Quartz Event Services Reference

Carbon > Events & Other Input

2007-10-31

Contents

Quartz Event Services Reference

Framework:	ApplicationServices/ApplicationServices.h
Declared in	CGEvent.h CGEventSource.h CGEventTypes.h CGRemoteOperation.h

Overview

This document describes the C API for event taps, which are filters used to observe and alter the stream of low-level user input events in Mac OS X. Event taps make it possible to monitor and filter input events from several points within the system, prior to their delivery to a foreground application. Event taps complement and extend the capabilities of the Carbon event monitor mechanism, which allows an application to observe input events delivered to other processes (see the function `GetEventMonitorTarget`).

Event taps are designed to serve as a Section 508 enabling technology. For example, consider a software system to assist a person with language impairments, designed to perform keyboard filtering with spoken review. Such a system could use an event tap to monitor all keystrokes, perform dictionary checks and matches, and recite the assembled word back to the user on detection of a word break in the input stream. If acceptable to the user, as indicated by an additional input keystroke or other gesture, the events would be posted into the system for delivery to the foreground application.

Introduced in Mac OS X version 10.4, event taps provide functionality similar to the Win32 functions `SetWinEventHook` when used to establish an out-of-context event hook, and `SendInput`. Quartz Event Services also includes an older set of event-related functions declared in the file `CGRemoteOperation.h`. These functions are still supported, but they are not recommended for new development.

Functions by Task

Working With Quartz Events

`CGEventGetTypeID` (page 19)
: Returns the type identifier for the opaque type `CGEventRef`.

`CGEventCreate` (page 11)
: Returns a new Quartz event.

`CGEventCreateData` (page 12)
: Returns a flattened data representation of a Quartz event.

`CGEventCreateFromData` (page 13)
: Returns a Quartz event created from a flattened data representation of the event.

`CGEventCreateMouseEvent` (page 14)
: Returns a new Quartz mouse event.

`CGEventCreateKeyboardEvent` (page 13)
: Returns a new Quartz keyboard event.

`CGEventCreateScrollWheelEvent` (page 15)
: Returns a new Quartz scrolling event.

`CGEventCreateCopy` (page 12)
: Returns a copy of an existing Quartz event.

`CGEventCreateSourceFromEvent` (page 16)
: Returns a Quartz event source created from an existing Quartz event.

`CGEventSetSource` (page 25)
: Sets the event source of a Quartz event.

`CGEventGetType` (page 18)
: Returns the event type of a Quartz event (left mouse down, for example).

`CGEventSetType` (page 25)
: Sets the event type of a Quartz event (left mouse down, for example).

`CGEventGetTimestamp` (page 18)
: Returns the timestamp of a Quartz event.

`CGEventSetTimestamp` (page 25)
: Sets the timestamp of a Quartz event.

`CGEventGetLocation` (page 18)
: Returns the location of a Quartz mouse event.

`CGEventGetUnflippedLocation` (page 19)
: Returns the location of a Quartz mouse event.

`CGEventSetLocation` (page 24)
: Sets the location of a Quartz mouse event.

`CGEventGetFlags` (page 17)
: Returns the event flags of a Quartz event.

`CGEventSetFlags` (page 23)
: Sets the event flags of a Quartz event.

`CGEventKeyboardGetUnicodeString` (page 20)
: Returns the Unicode string associated with a Quartz keyboard event.

`CGEventKeyboardSetUnicodeString` (page 20)
: Sets the Unicode string associated with a Quartz keyboard event.

`CGEventGetIntegerValueField` (page 17)
: Returns the integer value of a field in a Quartz event.

`CGEventSetIntegerValueField` (page 24)
: Sets the integer value of a field in a Quartz event.

`CGEventGetDoubleValueField` (page 16)
: Returns the floating-point value of a field in a Quartz event.

`CGEventSetDoubleValueField` (page 23)
: Sets the floating-point value of a field in a Quartz event.

Working With Quartz Event Taps

`CGEventTapCreate` (page 35)
  Creates an event tap.

`CGEventTapCreateForPSN` (page 36)
  Creates an event tap for a specified process.

`CGEventTapEnable` (page 37)
  Enables or disables an event tap.

`CGEventTapIsEnabled` (page 38)
  Returns a Boolean value indicating whether an event tap is enabled.

`CGEventTapPostEvent` (page 38)
  Posts a Quartz event from an event tap into the event stream.

`CGEventPost` (page 22)
  Posts a Quartz event into the event stream at a specified location.

`CGEventPostToPSN` (page 22)
  Posts a Quartz event into the event stream for a specific application.

`CGGetEventTapList` (page 39)
  Gets a list of currently installed event taps.

`CGEventMaskBit` (page 21)
  Generates an event mask for a single type of event.

Working With Quartz Event Sources

`CGEventSourceGetTypeID` (page 31)
  Returns the type identifier for the opaque type `CGEventSourceRef`.

`CGEventSourceCreate` (page 27)
  Returns a Quartz event source created with a specified source state.

`CGEventSourceGetKeyboardType` (page 28)
  Returns the keyboard type to be used with a Quartz event source.

`CGEventSourceSetKeyboardType` (page 32)
  Sets the keyboard type to be used with a Quartz event source.

`CGEventSourceGetSourceStateID` (page 30)
  Returns the source state associated with a Quartz event source.

`CGEventSourceButtonState` (page 26)
  Returns a Boolean value indicating the current button state of a Quartz event source.

`CGEventSourceKeyState` (page 31)
  Returns a Boolean value indicating the current keyboard state of a Quartz event source.

`CGEventSourceFlagsState` (page 27)
  Returns the current flags of a Quartz event source.

`CGEventSourceSecondsSinceLastEventType` (page 32)
  Returns the elapsed time since the last event for a Quartz event source.

`CGEventSourceCounterForEventType` (page 26)
  Returns a count of events of a given type seen since the window server started.

`CGEventSourceGetUserData` (page 31)
: Returns the 64-bit user-specified data for a Quartz event source.

`CGEventSourceSetUserData` (page 34)
: Sets the 64-bit user-specified data for a Quartz event source.

`CGEventSourceGetLocalEventsFilterDuringSuppressionState` (page 28)
: Returns the mask that indicates which classes of local hardware events are enabled during event suppression.

`CGEventSourceSetLocalEventsFilterDuringSuppressionState` (page 33)
: Sets the mask that indicates which classes of local hardware events are enabled during event suppression.

`CGEventSourceGetLocalEventsSuppressionInterval` (page 29)
: Returns the interval that local hardware events may be suppressed following the posting of a Quartz event.

`CGEventSourceSetLocalEventsSuppressionInterval` (page 33)
: Sets the interval that local hardware events may be suppressed following the posting of a Quartz event.

`CGEventSourceGetPixelsPerLine` (page 29)
: Gets the scale of pixels per line in a scrolling event source.

`CGEventSourceSetPixelsPerLine` (page 34)
: Sets the scale of pixels per line in a scrolling event source.

Obsolete Functions

`CGPostKeyboardEvent` (page 40)
: Synthesizes a low-level keyboard event on the local machine.

`CGPostMouseEvent` (page 41)
: Synthesizes a low-level mouse-button event on the local machine.

`CGPostScrollWheelEvent` (page 41)
: Synthesizes a low-level scrolling event on the local machine.

`CGEnableEventStateCombining` (page 11)
: Enables or disables the merging of actual key and mouse state with the application-specified state in a synthetic event.

`CGInhibitLocalEvents` (page 39)
: Turns off local hardware events in the current session.

`CGSetLocalEventsFilterDuringSuppressionState` (page 42)
: Filters local hardware events from the keyboard and mouse during the short interval after a synthetic event is posted.

`CGSetLocalEventsSuppressionInterval` (page 43)
: Sets the time interval in seconds that local hardware events are suppressed after posting a synthetic event.

`CGEventGetSource` (page 69) Deprecated in Mac OS X v10.4
: Returns a Quartz event source created from an existing Quartz event. (Deprecated. Use `CGEventCreateSourceFromEvent` (page 16) instead.)

Functions

CGEnableEventStateCombining

Enables or disables the merging of actual key and mouse state with the application-specified state in a synthetic event.

```
CGError CGEnableEventStateCombining (
   boolean_t doCombineState
);
```

Parameters

doCombineState

Pass `true` to specify that the actual key and mouse state are merged with the application-specified state in a synthetic event; otherwise, pass `false`.

Return Value

A result code. See the result codes described in *Quartz Display Services Reference*.

Discussion

By default, the flags that indicate modifier key state (Command, Option, Shift, Control, and so on) from the system's keyboard and from other event sources are ORed together as an event is posted into the system, and current key and mouse button state is considered in generating new events. This function allows your application to enable or disable the merging of event state. When combining is turned off, the event state propagated in the events posted by your application reflect state built up only by your application. The state within your application's generated event will not be combined with the system's current state, so the system-wide state reflecting key and mouse button state will remain unchanged. When called with `doCombineState` equal to `false`, this function initializes local (per application) state tracking information to a state of all keys, modifiers, and mouse buttons up. When called with `doCombineState` equal to `true`, the current global state of keys, modifiers, and mouse buttons are used in generating events.

This function is not recommended for general use because of undocumented special cases and undesirable side effects. The recommended replacement for this function is to use Quartz events and Quartz event sources. This allows you to control exactly which, if any, external event sources will contribute to the state used to create an event.

Availability

Available in Mac OS X v10.1 and later.

Declared In

`CGRemoteOperation.h`

CGEventCreate

Returns a new Quartz event.

```
CGEventRef CGEventCreate (
   CGEventSourceRef source
);
```

Parameters

source

The event source, or `NULL` to use a default source.

Return Value

A new event to be filled in, or `NULL` if the event could not be created. When you no longer need the event, you should release it using the function `CFRelease`.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CGEvent.h`

CGEventCreateCopy

Returns a copy of an existing Quartz event.

```
CGEventRef CGEventCreateCopy (
   CGEventRef event
);
```

Parameters

event

The event being copied.

