGrowBox = kEventWindowDrawGrowBox,
    kEventWindowDefGetGrowImageRegion = kEventWindowGetGrowImageRegion
};
```

Constants

`kEventWindowDefDrawFrame`
Equivalent to `kEventWindowDrawFrame`.

Available in Mac OS X v10.0 and later.

`kEventWindowDefDrawPart`
Equivalent to `kEventWindowDrawPart`.

Available in Mac OS X v10.0 and later.

`kEventWindowDefGetRegion`
Equivalent to `kEventWindowGetRegion`.

Available in Mac OS X v10.0 and later.

`kEventWindowDefHitTest`
Equivalent to `kEventWindowHitTest`.

Available in Mac OS X v10.0 and later.

`kEventWindowDefInit`
Equivalent to `kEventWindowInit`.

Available in Mac OS X v10.0 and later.

`kEventWindowDefDispose`
Equivalent to `kEventWindowDispose`.

Available in Mac OS X v10.0 and later.

`kEventWindowDefDragHilite`
Equivalent to `kEventWindowDragHilite`.

Available in Mac OS X v10.0 and later.

`kEventWindowDefModified`
Equivalent to `kEventWindowModified`.

Available in Mac OS X v10.0 and later.

`kEventWindowDefSetupProxyDragImage`
Equivalent to `kEventWindowSetupProxyDragImage`.

Available in Mac OS X v10.0 and later.

`kEventWindowDefStateChanged`
Equivalent to `kEventWindowStateChanged`.

Available in Mac OS X v10.0 and later.

`kEventWindowDefMeasureTitle`
Equivalent to `kEventWindowMeasureTitle`.

Available in Mac OS X v10.0 and later.

`kEventWindowDefDrawGrowBox`
Equivalent to `kEventWindowDrawGrowBox`.

Available in Mac OS X v10.0 and later.

`kEventWindowDefGetGrowImageRegion`
Equivalent to `kEventWindowGetGrowImageRegion`.

Available in Mac OS X v10.0 and later.

Discussion
For clarity, you can use these event names in place of the standard window events when using them in your custom window definitions. For descriptions of these events, see "Window Definition Message Constants" (page 193).

Window Bounds Attributes

Define constants that describe how a window's bounds are changing.

```
enum {
    kWindowBoundsChangeUserDrag = (1 << 0),
    kWindowBoundsChangeUserResize = (1 << 1),
    kWindowBoundsChangeSizeChanged = (1 << 2),
    kWindowBoundsChangeOriginChanged = (1 << 3),
    kWindowBoundsChangeZoom = (1 << 4)
};
```

Constants
`kWindowBoundsChangeUserDrag`
The bounds are changing because the user is dragging the window around.

Available in Mac OS X v10.0 and later.

`kWindowBoundsChangeUserResize`
The bounds are changing because the user is resizing the window.

Available in Mac OS X v10.0 and later.

`kWindowBoundsChangeSizeChanged`
The dimensions of the window (width and height) are changing.

Available in Mac OS X v10.0 and later.

`kWindowBoundsChangeOriginChanged`
The origin of the window is changing.

Available in Mac OS X v10.0 and later.

`kWindowBoundsChangeZoom`
The bounds are changing as a result of the user clicking the zoom button.

Available in Mac OS X v10.2 and later.

Discussion

When the toolbox sends out a `kEventWindowBoundsChanging` or `kEventWindowBoundsChanged` event, it also sends along a parameter containing attributes of the event (`kEventParamAttributes`). You can use these attributes to determine what aspect of the window changed (origin, size, or both), and whether or not some user action is driving the change (drag, resize, or zoom).

User Focus Auto-Select Constant

Defines a constant that tells the system to pick the best user focus window.

```
enum {
    kUserFocusAuto = -1
};
```

Constants

`kUserFocusAuto`
Pass this constant for the window reference in `SetUserFocusWindow` (page 68) to have the system choose the most appropriate window for user focus.

Available in Mac OS X v10.0 and later.

Window Event Parameters and Types

Define constants for parameters and attributes related to window events.