Return Value

A copy of the specified event. When you no longer need the copy, you should release it using the function `CFRelease`.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CGEvent.h`

CGEventCreateData

Returns a flattened data representation of a Quartz event.

```
CFDataRef CGEventCreateData (
   CFAllocatorRef allocator,
   CGEventRef event
);
```

Parameters

allocator

The allocator to use to allocate memory for the data object. To use the current default allocator, pass `NULL` or `kCFAllocatorDefault`.

event

The event to flatten.

Return Value

The flattened data representation of the event, or `NULL` if the `event` parameter is invalid. When you no longer need the data object, you should release it using the function `CFRelease`.

Discussion

You can use this function to flatten an event for network transport to another system.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CGEvent.h`

CGEventCreateFromData

Returns a Quartz event created from a flattened data representation of the event.

```
CGEventRef CGEventCreateFromData (
   CFAllocatorRef allocator,
   CFDataRef eventData
);
```

Parameters

allocator

The allocator to use to allocate memory for the event object. To use the current default allocator, pass `NULL` or `kCFAllocatorDefault`.

eventData

The flattened data representation of the event to reconstruct.

Return Value

An event built from the flattened data representation, or `NULL` if the `eventData` parameter is invalid.

Discussion

You can use this function to reconstruct a Quartz event received by network transport from another system.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CGEvent.h`

CGEventCreateKeyboardEvent

Returns a new Quartz keyboard event.

```
CGEventRef CGEventCreateKeyboardEvent (
   CGEventSourceRef source,
   CGKeyCode virtualKey,
   bool keyDown
);
```

Parameters

source

An event source taken from another event, or `NULL`.

virtualKey

The virtual key code for the event.

keyDown

Pass `true` to specify that the key position is down. To specify that the key position is up, pass `false`. This value is used to determine the type of the keyboard event—see "Event Types" (page 62).

Return Value

A new keyboard event, or `NULL` if the event could not be created. When you no longer need the event, you should release it using the function `CFRelease`.

Discussion

All keystrokes needed to generate a character must be entered, including modifier keys. For example, to produce a 'Z', the SHIFT key must be down, the 'z' key must go down, and then the SHIFT and 'z' key must be released:

```
CGEventRef event1, event2, event3, event4;
event1 = CGEventCreateKeyboardEvent (NULL, (CGKeyCode)56, true);
event2 = CGEventCreateKeyboardEvent (NULL, (CGKeyCode)6, true);
event3 = CGEventCreateKeyboardEvent (NULL, (CGKeyCode)6, false);
event4 = CGEventCreateKeyboardEvent (NULL, (CGKeyCode)56, false);
```

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CGEvent.h`

CGEventCreateMouseEvent

Returns a new Quartz mouse event.

```
CGEventRef CGEventCreateMouseEvent (
   CGEventSourceRef source,
   CGEventType mouseType,
   CGPoint mouseCursorPosition,
   CGMouseButton mouseButton
);
```

Parameters

source

An event source taken from another event, or `NULL`.

mouseType

A mouse event type. Pass one of the constants listed in "Event Types" (page 62).

mouseCursorPosition
: The position of the mouse cursor in global coordinates.

mouseButton
: The button that's changing state. Pass one of the constants listed in "Mouse Buttons" (page 65). This parameter is ignored unless the *mouseType* parameter is `kCGEventOtherMouseDown`, `kCGEventOtherMouseDragged`, or `kCGEventOtherMouseUp`.

Return Value
A new mouse event, or `NULL` if the event could not be created. When you no longer need the event, you should release it using the function `CFRelease`.

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

Declared In
`CGEvent.h`

CGEventCreateScrollWheelEvent

Returns a new Quartz scrolling event.

```
CGEventRef CGEventCreateScrollWheelEvent (
    CGEventSourceRef source,
    CGScrollEventUnit units,
    CGWheelCount wheelCount,
    int32_t wheel1,

);
```

Parameters

source
: An event source taken from another event, or `NULL`.

units
: The unit of measurement for the scrolling event. Pass one of the constants listed in "Scrolling Event Units" (page 66).

wheelCount
: The number of scrolling devices on the mouse, up to a maximum of 3.

wheel1
: A value that reflects the movement of the primary scrolling device on the mouse. Scrolling movement is generally represented by small signed integer values, typically in a range from -10 to +10. Large values may have unexpected results, depending on the application that processes the event.

...
: Up to two values that reflect the movements of the other scrolling devices on the mouse, if any.

Return Value
A new scrolling event, or `NULL` if the event could not be created. When you no longer need the event, you should release it using the function `CFRelease`.

Discussion
This function allows you to create a scrolling event and customize the event before posting it to the event system.

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

Declared In
`CGEvent.h`

CGEventCreateSourceFromEvent

Returns a Quartz event source created from an existing Quartz event.

```
CGEventSourceRef CGEventCreateSourceFromEvent (
   CGEventRef event
);
```

Parameters

`event`
The event to access.

Return Value
An event source created from the specified event, or `NULL` if the event was generated with a private event source owned by another process. When you no longer need this event source, you should release it using the function `CFRelease`.

Discussion
Event filters may use the event source to generate events that are compatible with an event being filtered.

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

Declared In
`CGEvent.h`

CGEventGetDoubleValueField

Returns the floating-point value of a field in a Quartz event.

```
double CGEventGetDoubleValueField (
   CGEventRef event,
   CGEventField field
);
```

Parameters

`event`
The event to access.

`field`
A field in the specified event. Pass one of the constants listed in "Event Fields" (page 49).

Return Value
A floating point representation of the current value of the specified field.

Discussion
In cases where the field value is represented within the event by a fixed point number or an integer, the result is scaled to the appropriate range as part of creating the floating point representation.

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

Declared In
`CGEvent.h`

CGEventGetFlags

Returns the event flags of a Quartz event.

```
CGEventFlags CGEventGetFlags (
   CGEventRef event
);
```

Parameters

event
  The event to access.

Return Value
The current flags of the specified event. For more information, see "Event Flags" (page 57).

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

Declared In
`CGEvent.h`

CGEventGetIntegerValueField

Returns the integer value of a field in a Quartz event.

```
int64_t CGEventGetIntegerValueField (
   CGEventRef event,
   CGEventField field
);
```

Parameters

event
  The event to access.

field
  A field in the specified event. Pass one of the constants listed in "Event Fields" (page 49).

Return Value
A 64-bit integer representation of the current value of the specified field.

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

Declared In
`CGEvent.h`

CGEventGetLocation

Returns the location of a Quartz mouse event.

```
CGPoint CGEventGetLocation (
   CGEventRef event
);
```

Parameters

event

The mouse event to locate.

Return Value

The current location of the specified mouse event in global display coordinates.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CGEvent.h`

CGEventGetTimestamp

Returns the timestamp of a Quartz event.

```
CGEventTimestamp CGEventGetTimestamp (
   CGEventRef event
);
```

Parameters

event

The event to access.

Return Value

The current timestamp of the specified event.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CGEvent.h`

CGEventGetType

Returns the event type of a Quartz event (left mouse down, for example).

```
CGEventType CGEventGetType (
   CGEventRef event
);
```

Parameters

event

The event to access.

Return Value
The current event type of the specified event. The return value is one of the constants listed in "Event Types" (page 62).

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

See Also
`CGEventSetType` (page 25)

Declared In
`CGEvent.h`

CGEventGetTypeID

Returns the type identifier for the opaque type `CGEventRef`.

```
CFTypeID CGEventGetTypeID (
   void
);
```

Return Value
The Core Foundation type identifier for the opaque type `CGEventRef` (page 46).

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

Declared In
`CGEvent.h`

CGEventGetUnflippedLocation

Returns the location of a Quartz mouse event.

```
CGPoint CGEventGetUnflippedLocation (
    CGEventRef event
);
```

Parameters

`event`
The mouse event whose location you wish to obtain.

Return Value
The current location of the specified mouse event relative to the lower-left corner of the main display.

Discussion
This function returns the location of the mouse cursor associated with the event. The coordinate system used is relative to the lower-left corner of the main display, and is compatible with the global coordinate system used by the Application Kit.

Note that the y-coordinate of the returned location is off by one from an idealized coordinate system originating at the lower-left corner of the main display. Effectively, the function is defined as follows:

```
CGPoint p = CGEventGetLocation(event);
p.y = main_display_height - p.y;
```

```
/* not p.y = (main_display_height - 1) - p.y */
return p;
```

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

Declared In
`CGEvent.h`

CGEventKeyboardGetUnicodeString

Returns the Unicode string associated with a Quartz keyboard event.

```
void CGEventKeyboardGetUnicodeString (
   CGEventRef event,
   UniCharCount maxStringLength,
   UniCharCount *actualStringLength,
   UniChar unicodeString[]
);
```

Parameters

`event`
: The keyboard event to access.

`maxStringLength`
: The length of the array you provide in the `unicodeString` parameter.

`actualStringLength`
: A pointer to a `UniCharCount` variable. On return, the variable contains the actual count of Unicode characters in the event data.

`unicodeString`
: A pointer to a `UniChar` array. You are responsible for allocating storage for the array. On return, your array contains the Unicode string associated with the specified event.

Discussion
When you call this function and specify a `NULL` string or a maximum string length of 0, the function still returns the actual count of Unicode characters in the event data.

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

Declared In
`CGEvent.h`

CGEventKeyboardSetUnicodeString

Sets the Unicode string associated with a Quartz keyboard event.

```
void CGEventKeyboardSetUnicodeString (
   CGEventRef event,
   UniCharCount stringLength,
   const UniChar unicodeString[]
);
```

Parameters

event

The keyboard event to access.

stringLength

The length of the array you provide in the `unicodeString` parameter.

unicodeString

An array that contains the new Unicode string associated with the specified event.

Discussion

By default, the system translates the virtual key code in a keyboard event into a Unicode string based on the keyboard ID in the event source. This function allows you to manually override this string. Note that application frameworks may ignore the Unicode string in a keyboard event and do their own translation based on the virtual keycode and perceived event state.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CGEvent.h`

CGEventMaskBit

Generates an event mask for a single type of event.

```
CGEventMask CGEventMaskBit (
   CGEventType eventType
);
```

Parameters

eventType

An event type constant. Pass one of the constants listed in "Event Types" (page 62).

Return Value

An event mask that represents the specified event.

Discussion

This macro converts an event type constant into a mask. You can use this mask to specify that an event tap should observe the event. For more information, see `CGEventMask` (page 45).

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CGEventTypes.h`

CGEventPost

Posts a Quartz event into the event stream at a specified location.

```
void CGEventPost (
   CGEventTapLocation tap,
   CGEventRef event
);
```

Parameters

tap

The location at which to post the event. Pass one of the constants listed in "Event Tap Locations" (page 60).

event

The event to post.

Discussion

This function posts the specified event immediately before any event taps instantiated for that location, and the event passes through any such taps.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CGEvent.h`

CGEventPostToPSN

Posts a Quartz event into the event stream for a specific application.

```
void CGEventPostToPSN (
   void *processSerialNumber,
   CGEventRef event
);
```

Parameters

processSerialNumber

The process to receive the event.

event

The event to post.

Discussion

This function makes it possible for an application to establish an event routing policy, for example, by tapping events at the `kCGAnnotatedSessionEventTap` location and then posting the events to another desired process.

This function posts the specified event immediately before any event taps instantiated for the specified process, and the event passes through any such taps.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CGEvent.h`

CGEventSetDoubleValueField

Sets the floating-point value of a field in a Quartz event.

```
void CGEventSetDoubleValueField (
   CGEventRef event,
   CGEventField field,
   double value
);
```

Parameters

event

The event to access.

field

A field in the specified event. Pass one of the constants listed in "Event Fields" (page 49).

value

The new value of the specified field.

Discussion

Before calling this function, the event type must be set using a typed event creation function such as `CGEventCreateMouseEvent` (page 14), or by calling `CGEventSetType` (page 25).

In cases where the field's value is represented within the event by a fixed point number or integer, the `value` parameter is scaled as needed and converted to the appropriate type.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CGEvent.h`

CGEventSetFlags

Sets the event flags of a Quartz event.

```
void CGEventSetFlags (
   CGEventRef event,
   CGEventFlags flags
);
```

Parameters

event

The event to access.

location

The new flags of the specified event. See "Event Flags" (page 57).

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CGEvent.h`

CGEventSetIntegerValueField

Sets the integer value of a field in a Quartz event.

```
void CGEventSetIntegerValueField (
   CGEventRef event,
   CGEventField field,
   int64_t value
);
```

Parameters

event

The event to access.

field

A field in the specified event. Pass one of the constants listed in "Event Fields" (page 49).

value

The new value of the specified field.

Discussion

Before calling this function, the event type must be set using a typed event creation function such as `CGEventCreateMouseEvent` (page 14), or by calling `CGEventSetType` (page 25).

If you are creating a mouse event generated by a tablet, call this function and specify the field `kCGMouseEventSubtype` with a value of `kCGEventMouseSubtypeTabletPoint` or `kCGEventMouseSubtypeTabletProximity` before setting other parameters.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CGEvent.h`

CGEventSetLocation

Sets the location of a Quartz mouse event.

```
void CGEventSetLocation (
   CGEventRef event,
   CGPoint location
);
```

Parameters

event

The mouse event whose location to set.

location

The new location of the specified mouse event in global display coordinates.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CGEvent.h`

CGEventSetSource

Sets the event source of a Quartz event.

```
void CGEventSetSource (
   CGEventRef event,
   CGEventSourceRef source
);
```

Parameters

event

The event to access.

source

The new event source of the specified event.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CGEvent.h`

CGEventSetTimestamp

Sets the timestamp of a Quartz event.

```
void CGEventSetTimestamp (
   CGEventRef event,
   CGEventTimestamp timestamp
);
```

Parameters

event

The event to access.

timestamp

The new timestamp of the specified event.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CGEvent.h`

CGEventSetType

Sets the event type of a Quartz event (left mouse down, for example).

```
void CGEventSetType (
   CGEventRef event,
   CGEventType type
);
```

Parameters

event

The event to access.

type

The new event type of the specified event. The return value is one of the constants listed in "Event Types" (page 62).

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

See Also
`CGEventGetType` (page 18)

Declared In
`CGEvent.h`

CGEventSourceButtonState

Returns a Boolean value indicating the current button state of a Quartz event source.

```
bool CGEventSourceButtonState (
   CGEventSourceStateID sourceState,
   CGMouseButton button
);
```

Parameters

sourceState

The source state to access. Pass one of the constants listed in "Event Source States" (page 58).

button

The mouse button to test. Pass one of the constants listed in "Mouse Buttons" (page 65).

Return Value
If `true`, the button is down. If `false`, the button is up.

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

Declared In
`CGEventSource.h`

CGEventSourceCounterForEventType

Returns a count of events of a given type seen since the window server started.

```
uint32_t CGEventSourceCounterForEventType (
   CGEventSourceStateID source,
   CGEventType evType
);
```

Parameters

sourceState

The source state to access. Pass one of the constants listed in "Event Source States" (page 58).

eventType

The event type to access. To get the count of input events—keyboard, mouse, or tablet—specify `kCGAnyInputEventType`.

Return Value
The count of events of the specified type seen since the window server started.

Discussion
Quartz provides these counters for applications that monitor user activity. For example, an application could prompt a typist to take a break to reduce repetitive stress injuries.

Modifier keys produce `kCGEventFlagsChanged` events, not `kCGEventKeyDown` events, and do so both on press and release. The volume, brightness, and CD eject keys on some keyboards (both desktop and laptop) do not generate key up or key down events.

For various reasons, the number of key up and key down events may not be the same when all keyboard keys are up. As a result, a mismatch does not necessarily indicate that some keys are down.

Key autorepeat events are not counted.

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

Declared In
`CGEventSource.h`

CGEventSourceCreate

Returns a Quartz event source created with a specified source state.

```
CGEventSourceRef CGEventSourceCreate (
   CGEventSourceStateID sourceState
);
```

Parameters

`sourceState`
: The event state table to use for this event source. Pass one of the constants listed in "Event Source States" (page 58).

Return Value
A new event source, or `NULL` if the specified source state is not valid. When you no longer need the event source, you should release it using the function `CFRelease`.

Discussion
If two or more event sources are using the same source state and one of them is released, the remaining event sources will behave as if all keys and buttons on input devices are up in generating new events from this source.

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

Declared In
`CGEventSource.h`

CGEventSourceFlagsState

Returns the current flags of a Quartz event source.

```
CGEventFlags CGEventSourceFlagsState (
   CGEventSourceStateID sourceState
);
```

Parameters

sourceState

The source state to access. Pass one of the constants listed in "Event Source States" (page 58).

Return Value

The current flags of the specified event source. For more information, see "Event Flags" (page 57).

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CGEventSource.h`

CGEventSourceGetKeyboardType

Returns the keyboard type to be used with a Quartz event source.

```
CGEventSourceKeyboardType CGEventSourceGetKeyboardType (
   CGEventSourceRef source
);
```

Parameters

source

The event source to access. Pass one of the constants listed in "Event Source States" (page 58).

Return Value

The keyboard type to be used with the specified event source.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CGEventSource.h`

CGEventSourceGetLocalEventsFilterDuringSuppressionState

Returns the mask that indicates which classes of local hardware events are enabled during event suppression.

```
CGEventFilterMask CGEventSourceGetLocalEventsFilterDuringSuppressionState (
   CGEventSourceRef source,
   CGEventSuppressionState state
);
```

Parameters

source

The event source to access.

state

The type of event suppression interval during which the filter is applied. Pass one of the constants listed in "Event Suppression States" (page 60).

Return Value
A mask that specifies the categories of local hardware events to enable during the event suppression interval. See "Event Filter Masks" (page 57).

Discussion
You can configure the system to suppress local hardware events from the keyboard or mouse during a short interval after a Quartz event is posted or during a synthetic mouse drag (mouse movement with the left or only mouse button down). For information about setting this local events filter, see `CGEventSourceSetLocalEventsFilterDuringSuppressionState` (page 33).

This function lets you specify an event source and a suppression state (event suppression interval or mouse drag), and returns a filter mask of event categories to be passed through during suppression.

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

Declared In
`CGEventSource.h`

CGEventSourceGetLocalEventsSuppressionInterval

Returns the interval that local hardware events may be suppressed following the posting of a Quartz event.

```
CFTimeInterval CGEventSourceGetLocalEventsSuppressionInterval (
   CGEventSourceRef source
);
```

Parameters

`source`
The event source to access.

Discussion
By default, the system does not suppress local hardware events from the keyboard or mouse during a short interval after a Quartz event is posted. You can use the function `CGEventSourceSetLocalEventsFilterDuringSuppressionState` (page 33) to modify this behavior.

This function gets the period of time in seconds that local hardware events may be suppressed after posting a Quartz event created with the specified event source. You can use the function `CGEventSourceSetLocalEventsSuppressionInterval` (page 33) to change this time interval.

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

Declared In
`CGEventSource.h`

CGEventSourceGetPixelsPerLine

Gets the scale of pixels per line in a scrolling event source.

```
double CGEventSourceGetPixelsPerLine (
    CGEventSourceRef source
);
```

Parameters

source

The event source to access.

Return Value

The scale of pixels per line in a scrolling event.

Discussion

This function returns the scale of pixels per line in the specified event source. For example, if the scale in the event source is 10.5 pixels per line, this function would return 10.5. Every scrolling event can be interpreted to be scrolling by pixel or by line. By default, the scale is about ten pixels per line. You can alter the scale with the function `CGEventSourceSetPixelsPerLine`.

Availability

Available in Mac OS X v10.5 and later.

See Also

`CGEventSourceSetPixelsPerLine` (page 34)

Declared In

`CGEventSource.h`

CGEventSourceGetSourceStateID

Returns the source state associated with a Quartz event source.

```
CGEventSourceStateID CGEventSourceGetSourceStateID (
   CGEventSourceRef source
);
```

Parameters

source

The event source to access.

Return Value

The source state associated with the specified event source.

Discussion

This function returns the ID of the source state table associated with an event source.