```
enum {
    kEventParamWindowFeatures = 'wftr',
    kEventParamWindowDefPart = 'wdpc',
    kEventParamWindowPartCode = wpar',
    kEventParamCurrentBounds = 'crct',
    kEventParamOriginalBounds = 'orct',
    kEventParamPreviousBounds = 'prct',
    kEventParamClickActivation = 'clac',
    kEventParamWindowRegionCode = 'wshp',
    kEventParamWindowDragHiliteFlag = 'wdhf',
    kEventParamWindowModifiedFlag = 'wmff',
```

```
    kEventParamWindowProxyGWorldPtr = 'wpgw',
    kEventParamWindowProxyImageRgn = 'wpir',
    kEventParamWindowProxyOutlineRgn = 'wpor',
    kEventParamWindowStateChangedFlags = 'wscf',
    kEventParamWindowTitleFullWidth = 'wtfw',
    kEventParamWindowTitleTextWidth = 'wttw',
    kEventParamWindowGrowRect = 'grct',
    kEventParamPreviousDockRect = 'pdrc',
    kEventParamPreviousDockDevice = 'pdgd',
    kEventParamCurrentDockRect = 'cdrc',
    kEventParamCurrentDockDevice = 'cdgd',
    kEventParamWindowTransitionAction = 'wtac',
    kEventParamWindowTransitionEffect = 'wtef',
    typeWindowRegionCode = 'wshp',
    typeWindowDefPartCode = 'wdpt',
    typeClickActivationResult = 'clac',
    typeWindowTransitionAction = 'wtac',
    typeWindowTransitionEffect = 'wtef'
};
```

Constants

`kEventParamWindowFeatures`
typeUInt32

Available in Mac OS X v10.0 and later.

`kEventParamWindowDefPart`
typeWindowDefPartCode

Available in Mac OS X v10.0 and later.

`kEventParamWindowPartCode`
typeWindowPartCode

Available in Mac OS X v10.3 and later.

`kEventParamCurrentBounds`
typeQDRectangle

Available in Mac OS X v10.0 and later.

`kEventParamOriginalBounds`
typeQDRectangle

Available in Mac OS X v10.0 and later.

`kEventParamPreviousBounds`
typeQDRectangle

Available in Mac OS X v10.0 and later.

`kEventParamClickActivation`
typeClickActivationResult

Available in Mac OS X v10.0 and later.

`kEventParamWindowRegionCode`
typeWindowRegionCode

Available in Mac OS X v10.0 and later.

`kEventParamWindowDragHiliteFlag`
typeBoolean

Available in Mac OS X v10.0 and later.

`kEventParamWindowModifiedFlag`
typeBoolean

Available in Mac OS X v10.0 and later.

`kEventParamWindowProxyGWorldPtr`
typeGWorldPtr

Available in Mac OS X v10.0 and later.

`kEventParamWindowProxyImageRgn`
typeQDRgnHandle

Available in Mac OS X v10.0 and later.

`kEventParamWindowProxyOutlineRgn`
typeQDRgnHandle

Available in Mac OS X v10.0 and later.

`kEventParamWindowStateChangedFlags`
typeUInt32

Available in Mac OS X v10.0 and later.

`kEventParamWindowTitleFullWidth`
typeSInt16

Available in Mac OS X v10.0 and later.

`kEventParamWindowTitleTextWidth`
typeSInt16

Available in Mac OS X v10.0 and later.

`kEventParamWindowGrowRect`
typeQDRectangle

Available in Mac OS X v10.0 and later.

`kEventParamPreviousDockRect`
typeHIRect

Available in Mac OS X v10.2 and later.

`kEventParamPreviousDockDevice`
typeGDHandle

Available in Mac OS X v10.3 and later.

`kEventParamCurrentDockRect`
typeHIRect

Available in Mac OS X v10.2 and later.

`kEventParamCurrentDockDevice`
typeGDHandle

Available in Mac OS X v10.3 and later.

`kEventParamWindowTransitionAction`
typeWindowTransitionAction

Available in Mac OS X v10.3 and later.

`kEventParamWindowTransitionEffect`
typeWindowTransitionEffect

Available in Mac OS X v10.3 and later.

`typeWindowRegionCode`
WindowRegionCode

Available in Mac OS X v10.0 and later.

`typeWindowDefPartCode`
WindowDefPartCode

Available in Mac OS X v10.0 and later.

`typeWindowPartCode`
WindowPartCode

Available in Mac OS X v10.3 and later.

`typeClickActivationResult`
ClickActivationResult

Available in Mac OS X v10.0 and later.

`typeWindowTransitionAction`
WindowTransitionAction

Available in Mac OS X v10.3 and later.

`typeWindowTransitionEffect`
WindowTransitionEffect

Available in Mac OS X v10.3 and later.

Modal Window Event Parameters and Types

Define constants related to events from `kEventClassWindow` used to determine whether a mouse-down or mouse-up event is blocked by a modal window.

```
enum {
    typeModalClickResult = 'wmcr',
    typeWindowModality = 'wmod',
    kEventParamModalClickResult = typeModalClickResult,
    kEventParamModalWindow = 'mwin',
    kEventParamWindowModality = typeWindowModality
};
```

Constants

`typeModalClickResult`
HIModalClickResult

Available in Mac OS X v10.4 and later.

`typeWindowModality`
WindowModality

Available in Mac OS X v10.4 and later.

`kEventParamModalClickResult`
On exit, a value indicating how the click should be handled. For details, see "Modal Window Click Constants" (page 203).

Available in Mac OS X v10.4 and later.

`kEventParamModalWindow`
On exit, the modal window that caused the click to be blocked, if any. The sender of this event uses this information to determine which window should be activated if the application is inactive. This parameter is only required if the `kEventParamModalClickResult` parameter contains `kHIModalClickIsModal`. If an event handler wants to report that a click has been blocked by modality, but cannot determine which window blocked the click, it is acceptable to either not add this parameter to the event, or to set the parameter to a `NULL` window reference.

Available in Mac OS X v10.4 and later.

`kEventParamWindowModality`
On exit, the modality of the modal window that is in front of the clicked window, if any. This parameter is required only if the `kEventParamModalClickResult` parameter contains `kHIModalClickIsModal`.

Available in Mac OS X v10.4 and later.

Availability
Available in Mac OS X v10.4 and later.

Modal Window Click Constants

Define constants that describe responses to the `kEventWindowGetModalityClick` event.

```
typedef UInt32 HIModalClickResult;
enum {
    kHIModalClickIsModal = 1 << 0,
    kHIModalClickAllowEvent = 1 << 1,
    kHIModalClickAnnounce = 1 << 2,
    kHIModalClickRaiseWindow = 1 << 3,
};
```

Constants

`kHIModalClickIsModal`

A modal window prevents the mouse event from being passed to the clicked window. If this bit is set, the `kEventParamModalWindow` and `kEventParamWindowModality` parameters should be set before the event handler returns. If this bit is clear, normal event handling occurs: the clicked window is typically z-ordered to the top of its window group, activated, becomes the user focus window, and receives the mouse event for further processing.

Available in Mac OS X v10.4 and later.

`kHIModalClickAllowEvent`

If `kHIModalClickIsModal` is set, the `kHIModalClickAllowEvent` flag indicates whether the click event should be allowed to pass to the clicked window. If `kHIModalClickIsModal` is not set, the setting of `kHIModalClickAllowEvent` is ignored.

Available in Mac OS X v10.4 and later.

`kHIModalClickAnnounce`

If `kHIModalClickIsModal` is set and `kHIModalClickAllowEvent` is not set, `kHIModalClickAnnounce` indicates whether the caller should announce that the click has been blocked by a modal window using the appropriate UI (typically, by calling `SysBeep`). If `kHIModalClickIsModal` is not set, or if `kHIModalClickAllowEvent` is set, the setting of `kHIModalClickAnnounce` is ignored.

Available in Mac OS X v10.4 and later.

`kHIModalClickRaiseWindow`

If `kHIModalClickIsModal` and `kHIModalClickAllowEvent` are set, `kHIModalClickRaiseWindow` indicates whether the clicked window should be z-ordered to the top of its window group. The window is not, however, activated, nor does it become the user focus window. If `kHIModalClickIsModal` or `kHIModalClickAllowEvent` is not set, `kHIModalClickRaiseWindow` is ignored.

Available in Mac OS X v10.4 and later.

Result Codes

The most common result codes returned by the Carbon Event Manager are listed below.

Result Code	Value	Description
`noErr`	0	No error.
`eventAlreadyPostedErr`	-9860	Returned from `PostEventToQueue` (page 54) if the event in question is already in the queue you are posting it to (or any other queue).
`eventTargetBusyErr`	-9861	The event target you are trying to modify is busy (for example, dispatching an event).
`eventClassInvalidErr`	-9862	This is obsolete and will be removed.
`eventClassIncorrectErr`	-9864	This is obsolete and will be removed.