For event sources created with the `kCGEventSourceStatePrivate` source state, this function returns the ID of the private source state table created for the event source. This unique ID may be passed to the `CGEventSourceCreate` function to create a second event source sharing the same state table. This may be useful, for example, in creating separate mouse and keyboard sources which share a common private state.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CGEventSource.h`

CGEventSourceGetTypeID

Returns the type identifier for the opaque type `CGEventSourceRef`.

```
CFTypeID CGEventSourceGetTypeID (
   void
);
```

Return Value
The Core Foundation type identifier for the opaque type `CGEventSourceRef` (page 46).

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

Declared In
`CGEventSource.h`

CGEventSourceGetUserData

Returns the 64-bit user-specified data for a Quartz event source.

```
int64_t CGEventSourceGetUserData (
   CGEventSourceRef source
);
```

Parameters

`source`
The event source to access.

Return Value
The user-specified data.

Discussion
Each input event includes 64 bits of user-specified data. This function gets the user-specified data for all events created by the specified event source. This data may also be obtained per event using the `CGEventGetIntegerValueField` (page 17) function.

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

Declared In
`CGEventSource.h`

CGEventSourceKeyState

Returns a Boolean value indicating the current keyboard state of a Quartz event source.

```
bool CGEventSourceKeyState (
   CGEventSourceStateID sourceState,
   CGKeyCode key
);
```

Parameters

`sourceState`
The source state to access. Pass one of the constants listed in "Event Source States" (page 58).

key

The virtual key code to test.

Return Value

If `true`, the key is down. If `false`, the key is up.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CGEventSource.h`

CGEventSourceSecondsSinceLastEventType

Returns the elapsed time since the last event for a Quartz event source.

```
CFTimeInterval CGEventSourceSecondsSinceLastEventType (
   CGEventSourceStateID source,
   CGEventType eventType
);
```

Parameters

source

The source state to access. Pass one of the constants listed in "Event Source States" (page 58).

eventType

The event type to access. To get the elapsed time since the previous input event—keyboard, mouse, or tablet—specify `kCGAnyInputEventType`.

Return Value

The time in seconds since the previous input event of the specified type.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CGEventSource.h`

CGEventSourceSetKeyboardType

Sets the keyboard type to be used with a Quartz event source.

```
void CGEventSourceSetKeyboardType (
   CGEventSourceRef source,
   CGEventSourceKeyboardType keyboardType
);
```

Parameters

source

The event source to access.

keyboardType

The keyboard type to be used with the specified event source.

Availability

Available in Mac OS X v10.4 and later.

Declared In
`CGEventSource.h`

CGEventSourceSetLocalEventsFilterDuringSuppressionState

Sets the mask that indicates which classes of local hardware events are enabled during event suppression.

```
void CGEventSourceSetLocalEventsFilterDuringSuppressionState (
   CGEventSourceRef source,
   CGEventFilterMask filter,
   CGEventSuppressionState state
);
```

Parameters

`source`
: The event source to access.

`filter`
: A mask that specifies the categories of local hardware events to enable during the event suppression interval. See "Event Filter Masks" (page 57).

`state`
: The type of event suppression interval during which the filter is applied. Pass one of the constants listed in "Event Suppression States" (page 60).

Discussion
By default, the system does not suppress local hardware events from the keyboard or mouse during a short interval after a Quartz event is posted—see `CGEventSourceSetLocalEventsSuppressionInterval` (page 33)—and during a synthetic mouse drag (mouse movement with the left or only mouse button down).

Some applications may want to disable events from some of the local hardware during this interval. For example, if you post mouse events only, you may wish to suppress local mouse events and permit local keyboard events to pass through. This function lets you specify an event source, a suppression state (event suppression interval or mouse drag), and a filter mask of event classes to be passed through. The new local events filter takes effect with the next Quartz event you post using this event source.

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

Declared In
`CGEventSource.h`

CGEventSourceSetLocalEventsSuppressionInterval

Sets the interval that local hardware events may be suppressed following the posting of a Quartz event.

```
void CGEventSourceSetLocalEventsSuppressionInterval (
   CGEventSourceRef source,
   CFTimeInterval seconds
);
```

Parameters

`source`
: The event source to access.

seconds

The period of time in seconds that local hardware events (keyboard or mouse) are suppressed after posting a Quartz event created with the specified event source. The value should be a number in the range [0.0, 10.0].

Discussion

By default, the system does not suppress local hardware events from the keyboard or mouse during a short interval after a Quartz event is posted. You can use the function `CGEventSourceSetLocalEventsFilterDuringSuppressionState` (page 33) to modify this behavior.

This function sets the period of time in seconds that local hardware events may be suppressed after posting a Quartz event created with the specified event source. The default suppression interval is 0.25 seconds.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CGEventSource.h`

CGEventSourceSetPixelsPerLine

Sets the scale of pixels per line in a scrolling event source.

```
void CGEventSourceSetPixelsPerLine (
    CGEventSourceRef source,
    double pixelsPerLine
);
```

Parameters

source

The event source to access.

pixelsPerLine

The scale of pixels per line in the specified event source.

Discussion

This function sets the scale of pixels per line in the specified event source. For example, if you pass the value 12.0 in the *pixelsPerLine* parameter, the scale of pixels per line in the event source would be changed to 12.0. Every scrolling event can be interpreted to be scrolling by pixel or by line. By default, the scale is about ten pixels per line. You can retrieve the scale with the function `CGEventSourceGetPixelsPerLine`.

Availability

Available in Mac OS X v10.5 and later.

See Also

`CGEventSourceGetPixelsPerLine` (page 29)

Declared In

`CGEventSource.h`

CGEventSourceSetUserData

Sets the 64-bit user-specified data for a Quartz event source.

```
void CGEventSourceSetUserData (
   CGEventSourceRef source,
   int64_t userData
);
```

Parameters

source
: The event source to access.

userData
: The user-specified data. For example, you could specify a vendor hardware ID.

Discussion

Each input event includes 64 bits of user-specified data. This function sets the user-specified data for all events created by the specified event source. This data may also be set per event using the `CGEventGetIntegerValueField` (page 17) function.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CGEventSource.h`

CGEventTapCreate

Creates an event tap.

```
CFMachPortRef CGEventTapCreate (
   CGEventTapLocation tap,
   CGEventTapPlacement place,
   CGEventTapOptions options,
   CGEventMask eventsOfInterest,
   CGEventTapCallBack callback,
   void *refcon
);
```

Parameters

tap
: The location of the new event tap. Pass one of the constants listed in "Event Tap Locations" (page 60). Only processes running as the root user may locate an event tap at the point where HID events enter the window server; for other users, this function returns `NULL`.

place
: The placement of the new event tap in the list of active event taps. Pass one of the constants listed in "Event Tap Placement" (page 61).

options
: A constant that specifies whether the new event tap is a passive listener or an active filter.

eventsOfInterest
: A bit mask that specifies the set of events to be observed. For a list of possible events, see "Event Types" (page 62). For information on how to specify the mask, see `CGEventMask` (page 45). If the event tap is not permitted to monitor one or more of the events specified in the `eventsOfInterest` parameter, then the appropriate bits in the mask are cleared. If that action results in an empty mask, this function returns `NULL`.

callback

An event tap callback function that you provide. Your callback function is invoked from the run loop to which the event tap is added as a source. The thread safety of the callback is defined by the run loop's environment. To learn more about event tap callbacks, see `CGEventTapCallBack` (page 44).

refcon

A pointer to user-defined data. This pointer is passed into the callback function specified in the `callback` parameter.

Return Value

A Core Foundation mach port that represents the new event tap, or `NULL` if the event tap could not be created. When you are finished using the event tap, you should release the mach port using the function `CFRelease`. Releasing the mach port also releases the tap.

Discussion

Event taps receive key up and key down events if one of the following conditions is true:

- The current process is running as the root user.
- Access for assistive devices is enabled. In Mac OS X v10.4, you can enable this feature using System Preferences, Universal Access panel, Keyboard view.

After creating an event tap, you can add it to a run loop as follows:

1. Pass the event tap to the `CFMachPortCreateRunLoopSource` function to create a run loop event source.
2. Call the `CFRunLoopAddSource` function to add the source to the appropriate run loop.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CGEvent.h`

CGEventTapCreateForPSN

Creates an event tap for a specified process.

```
CFMachPortRef CGEventTapCreateForPSN (
   void *processSerialNumber,
   CGEventTapPlacement place,
   CGEventTapOptions options,
   CGEventMask eventsOfInterest,
   CGEventTapCallBack callback,
   void *refcon
);
```

Parameters

processSerialNumber

The process to monitor.

place

The placement of the new event tap in the list of active event taps. Pass one of the constants listed in "Event Tap Placement" (page 61).

options
: A constant that specifies whether the new event tap is a passive listener or an active filter.

eventsOfInterest
: A bit mask that specifies the set of events to be observed. For a list of possible events, see "Event Types" (page 62). For information on how to specify the mask, see `CGEventMask` (page 45). If the event tap is not permitted to monitor one or more of the events specified in the `eventsOfInterest` parameter, then the appropriate bits in the mask are cleared. If that action results in an empty mask, this function returns `NULL`.

callback
: An event tap callback function that you provide. Your callback function is invoked from the run loop to which the event tap is added as a source. The thread safety of the callback is defined by the run loop's environment. To learn more about event tap callbacks, see `CGEventTapCallBack` (page 44).

refcon
: A pointer to user-defined data. This pointer is passed into the callback function specified in the `callback` parameter.

Return Value
A Core Foundation mach port that represents the new event tap, or `NULL` if the event tap could not be created. When you are finished using the event tap, you should release the mach port using the function `CFRelease`. Releasing the mach port also releases the tap.

Discussion
This function creates an event tap that receives events being routed by the window server to the specified process. For more information about creating event taps, see `CGEventTapCreate` (page 35).

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

Declared In
`CGEvent.h`

CGEventTapEnable

Enables or disables an event tap.

```
void CGEventTapEnable (
   CFMachPortRef myTap,
   bool enable
);
```

Parameters

myTap
: The event tap to enable or disable.

enable
: Pass `true` to enable the event tap. To disable it, pass `false`.

Discussion
Event taps are normally enabled when created. If an event tap becomes unresponsive, or if a user requests that event taps be disabled, then a `kCGEventTapDisabled` event is passed to the event tap callback function. Event taps may be re-enabled by calling this function.

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

Declared In
`CGEvent.h`

CGEventTapIsEnabled

Returns a Boolean value indicating whether an event tap is enabled.

```
bool CGEventTapIsEnabled (
   CFMachPortRef myTap
);
```

Parameters

`myTap`
: The event tap to test.

Return Value
If `true`, the specified event tap is enabled; otherwise, `false`.

Discussion
For more information, see the function `CGEventTapEnable` (page 37).

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

Declared In
`CGEvent.h`

CGEventTapPostEvent

Posts a Quartz event from an event tap into the event stream.

```
void CGEventTapPostEvent (
   CGEventTapProxy proxy,
   CGEventRef event
);
```

Parameters

`proxy`
: A proxy that identifies the event tap posting the event. Your event tap callback function is passed this proxy when it is invoked.

`event`
: The event to post.

Discussion
You can use this function to post a new event at the same point to which an event returned from an event tap callback function would be posted. The new event enters the system before the event returned by the callback enters the system. Events posted into the system will be seen by all taps placed after the tap posting the event.

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

Declared In
`CGEvent.h`

CGGetEventTapList

Gets a list of currently installed event taps.

```
CGError CGGetEventTapList (
   CGTableCount maxNumberOfTaps,
   CGEventTapInformation tapList[],
   CGTableCount *eventTapCount
);
```

Parameters

maxNumberOfTaps

The length of the array you provide in the `tapList` parameter.

tapList

An array of event tap information structures. You are responsible for allocating storage for the array. On return, your array contains a list of currently installed event taps. If you pass `NULL` in this parameter, the `maxNumberOfTaps` parameter is ignored, and the `eventTapCount` variable is filled in with the number of event taps that are currently installed.

eventTapCount

A pointer to a `CGTableCount` variable. On return, the variable contains actual number of array elements filled in.

Return Value
A result code. See the result codes described in *Quartz Display Services Reference*.

Discussion
Each call to this function has the side effect of resetting the minimum and maximum latency values in the `tapList` parameter to the corresponding average values. Values reported in these fields reflect the minimum and maximum values seen since the preceding call, or the instantiation of the tap. This allows a monitoring tool to evaluate the best and worst case latency over time and under various operating conditions.

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

Declared In
`CGEvent.h`

CGInhibitLocalEvents

Turns off local hardware events in the current session.

```
CGError CGInhibitLocalEvents (
   boolean_t doInhibit
);
```

Parameters

doInhibit

Pass `true` to specify that local hardware events on the remote system should be inhibited; otherwise, pass `false`.

Return Value
A result code. See the result codes described in *Quartz Display Services Reference*.

Discussion

This function is typically used during remote operation of a system to disconnect the keyboard and mouse for a short period of time, as in automated system testing or telecommuting applications.

The `CGInhibitLocalEvents` function is not recommended for general use because of undocumented special cases and undesirable side effects. For example, this function can permanently disable the keyboard and mouse, rendering the system unusable. The recommended replacement for this function is `CGEventSourceSetLocalEventsFilterDuringSuppressionState` (page 33).

Special Considerations

In Mac OS X v10.2 and earlier, this function inhibits local events only after a synthetic keyboard or mouse event is posted by the calling application. In Mac OS X v10.3 and later, event inhibition takes effect immediately. If your application terminates for any reason, event inhibition on the remote system is immediately turned off.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CGRemoteOperation.h`

CGPostKeyboardEvent

Synthesizes a low-level keyboard event on the local machine.

```
CGError CGPostKeyboardEvent (
   CGCharCode keyChar,
   CGKeyCode virtualKey,
   boolean_t keyDown
);
```

Parameters

keyChar

The value of the character to generate, or 0 to specify that the system should guess an appropriate value based on the default key mapping.

virtualKey

The virtual key code for the event. See `CGKeyCode` (page 49).

keyDown

Pass `true` to specify that the key position is down; otherwise, pass `false`.

Return Value

A result code. See the result codes described in *Quartz Display Services Reference*.

Discussion

This function is not recommended for general use because of undocumented special cases and undesirable side effects. The recommended replacement for this function is `CGEventCreateKeyboardEvent` (page 13), which allows you to create a keyboard event and customize the event before posting it to the event system.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CGRemoteOperation.h`

CGPostMouseEvent

Synthesizes a low-level mouse-button event on the local machine.

```
CGError CGPostMouseEvent (
   CGPoint mouseCursorPosition,
   boolean_t updateMouseCursorPosition,
   CGButtonCount buttonCount,
   boolean_t mouseButtonDown,
   ...
);
```

Parameters

mouseCursorPosition
: The new coordinates of the mouse in global display space.

updateMouseCursorPosition
: Pass `true` if the on-screen cursor should be moved to the location specified in the `mouseCursorPosition` parameter; otherwise, pass `false`.

buttonCount
: The number of mouse buttons, up to a maximum of 32.

mouseButtonDown
: Pass `true` to specify that the primary or left mouse button is down; otherwise, pass `false`.

...
: Zero or more Boolean values that specify whether the remaining mouse buttons are down (`true`) or up (`false`). The second value, if any, should specify the state of the secondary mouse button (right). A third value would specify the state of the center button, and the remaining buttons would be in USB device order.

Return Value

A result code. See the result codes described in *Quartz Display Services Reference*.

Discussion

Based on the arguments you pass to this function, the function generates the appropriate mouse-down, mouse-up, mouse-move, or mouse-drag events by comparing the new state with the current state.

This function is not recommended for general use because of undocumented special cases and undesirable side effects. The recommended replacement for this function is `CGEventCreateMouseEvent` (page 14), which allows you to create a mouse event and customize the event before posting it to the event system.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CGRemoteOperation.h`

CGPostScrollWheelEvent

Synthesizes a low-level scrolling event on the local machine.

```
CGError CGPostScrollWheelEvent (
   CGWheelCount wheelCount,
   int32_t wheel1,
   ...
);
```

Parameters

wheelCount
: The number of scrolling devices, up to a maximum of 3.

wheel1
: A value that reflects the movement of the primary scrolling device on the mouse.

...
: Up to two values that reflect the movements of the other scrolling devices on the mouse (if any).

Return Value

A result code. See the result codes described in *Quartz Display Services Reference*.

Discussion

Scrolling movement is generally represented by small signed integer values, typically in a range from -10 to +10. Large values may have unexpected results, depending on the application that processes the event.

This function is not recommended for general use because of undocumented special cases and undesirable side effects. The recommended replacement for this function is `CGEventCreateScrollWheelEvent` (page 15), which allows you to create a scrolling event and customize the event before posting it to the event system.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`CGRemoteOperation.h`

CGSetLocalEventsFilterDuringSuppressionState

Filters local hardware events from the keyboard and mouse during the short interval after a synthetic event is posted.

```
CGError CGSetLocalEventsFilterDuringSuppressionState (
   CGEventFilterMask filter,
   CGEventSuppressionState state
);
```

Parameters

filter
: The class of local hardware events to enable after a synthetic event is posted. Pass one of the constants listed in "Event Filter Masks" (page 57).

state
: The type of interval during which the filter is applied. Pass one of the constants listed in "Event Suppression States" (page 60).

Return Value

A result code. See the result codes described in *Quartz Display Services Reference*.

Discussion
By default, the system suppresses local hardware events from the keyboard and mouse during a short interval after a synthetic event is posted and during a synthetic mouse drag (mouse movement with the left or only mouse button down).

Some applications may want to enable events from some of the local hardware. For example, if you post mouse events only, you may wish to permit local keyboard hardware events to pass through.

This function lets you specify a state (event suppression interval or mouse drag), and a mask of event categories to be passed through. The new filter state takes effect with the next synthetic event you post.

This function is not recommended for general use because of undocumented special cases and undesirable side effects. The recommended replacement for this function is `CGEventSourceSetLocalEventsFilterDuringSuppressionState` (page 33), which allows the filter behavior to be associated only with events created from a specific event source.

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

Declared In
`CGRemoteOperation.h`

CGSetLocalEventsSuppressionInterval

Sets the time interval in seconds that local hardware events are suppressed after posting a synthetic event.

```
CGError CGSetLocalEventsSuppressionInterval (
   CFTimeInterval seconds
);
```

Parameters

seconds

The desired time interval in seconds. The value should be a number in the range [0.0, 10.0].

Return Value
A result code. If the `seconds` parameter is outside the allowed range, returns `kCGErrorRangeCheck`.

Discussion
This function determines how long local events matching an event filter are to be suppressed following the posting of a synthetic event. The default time interval for event suppression is 0.25 seconds.

This function is not recommended for general use because of undocumented special cases and undesirable side effects. The recommended replacement for this function is `CGEventSourceSetLocalEventsSuppressionInterval` (page 33), which allows the suppression interval to be adjusted for a specific event source, affecting only events posted using that event source.

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

Declared In
`CGRemoteOperation.h`

Callbacks

CGEventTapCallBack

A client-supplied callback function that's invoked whenever an associated event tap receives a Quartz event.

```
typedef CGEventRef (*CGEventTapCallBack) (
   CGEventTapProxy proxy,
   CGEventType type,
   CGEventRef event,
   void *refcon
);
```

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

```
CGEventRef MyEventTapCallBack (
   CGEventTapProxy proxy,
   CGEventType type,
   CGEventRef event,
   void *refcon
);
```

Parameters

proxy

A proxy for the event tap. See `CGEventTapProxy` (page 48). This callback function may pass this proxy to other functions such as the event-posting routines.

type

The event type of this event. See "Event Types" (page 62).

event

The incoming event. This event is owned by the caller, and you do not need to release it.

refcon

A pointer to user-defined data. You specify this pointer when you create the event tap. Several different event taps could use the same callback function, each tap with its own user-defined data.

Discussion

If the event tap is an active filter, your callback function should return one of the following:

- The (possibly modified) event that is passed in. This event is passed back to the event system.
- A newly-constructed event. After the new event has been passed back to the event system, the new event will be released along with the original event.
- `NULL` if the event passed in is to be deleted.