Result Code	Value	Description
eventHandlerAlreadyInstalledErr	-9866	Returned from InstallEventHandler (page 43) if the handler callback you pass is already installed for a given event type you are trying to register.
eventInternalErr	-9868	A generic error.
eventKindIncorrectErr	-9869	This is obsolete and will be removed.
eventParameterNotFoundErr	-9870	The piece of data you are requesting from an event is not present.
eventNotHandledErr	-9874	This is what you should return from an event handler when your handler has received an event it doesn't currently want to (or isn't able to) handle. If you handle an event, you should return noErr from your event handler.
eventLoopTimedOutErr	-9875	The event loop has timed out. This can be returned from calls to ReceiveNextEvent (page 57) or RunCurrentEventLoop (page 64).
eventLoopQuitErr	-9876	The event loop was quit, probably by a call to QuitEventLoop (page 56). This can be returned from ReceiveNextEvent (page 57) or RunCurrentEventLoop (page 64).
eventNotInQueueErr	-9877	Returned from RemoveEventFromQueue (page 60) when trying to remove an event that's not in any queue.
eventHotKeyExistsErr	-9878	The hot key combination you chose already exists
eventHotKeyInvalidErr	-9879	This error code is not currently used.

Deprecated Carbon Event Manager Functions

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

Deprecated in Mac OS X v10.4

ChangeMouseTrackingRegion

(Deprecated in Mac OS X v10.4. Use `HIViewChangeTrackingArea` instead.)

Special Considerations

Tracking areas are HIView-based rather than window-based. HIViews support compositing and Quartz, and provide a much easier way to handle user elements in windows. For more details about tracking areas, see the mouse tracking region section in *Carbon Event Manager Programming Guide*. For details about HIViews, see *HIView Programming Guide*.

Availability

Available in Mac OS X v10.2 and later.

Deprecated in Mac OS X v10.4.

Declared In

CarbonEvents.h

ClipMouseTrackingRegion

(Deprecated in Mac OS X v10.4. No replacement function. Use HIView-based mouse tracking areas instead.)

Special Considerations

Tracking areas are HIView-based rather than window-based. HIViews support compositing and Quartz, and provide a much easier way to handle user elements in windows. You generally don't need to modify the clipping of a tracking area. For more details about tracking areas, see the mouse tracking region section in *Carbon Event Manager Programming Guide*. For details about HIViews, see *HIView Programming Guide*.

Availability
Available in Mac OS X v10.2 and later.
Deprecated in Mac OS X v10.4.

Declared In
CarbonEvents.h

ClipWindowMouseTrackingRegions

(Deprecated in Mac OS X v10.4. No replacement function. Use HIView-based tracking areas instead.)

Special Considerations

Tracking areas are HIView-based rather than window-based. HIViews support compositing and Quartz, and provide a much easier way to handle user elements in windows. You generally don't need to modify the clipping of a tracking area. For more details about tracking areas, see the mouse tracking region section in *Carbon Event Manager Programming Guide*. For details about HIViews, see *HIView Programming Guide*.

Availability
Available in Mac OS X v10.2 and later.
Deprecated in Mac OS X v10.4.

Declared In
CarbonEvents.h

CreateMouseTrackingRegion

Creates a mouse tracking region. (Deprecated in Mac OS X v10.4. Use the HIView function `HIViewNewTrackingArea` instead.)

```
OSStatus CreateMouseTrackingRegion(
   WindowRef inWindow,
   RgnHandle inRegion,
   RgnHandle inClip,
   MouseTrackingOptions inOptions,
   MouseTrackingRegionID inID,
   void * inRefCon,
   EventTargetRef inTargetToNotify,
   MouseTrackingRef * outTrackingRef
);
```

Parameters

inWindow
The window to contain the tracking region.

inRegion
The region for which you want to receive mouse entered/exited events.

inClip
The clip region for the `inRegion` region (can be `NULL`).

inOptions
Tracking options that define whether the `inRegion` region is in local or global coordinates.

inID
A signature and ID to uniquely define this tracking region. See `MouseTrackingRegionID` (page 83) for information about the structure of this ID.

inRefCon
A pointer to an application-defined value. You can obtain this value by calling `GetMouseTrackingRegionRefCon` (page 210).

inTargetToNotify
The event target to send the mouse tracking event. If you pass `NULL`, the event target is the owning window specified in `inWindow`.

outTrackingRef
On return, a pointer to the new mouse tracking region.

Return value
A result code. See "Carbon Event Manager Result Codes" (page 204).

Discussion
`CreateMouseTrackingRegion` allows you to define regions in your window, and the specified event target is notified (using `kEventMouseEntered` or `kEventMouseExited` events) when the mouse cursor interacts with the region. Your application can define any number of regions as long as each has a unique ID. This function is especially useful for creating rollover effects without having to constantly poll the mouse.

If you need to keep track of the state of the mouse (down or up) in a region, you should use `TrackMouseRegion` (page 72), either instead of, or in conjunction with, mouse tracking regions.

Special Considerations
Tracking areas are HIView-based rather than window-based. HIViews support compositing and Quartz, and provide a much easier way to handle user elements in windows. For more details about tracking areas, see the mouse tracking region section in *Carbon Event Manager Programming Guide*. For details about HIViews, see *HIView Programming Guide*.

Availability
Available in Mac OS X v10.2 and later.
Deprecated in Mac OS X v10.4.

GetMouseTrackingRegionID
(Deprecated in Mac OS X v10.4. Use `HIViewGetTrackingAreaID` instead.)

Special Considerations
Tracking areas are HIView-based rather than window-based. HIViews support compositing and Quartz, and provide a much easier way to handle user elements in windows. For more details about tracking areas, see the mouse tracking region section in *Carbon Event Manager Programming Guide*. For details about HIViews, see *HIView Programming Guide*.

Availability
Available in Mac OS X v10.2 and later.

Deprecated in Mac OS X v10.4.

Declared In
CarbonEvents.h

GetMouseTrackingRegionRefCon

Obtains the reference constant for a mouse tracking region. (Deprecated in Mac OS X v10.4. No replacement function. Use HIView-based mouse tracking areas instead.)

```
OSStatus GetMouseTrackingRegionRefCon(
    MouseTrackingRef inMouseRef,
    void ** outRefCon
);
```

Parameters

inMouseRef
The mouse tracking region whose reference count you want to obtain.

outRefCon
On return, a handler for the mouse tracking region.

Return value
A result code. See "Carbon Event Manager Result Codes" (page 204).

Discussion
You use this function to obtain the reference constant you set in the `CreateMouseTrackingRegion` (page 208) function.

Special Considerations

Tracking areas are HIView-based rather than window-based. HIViews support compositing and Quartz, and provide a much easier way to handle user elements in windows. Mouse tracking areas do not support a reference constant. Instead, you can obtain the tracking area ID (using `HIViewGetTrackingAreaID`) and use that as a key to look up extended data in your own tables. For more details about tracking areas, see the mouse tracking region section in *Carbon Event Manager Programming Guide*. For details about HIViews, see *HIView Programming Guide*.

Availability
Available in Mac OS X v10.2 and later.
Deprecated in Mac OS X v10.4.

Declared In
CarbonEvents.h

MoveMouseTrackingRegion

(Deprecated in Mac OS X v10.4. No replacement function. Use HIView-based tracking areas instead.)

Special Considerations

Tracking areas are HIView-based rather than window-based. HIViews support compositing and Quartz, and provide a much easier way to handle user elements in windows. HIView-based mouse tracking areas move automatically when the HIView moves. For more details about tracking areas, see the mouse tracking region section in *Carbon Event Manager Programming Guide*. For details about HIViews, see *HIView Programming Guide*.

Availability

Available in Mac OS X v10.2 and later.

Deprecated in Mac OS X v10.4.

Declared In

CarbonEvents.h

MoveWindowMouseTrackingRegions

(Deprecated in Mac OS X v10.4. No replacement function. Use HIView-based tracking areas instead.)

Special Considerations

Tracking areas are HIView-based rather than window-based. HIViews support compositing and Quartz, and provide a much easier way to handle user elements in windows. HIView-based mouse tracking areas move automatically when the HIView moves. For more details about tracking areas, see the mouse tracking region section in *Carbon Event Manager Programming Guide*. For details about HIViews, see *HIView Programming Guide*.

Availability

Available in Mac OS X v10.2 and later.

Deprecated in Mac OS X v10.4.

Declared In

CarbonEvents.h

ReleaseMouseTrackingRegion

Releases a mouse tracking region. (Deprecated in Mac OS X v10.4. Use `HIViewDisposeTrackingArea` instead.)

```
OSStatus ReleaseMouseTrackingRegion (
     MouseTrackingRef inMouseRef
);
```

Parameters

inMouseRef

The mouse tracking region to release.

Return value

A result code. See "Carbon Event Manager Result Codes" (page 204).

Discussion

`ReleaseMouseTrackingRegion` decreases the reference count for the region. If the reference count drops to zero, the mouse tracking region is disposed.

Special Considerations

Tracking areas are HIView-based rather than window-based. HIViews support compositing and Quartz, and provide a much easier way to handle user elements in windows. For more details about tracking areas, see the mouse tracking region section in *Carbon Event Manager Programming Guide*. For details about HIViews, see *HIView Programming Guide*.