If the event tap is an passive listener, your callback function may return the event that is passed in, or `NULL`. In either case, the event stream is not affected.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`CGEventTypes.h`

Data Types

CGButtonCount

Represents the number of buttons being set in a synthetic mouse event.

```
typedef uint32_t CGButtonCount;
```

Discussion
In mouse events, the button count parameter ranges from 0 to 31. See the function `CGPostMouseEvent` (page 41).

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

Declared In
`CGRemoteOperation.h`

CGCharCode

Represents a character generated by pressing one or more keys on a keyboard.

```
typedef uint16_t CGCharCode;
```

Discussion
This data type represents a 16-bit character code. Values of this type may or may not correspond to UTF-16 character codes. See the function `CGPostKeyboardEvent` (page 40).

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

Declared In
`CGRemoteOperation.h`

CGEventMask

Defines a mask that identifies the set of Quartz events to be observed in an event tap.

```
typedef uint64_t CGEventMask;
```

Discussion
When you call either `CGEventTapCreate` (page 35) or `CGEventTapCreateForPSN` (page 36) to register an event tap, you supply a bit mask that identifies the set of events to be observed. You specify each event using one of the event type constants listed in "Event Types" (page 62). To form the bit mask, use the `CGEventMaskBit` macro to convert each constant into an event mask and then OR the individual masks together. For example:

```
CGEventMask mask = CGEventMaskBit(kCGEventLeftMouseDown) |
                   CGEventMaskBit(kCGEventLeftMouseUp);
```

You can also supply a mask to observe all events:

```
CGEventMask mask = kCGEventMaskForAllEvents;
```

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

Declared In
`CGEventTypes.h`

CGEventRef

Defines an opaque type that represents a low-level hardware event.

```
typedef struct __CGEvent *CGEventRef;
```

Discussion
Low-level hardware events of this type are referred to as Quartz events. A typical event in Mac OS X originates when the user manipulates an input device such as a mouse or a keyboard. The device driver associated with that device, through the I/O Kit, creates a low-level event, puts it in the window server's event queue, and notifies the window server. The window server creates a Quartz event, annotates the event, and dispatches the event to the appropriate run-loop port of the target process. There the event is picked up by the Carbon Event Manager and forwarded to the event-handling mechanism appropriate to the application environment. You can use event taps to gain access to Quartz events at several different steps in this process.

This opaque type is derived from CFType and inherits the properties that all Core Foundation types have in common. For more information, see *CFType Reference*.

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

Declared In
`CGEventTypes.h`

CGEventSourceKeyboardType

Defines a code that represents the type of keyboard used with a specified event source.

```
typedef uint32_t CGEventSourceKeyboardType;
```

Discussion
This code is the same keyboard type identifier used with the `UCKeyTranslate` function to drive keyboard translation.

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

Declared In
`CGEventTypes.h`

CGEventSourceRef

Defines an opaque type that represents the source of a Quartz event.

```
typedef struct __CGEventSource * CGEventSourceRef;
```

Discussion
A Quartz event source is an object that contains accumulated state related to event generation and event posting. Every event source has an associated global event state table called a source state. When you call `CGEventSourceCreate` (page 27) to create an event source, you specify which source state to use. For more information about source states, see "Event Source States" (page 58).

A typical use of an event source would be to obtain the source from a Quartz event received by an event tap callback function, and then to use that source for any new events created as a result of the received event. This has the effect of marking the events as being related.

This opaque type is derived from CFType and inherits the properties that all Core Foundation types have in common. For more information, see *CFType Reference*.

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

Declared In
`CGEventTypes.h`

CGEventTapInformation

Defines the structure used to report information about event taps.

```
typedef struct CGEventTapInformation
{
    uint32_t            eventTapID;
    CGEventTapLocation  tapPoint;
    CGEventTapOptions   options;
    CGEventMask         eventsOfInterest;
    pid_t               tappingProcess;
    pid_t               processBeingTapped;
    bool                enabled;
    float               minUsecLatency;
    float               avgUsecLatency;
    float               maxUsecLatency;
} CGEventTapInformation;
```

Fields

`eventTapID`
The unique identifier for the event tap.

`tapPoint`
The location of the event tap. See "Event Tap Locations" (page 60).

`options`
The type of event tap (passive listener or active filter).

`eventsOfInterest`
The mask that identifies the set of events to be observed.

`tappingProcess`
The process ID of the application that created the event tap.

`processBeingTapped`
: The process ID of the target application (non-zero only if the event tap was created using the function `CGEventTapCreateForPSN` (page 36).

`enabled`
: `TRUE` if the event tap is currently enabled; otherwise `FALSE`.

`minUsecLatency`
: Minimum latency in microseconds. In this data structure, **latency** is defined as the time in microseconds it takes for an event tap to process and respond to an event passed to it.

`avgUsecLatency`
: Average latency in microseconds. This is a weighted average that gives greater weight to more recent events.

`maxUsecLatency`
: Maximum latency in microseconds.

Discussion
To learn how to obtain information about event taps, see the function `CGGetEventTapList` (page 39).

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

Declared In
`CGEventTypes.h`

CGEventTapProxy

Defines an opaque type that represents state within the client application that's associated with an event tap.

```
typedef struct __CGEventTapProxy * CGEventTapProxy;
```

Discussion
An event tap proxy object is passed to your event tap callback function when it receives a new Quartz event. Your callback function needs the proxy to post Quartz events using the function `CGEventTapPostEvent` (page 38).

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

Declared In
`CGEventTypes.h`

CGEventTimestamp

Defines the elapsed time in nanoseconds since startup that a Quartz event occurred.

```
typedef uint64_t CGEventTimestamp;
```

Discussion
An event timestamp is a big, unsigned, 64-bit number. That's big, really big. You just won't believe how vastly, hugely, mind-bogglingly big it is. You may think your application has been running for a long time, but that's just peanuts to an event timestamp.

For information about how event timestamps are used, see the functions `CGEventGetTimestamp` (page 18) and `CGEventSetTimestamp` (page 25).

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

Declared In
`CGEventTypes.h`

CGKeyCode

Represents the virtual key codes used in keyboard events.

```
typedef uint16_t CGKeyCode;
```

Discussion
In Mac OS X, the hardware scan codes generated by keyboards are mapped to a set of virtual key codes that are hardware-independent. Pressing a given key always generates the same virtual key code on any supported keyboard.

As keys are pressed, the system uses the virtual key codes to create low-level keyboard events. For information on how to simulate a keyboard event, see the function `CGEventCreateKeyboardEvent` (page 13).

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

Declared In
`CGRemoteOperation.h`

CGWheelCount

Represents the number of wheels being set in a scroll wheel event.

```
typedef uint32_t CGWheelCount;
```

Discussion
See the function `CGPostScrollWheelEvent` (page 41).

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

Declared In
`CGRemoteOperation.h`

Constants

Event Fields

Constants used as keys to access specialized fields in low-level events.

```
enum _CGEventField {
    kCGMouseEventNumber = 0,
    kCGMouseEventClickState = 1,
    kCGMouseEventPressure = 2,
    kCGMouseEventButtonNumber = 3,
    kCGMouseEventDeltaX = 4,
    kCGMouseEventDeltaY = 5,
    kCGMouseEventInstantMouser = 6,
    kCGMouseEventSubtype = 7,
    kCGKeyboardEventAutorepeat = 8,
    kCGKeyboardEventKeycode = 9,
    kCGKeyboardEventKeyboardType = 10,
    kCGScrollWheelEventDeltaAxis1 = 11,
    kCGScrollWheelEventDeltaAxis2 = 12,
    kCGScrollWheelEventDeltaAxis3 = 13,
    kCGScrollWheelEventFixedPtDeltaAxis1 = 93,
    kCGScrollWheelEventFixedPtDeltaAxis2 = 94,
    kCGScrollWheelEventFixedPtDeltaAxis3 = 95,
    kCGScrollWheelEventPointDeltaAxis1 = 96,
    kCGScrollWheelEventPointDeltaAxis2 = 97,
    kCGScrollWheelEventPointDeltaAxis3 = 98,
    kCGScrollWheelEventInstantMouser = 14,
    kCGTabletEventPointX = 15,
    kCGTabletEventPointY = 16,
    kCGTabletEventPointZ = 17,
    kCGTabletEventPointButtons = 18,
    kCGTabletEventPointPressure = 19,
    kCGTabletEventTiltX = 20,
    kCGTabletEventTiltY = 21,
    kCGTabletEventRotation = 22,
    kCGTabletEventTangentialPressure = 23,
    kCGTabletEventDeviceID = 24,
    kCGTabletEventVendor1 = 25,
    kCGTabletEventVendor2 = 26,
    kCGTabletEventVendor3 = 27,
    kCGTabletProximityEventVendorID = 28,
    kCGTabletProximityEventTabletID = 29,
    kCGTabletProximityEventPointerID = 30,
    kCGTabletProximityEventDeviceID = 31,
    kCGTabletProximityEventSystemTabletID = 32,
    kCGTabletProximityEventVendorPointerType = 33,
    kCGTabletProximityEventVendorPointerSerialNumber = 34,
    kCGTabletProximityEventVendorUniqueID = 35,
    kCGTabletProximityEventCapabilityMask = 36,
    kCGTabletProximityEventPointerType = 37,
    kCGTabletProximityEventEnterProximity = 38,
    kCGEventTargetProcessSerialNumber = 39,
    kCGEventTargetUnixProcessID = 40,
    kCGEventSourceUnixProcessID = 41,
    kCGEventSourceUserData = 42,
    kCGEventSourceUserID = 43,
    kCGEventSourceGroupID = 44,
    kCGEventSourceStateID = 45,
    kCGScrollWheelEventIsContinuous = 88
};
typedef uint32_t CGEventField;
```

Constants

`kCGMouseEventNumber`
: Key to access an integer field that contains the mouse button event number. Matching mouse-down and mouse-up events will have the same event number.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGMouseEventClickState`
: Key to access an integer field that contains the mouse button click state. A click state of 1 represents a single click. A click state of 2 represents a double-click. A click state of 3 represents a triple-click.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGMouseEventPressure`
: Key to access a double field that contains the mouse button pressure. The pressure value may range from 0 to 1, with 0 representing the mouse being up. This value is commonly set by tablet pens mimicking a mouse.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGMouseEventButtonNumber`
: Key to access an integer field that contains the mouse button number. For information about the possible values, see "Mouse Buttons" (page 65).

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGMouseEventDeltaX`
: Key to access an integer field that contains the horizontal mouse delta since the last mouse movement event.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGMouseEventDeltaY`
: Key to access an integer field that contains the vertical mouse delta since the last mouse movement event.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGMouseEventInstantMouser`
: Key to access an integer field. The value is non-zero if the event should be ignored by the Inkwell subsystem.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGMouseEventSubtype`
: Key to access an integer field that encodes the mouse event subtype as a `kCFNumberIntType`.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGKeyboardEventAutorepeat`
: Key to access an integer field, non-zero when this is an autorepeat of a key-down, and zero otherwise.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGKeyboardEventKeycode`
: Key to access an integer field that contains the virtual keycode of the key-down or key-up event.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGKeyboardEventKeyboardType`
: Key to access an integer field that contains the keyboard type identifier.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGScrollWheelEventDeltaAxis1`
: Key to access an integer field that contains scrolling data. This field typically contains the change in vertical position since the last scrolling event from a Mighty Mouse scroller or a single-wheel mouse scroller.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGScrollWheelEventDeltaAxis2`
: Key to access an integer field that contains scrolling data. This field typically contains the change in horizontal position since the last scrolling event from a Mighty Mouse scroller.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGScrollWheelEventDeltaAxis3`
: This field is not used.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGScrollWheelEventFixedPtDeltaAxis1`
: Key to access a field that contains scrolling data. The scrolling data represents a line-based or pixel-based change in vertical position since the last scrolling event from a Mighty Mouse scroller or a single-wheel mouse scroller. The scrolling data uses a fixed-point 16.16 signed integer format. For example, if the field contains a value of 1.0, the integer 0x00010000 is returned by `CGEventGetIntegerValueField`. If this key is passed to `CGEventGetDoubleValueField`, the fixed-point value is converted to a double value.

  Available in Mac OS X v10.5 and later.

  Declared in `CGEventTypes.h`.

`kCGScrollWheelEventFixedPtDeltaAxis2`
: Key to access a field that contains scrolling data. The scrolling data represents a line-based or pixel-based change in horizontal position since the last scrolling event from a Mighty Mouse scroller. The scrolling data uses a fixed-point 16.16 signed integer format. For example, if the field contains a value of 1.0, the integer 0x00010000 is returned by `CGEventGetIntegerValueField`. If this key is passed to `CGEventGetDoubleValueField`, the fixed-point value is converted to a double value.

  Available in Mac OS X v10.5 and later.

  Declared in `CGEventTypes.h`.

`kCGScrollWheelEventFixedPtDeltaAxis3`
: This field is not used.

  Available in Mac OS X v10.5 and later.

  Declared in `CGEventTypes.h`.

`kCGScrollWheelEventPointDeltaAxis1`
Key to access an integer field that contains pixel-based scrolling data. The scrolling data represents the change in vertical position since the last scrolling event from a Mighty Mouse scroller or a single-wheel mouse scroller.

Available in Mac OS X v10.5 and later.

Declared in `CGEventTypes.h`.

`kCGScrollWheelEventPointDeltaAxis2`
Key to access an integer field that contains pixel-based scrolling data. The scrolling data represents the change in horizontal position since the last scrolling event from a Mighty Mouse scroller.

Available in Mac OS X v10.5 and later.

Declared in `CGEventTypes.h`.

`kCGScrollWheelEventPointDeltaAxis3`
This field is not used.

Available in Mac OS X v10.5 and later.

Declared in `CGEventTypes.h`.

`kCGScrollWheelEventInstantMouser`
Key to access an integer field that indicates whether the event should be ignored by the Inkwell subsystem. If the value is non-zero, the event should be ignored.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGTabletEventPointX`
Key to access an integer field that contains the absolute X coordinate in tablet space at full tablet resolution.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGTabletEventPointY`
Key to access an integer field that contains the absolute Y coordinate in tablet space at full tablet resolution.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGTabletEventPointZ`
Key to access an integer field that contains the absolute Z coordinate in tablet space at full tablet resolution.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGTabletEventPointButtons`
Key to access an integer field that contains the tablet button state. Bit 0 is the first button, and a set bit represents a closed or pressed button. Up to 16 buttons are supported.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGTabletEventPointPressure`
: Key to access a double field that contains the tablet pen pressure. A value of 0.0 represents no pressure, and 1.0 represents maximum pressure.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGTabletEventTiltX`
: Key to access a double field that contains the horizontal tablet pen tilt. A value of 0.0 represents no tilt, and 1.0 represents maximum tilt.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGTabletEventTiltY`
: Key to access a double field that contains the vertical tablet pen tilt. A value of 0.0 represents no tilt, and 1.0 represents maximum tilt.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGTabletEventRotation`
: Key to access a double field that contains the tablet pen rotation.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGTabletEventTangentialPressure`
: Key to access a double field that contains the tangential pressure on the device. A value of 0.0 represents no pressure, and 1.0 represents maximum pressure.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGTabletEventDeviceID`
: Key to access an integer field that contains the system-assigned unique device ID.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGTabletEventVendor1`
: Key to access an integer field that contains a vendor-specified value.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGTabletEventVendor2`
: Key to access an integer field that contains a vendor-specified value.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGTabletEventVendor3`
: Key to access an integer field that contains a vendor-specified value.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGTabletProximityEventVendorID`
: Key to access an integer field that contains the vendor-defined ID, typically the USB vendor ID.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGTabletProximityEventTabletID`
: Key to access an integer field that contains the vendor-defined tablet ID, typically the USB product ID.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGTabletProximityEventPointerID`
: Key to access an integer field that contains the vendor-defined ID of the pointing device.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGTabletProximityEventDeviceID`
: Key to access an integer field that contains the system-assigned device ID.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGTabletProximityEventSystemTabletID`
: Key to access an integer field that contains the system-assigned unique tablet ID.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGTabletProximityEventVendorPointerType`
: Key to access an integer field that contains the vendor-assigned pointer type.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGTabletProximityEventVendorPointerSerialNumber`
: Key to access an integer field that contains the vendor-defined pointer serial number.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGTabletProximityEventVendorUniqueID`
: Key to access an integer field that contains the vendor-defined unique ID.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGTabletProximityEventCapabilityMask`
: Key to access an integer field that contains the device capabilities mask.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGTabletProximityEventPointerType`
: Key to access an integer field that contains the pointer type.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGTabletProximityEventEnterProximity`
: Key to access an integer field that indicates whether the pen is in proximity to the tablet. The value is non-zero if the pen is in proximity to the tablet and zero when leaving the tablet.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGEventTargetProcessSerialNumber`
: Key to access a field that contains the event target process serial number. The value is a 64-bit long word.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGEventTargetUnixProcessID`
: Key to access a field that contains the event target Unix process ID.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGEventSourceUnixProcessID`
: Key to access a field that contains the event source Unix process ID.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGEventSourceUserData`
: Key to access a field that contains the event source user-supplied data, up to 64 bits.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGEventSourceUserID`
: Key to access a field that contains the event source Unix effective UID.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGEventSourceGroupID`
: Key to access a field that contains the event source Unix effective GID.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGEventSourceStateID`
: Key to access a field that contains the event source state ID used to create this event.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGScrollWheelEventIsContinuous`
: Key to access an integer field that indicates whether a scrolling event contains continuous, pixel-based scrolling data. The value is non-zero when the scrolling data is pixel-based and zero when the scrolling data is line-based.

  Available in Mac OS X v10.5 and later.

  Declared in `CGEventTypes.h`.

Discussion

These constants are used as keys to access certain specialized event fields when using low-level accessor functions such as `CGEventGetIntegerValueField` (page 17), `CGEventSetIntegerValueField` (page 24), `CGEventGetDoubleValueField` (page 16), and `CGEventSetDoubleValueField` (page 23).

Event Filter Masks

Specify masks for classes of low-level events that can be filtered during event suppression states.

```
enum CGEventFilterMask {
    kCGEventFilterMaskPermitLocalMouseEvents = 0x00000001,
    kCGEventFilterMaskPermitLocalKeyboardEvents = 0x00000002,
    kCGEventFilterMaskPermitSystemDefinedEvents = 0x00000004,
    kCGEventFilterMaskPermitAllEvents = kCGEventFilterMaskPermitLocalMouseEvents
 | kCGEventFilterMaskPermitLocalKeyboardEvents |
kCGEventFilterMaskPermitSystemDefinedEvents
};
typedef uint32_t CGEventFilterMask;
```

Event Flags

Constants that indicate the modifier key state at the time an event is created, as well as other event-related states.

```
enum _CGEventFlags {
    kCGEventFlagMaskAlphaShift =    NX_ALPHASHIFTMASK,
    kCGEventFlagMaskShift =         NX_SHIFTMASK,
    kCGEventFlagMaskControl =       NX_CONTROLMASK,
    kCGEventFlagMaskAlternate =     NX_ALTERNATEMASK,
    kCGEventFlagMaskCommand =       NX_COMMANDMASK,
    kCGEventFlagMaskHelp =          NX_HELPMASK,
    kCGEventFlagMaskSecondaryFn =   NX_SECONDARYFNMASK,
    kCGEventFlagMaskNumericPad =    NX_NUMERICPADMASK,
    kCGEventFlagMaskNonCoalesced =  NX_NONCOALSESCEDMASK
};
typedef uint64_t CGEventFlags;
```

Constants

`kCGEventFlagMaskAlphaShift`

Indicates that the Caps Lock key is down for a keyboard, mouse, or flag-changed event.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGEventFlagMaskShift`

Indicates that the Shift key is down for a keyboard, mouse, or flag-changed event.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGEventFlagMaskControl`

Indicates that the Control key is down for a keyboard, mouse, or flag-changed event.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGEventFlagMaskAlternate`

Indicates that the Alt or Option key is down for a keyboard, mouse, or flag-changed event.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGEventFlagMaskCommand`
: Indicates that the Command key is down for a keyboard, mouse, or flag-changed event.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGEventFlagMaskHelp`
: Indicates that the Help modifier key is down for a keyboard, mouse, or flag-changed event. This key is not present on most keyboards, and is different than the Help key found in the same row as Home and Page Up.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGEventFlagMaskSecondaryFn`
: Indicates that the Fn (Function) key is down for a keyboard, mouse, or flag-changed event. This key is found primarily on laptop keyboards.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGEventFlagMaskNumericPad`
: Identifies key events from the numeric keypad area on extended keyboards.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGEventFlagMaskNonCoalesced`
: Indicates that mouse and pen movement events are not being coalesced.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

Discussion

These constants specify masks for the bits in an event flags bit mask. Event flags indicate the modifier key state at the time an event is created, as well as other event-related states. Event flags are used in accessor functions such as `CGEventGetFlags` (page 17), `CGEventSetFlags` (page 23), and `CGEventSourceFlagsState` (page 27).