Availability

Available in Mac OS X v10.2 and later.

Deprecated in Mac OS X v10.4.

Declared In

CarbonEvents.h

ReleaseWindowMouseTrackingRegions

(Deprecated in Mac OS X v10.4. No replacement function. Use HIView-based tracking areas instead.)

Special Considerations

Tracking areas are HIView-based rather than window-based. HIViews support compositing and Quartz, and provide a much easier way to handle user elements in windows. If you need to release multiple tracking areas at once, you should keep track of them in your own data structures and release each one. For more details about tracking areas, see the mouse tracking region section in *Carbon Event Manager Programming Guide*. For details about HIViews, see *HIView Programming Guide*.

Availability

Available in Mac OS X v10.2 and later.

Deprecated in Mac OS X v10.4.

Declared In

CarbonEvents.h

RetainMouseTrackingRegion

Retains a mouse tracking region. (Deprecated in Mac OS X v10.4. No replacement function. Use HIView-based tracking areas instead.)

```
OSStatus RetainMouseTrackingRegion (
     MouseTrackingRef inMouseRef
);
```

Parameters

inMouseRef

The mouse tracking region to retain.

Return value

A result code. See "Carbon Event Manager Result Codes" (page 204).

Discussion

`RetainMouseTrackingRegion` increases the reference count for the region.

Special Considerations

Tracking areas are HIView-based rather than window-based. HIViews support compositing and Quartz, and provide a much easier way to handle user elements in windows. Mouse tracking areas do not have a retain/release semantic, so there is no direct replacement for `RetainMouseTrackingRegion`. For more details about tracking areas, see the mouse tracking region section in *Carbon Event Manager Programming Guide*. For details about HIViews, see *HIView Programming Guide*.

Availability

Available in Mac OS X v10.2 and later.

Deprecated in Mac OS X v10.4.

Declared In

CarbonEvents.h

SetMouseTrackingRegionEnabled

(Deprecated in Mac OS X v10.4. No replacement function. Use HIView-based tracking areas instead.)

Special Considerations

Tracking areas are HIView-based rather than window-based. HIViews support compositing and Quartz, and provide a much easier way to handle user elements in windows. To disable tracking areas, you can either delete the tracking area or ignore kEventControlTrackingAreaEntered events. For more details about tracking areas, see the mouse tracking region section in *Carbon Event Manager Programming Guide*. For details about HIViews, see *HIView Programming Guide*.

Availability

Available in Mac OS X v10.2 and later.

Deprecated in Mac OS X v10.4.

Declared In

CarbonEvents.h

SetWindowMouseTrackingRegionsEnabled

(Deprecated in Mac OS X v10.4. Use HIView-based tracking areas instead.)

Special Considerations

Tracking areas are HIView-based rather than window-based. HIViews support compositing and Quartz, and provide a much easier way to handle user elements in windows. To disable tracking areas, you can either delete the tracking area or ignore kEventControlTrackingAreaEntered events. For more details about tracking areas, see the mouse tracking region section in *Carbon Event Manager Programming Guide*. For details about HIViews, see *HIView Programming Guide*.

Availability

Available in Mac OS X v10.2 and later.

Deprecated in Mac OS X v10.4.

Declared In
CarbonEvents.h

Document Revision History

This table describes the changes to *Carbon Event Manager Reference*.

Date	Notes
2006-07-24	Added more deprecation information. Added documentation for GetCurrentEventKeyModifiers. Fixed minor bugs.
2006-03-08	Added information about deprecated functions.
2005-08-11	Added additional bug fixes and updates.
2005-06-04	Updated for Mac OS X v10.4. Includes bug fixes.
2004-02-01	Added `GetMouseTrackingRegionRefCon` (page 210).
	Correction to `CreateMouseTrackingRegion` (page 208): You must call `GetMouseTrackingRegionRefCon` (page 210) to obtain the reference constant; it is not automatically passed to your event handler.
2003-04-01	Added documentation for the function `GetLastUserEventTime`.
	Fixed formatting errors.
2003-01-01	Updated formatting and linking.
	Added standard handler and default behavior for many events.
	Added parameter tables for various events from *Inside Mac OS X: Handling Carbon Events*.
	Updated some API information for Jaguar (Mac OS X v10.2).

Document Revision History

Index

F

G

H

I

K

M

N

O

P

Q

R

S

T

U

V

W