Event Source States

Constants that specify the possible source states of an event source.

```
enum {
    kCGEventSourceStatePrivate = -1,
    kCGEventSourceStateCombinedSessionState = 0,
    kCGEventSourceStateHIDSystemState = 1
};
typedef uint32_t CGEventSourceStateID;
```

Constants

`kCGEventSourceStatePrivate`
: Specifies that an event source should use a private event state table.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGEventSourceStateCombinedSessionState`
: Specifies that an event source should use the event state table that reflects the combined state of all event sources posting to the current user login session.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGEventSourceStateHIDSystemState`
: Specifies that an event source should use the event state table that reflects the combined state of all hardware event sources posting from the HID system.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

Discussion

A source state refers to a global event state table. These tables contain accumulated information on modifier flag state, keyboard key state, mouse button state, and related internal parameters placed in effect by posting events with associated sources.

Two pre-existing event state tables are defined:

- The `kCGEventSourceStateCombinedSessionState` table reflects the combined state of all event sources posting to the current user login session. If your program is posting events from within a login session, you should use this source state when you create an event source.
- The `kCGEventSourceStateHIDSystemState` table reflects the combined state of all hardware event sources posting from the HID system. If your program is a daemon or a user space device driver interpreting hardware state and generating events, you should use this source state when you create an event source.

Specialized applications such as remote control programs may want to generate and track event source state independent of other processes. These programs should use the `kCGEventSourceStatePrivate` value in creating their event source. An independent state table and unique source state ID (`CGEventSourceStateID`) are created to track the event source's state. This independent state table is owned by the creating event source and released with it.

Event Source Token

Specifies any input event type.

```
#define kCGAnyInputEventType ((CGEventType)(~0))
```

Constants

`kCGAnyInputEventType`
: A constant that specifies any input event type.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

Discussion

This constant is typically used with the function `CGEventSourceSecondsSinceLastEventType` (page 32) to specify that you want the elapsed time since the last input event of any type.

Event Suppression States

Specify the event suppression states that can occur after posting an event.

```
enum CGEventSuppressionState {
    kCGEventSuppressionStateSuppressionInterval = 0,
    kCGEventSuppressionStateRemoteMouseDrag = 1,
    kCGNumberOfEventSuppressionStates = 2
};
typedef uint32_t CGEventSuppressionState;
```

Constants

`kCGEventSuppressionStateSuppressionInterval`

Specifies that certain local hardware events may be suppressed for a short interval after posting an event.

Available in Mac OS X v10.3 and later.

Declared in `CGRemoteOperation.h`.

`kCGEventSuppressionStateRemoteMouseDrag`

Specifies that certain local hardware events may be suppressed during a mouse drag operation (mouse movement with the left or only mouse button down).

Available in Mac OS X v10.3 and later.

Declared in `CGRemoteOperation.h`.

Discussion

These constants specify the types of event suppression intervals during which an event filter is applied after posting an event.

Event Tap Locations

Constants that specify possible tapping points for events.

```
enum _CGEventTapLocation {
    kCGHIDEventTap = 0,
    kCGSessionEventTap,
    kCGAnnotatedSessionEventTap
};
typedef uint32_t CGEventTapLocation;
```

Constants

`kCGHIDEventTap`

Specifies that an event tap is placed at the point where HID system events enter the window server.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGSessionEventTap`

Specifies that an event tap is placed at the point where HID system and remote control events enter a login session.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGAnnotatedSessionEventTap`

Specifies that an event tap is placed at the point where session events have been annotated to flow to an application.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

Discussion

In addition to the three tapping points described above, an event tap may also be placed where annotated events are delivered to a specific application. For more information, see the function `CGEventTapCreateForPSN` (page 36).

Event Tap Options

Constants that specify whether a new event tap is an active filter or a passive listener.

```
enum _CGEventTapOptions {
    kCGEventTapOptionDefault = 0x00000000,
    kCGEventTapOptionListenOnly = 0x00000001
};
typedef uint32_t CGEventTapOptions;
```

Constants

`kCGEventTapOptionDefault`

Specifies that a new event tap is an active filter. (Applications targeting Mac OS X v10.4 should use the literal value to create an active filter event tap, as this constant was omitted from the header.)

Available in Mac OS X v10.5 and later.

Declared in `CGEventTypes.h`.

`kCGEventTapOptionListenOnly`

Specifies that a new event tap is a passive listener.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

Discussion

When you create an event tap, you indicate whether it is a passive listener or active event filter. A passive listener receives events but cannot modify or divert them. An active filter may pass an event through unmodified, modify an event, or discard an event.

Event Tap Placement

Constants that specify where a new event tap is inserted into the list of active event taps.

```
enum _CGEventTapPlacement {
    kCGHeadInsertEventTap = 0,
    kCGTailAppendEventTap
};
typedef uint32_t CGEventTapPlacement;
```

Constants

`kCGHeadInsertEventTap`

Specifies that a new event tap should be inserted before any pre-existing event taps at the same location.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGTailAppendEventTap`

Specifies that a new event tap should be inserted after any pre-existing event taps at the same location.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

Discussion

Event taps may be inserted at a specified location at the head of pre-existing filters, or appended after any pre-existing filters.

Event Types

Constants that specify the different types of input events.

```
enum _CGEventType {
    kCGEventNull                = NX_NULLEVENT,
    kCGEventLeftMouseDown       = NX_LMOUSEDOWN,
    kCGEventLeftMouseUp         = NX_LMOUSEUP,
    kCGEventRightMouseDown      = NX_RMOUSEDOWN,
    kCGEventRightMouseUp        = NX_RMOUSEUP,
    kCGEventMouseMoved          = NX_MOUSEMOVED,
    kCGEventLeftMouseDragged    = NX_LMOUSEDRAGGED,
    kCGEventRightMouseDragged   = NX_RMOUSEDRAGGED,
    kCGEventKeyDown             = NX_KEYDOWN,
    kCGEventKeyUp               = NX_KEYUP,
    kCGEventFlagsChanged        = NX_FLAGSCHANGED,
    kCGEventScrollWheel         = NX_SCROLLWHEELMOVED,
    kCGEventTabletPointer       = NX_TABLETPOINTER,
    kCGEventTabletProximity     = NX_TABLETPROXIMITY,
    kCGEventOtherMouseDown      = NX_OMOUSEDOWN,
    kCGEventOtherMouseUp        = NX_OMOUSEUP,
    kCGEventOtherMouseDragged   = NX_OMOUSEDRAGGED,
    kCGEventTapDisabledByTimeout = 0xFFFFFFFE,
    kCGEventTapDisabledByUserInput = 0xFFFFFFFF
};
typedef uint32_t CGEventType;
```

Constants

`kCGEventNull`

Specifies a null event.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGEventLeftMouseDown`
: Specifies a mouse down event with the left button.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGEventLeftMouseUp`
: Specifies a mouse up event with the left button.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGEventRightMouseDown`
: Specifies a mouse down event with the right button.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGEventRightMouseUp`
: Specifies a mouse up event with the right button.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGEventMouseMoved`
: Specifies a mouse moved event.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGEventLeftMouseDragged`
: Specifies a mouse drag event with the left button down.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGEventRightMouseDragged`
: Specifies a mouse drag event with the right button down.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGEventKeyDown`
: Specifies a key down event.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGEventKeyUp`
: Specifies a key up event.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGEventFlagsChanged`
: Specifies a key changed event for a modifier or status key.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGEventScrollWheel`
: Specifies a scroll wheel moved event.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGEventTabletPointer`
: Specifies a tablet pointer event.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGEventTabletProximity`
: Specifies a tablet proximity event.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGEventOtherMouseDown`
: Specifies a mouse down event with one of buttons 2-31.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGEventOtherMouseUp`
: Specifies a mouse up event with one of buttons 2-31.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGEventOtherMouseDragged`
: Specifies a mouse drag event with one of buttons 2-31 down.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGEventTapDisabledByTimeout`
: Specifies an event indicating the event tap is disabled because of timeout.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

`kCGEventTapDisabledByUserInput`
: Specifies an event indicating the event tap is disabled because of user input.

  Available in Mac OS X v10.4 and later.

  Declared in `CGEventTypes.h`.

Discussion
These constants are used:

- In the functions `CGEventTapCreate` (page 35) and `CGEventTapCreateForPSN` (page 36) to specify the events of interest for the new event tap.
- To indicate the event type passed to your event tap callback function.
- In the function `CGEventCreateMouseEvent` (page 14) to specify the type of mouse event.
- In the functions `CGEventGetType` (page 18) and `CGEventSetType` (page 25) to identify the event type.
- In the functions `CGEventSourceCounterForEventType` (page 26) and `CGEventSourceSecondsSinceLastEventType` (page 32) to indicate the event type.

Note that tablet devices may generate mouse events with embedded tablet data, or tablet pointer and proximity events. Tablet mouse events allow tablets to be used with applications that are not tablet-aware.

Event Type Mask

Specifies an event mask that represents all event types.

```
#define kCGEventMaskForAllEvents (~(CGEventMask)0)
```

Constants

`kCGEventMaskForAllEvents`

An event mask that specifies all event types.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

Discussion

This constant is typically used with the functions `CGEventTapCreate` (page 35) and `CGEventTapCreateForPSN` (page 36) to register an event tap that observes all input events.

Mouse Buttons

Constants that specify buttons on a one, two, or three-button mouse.

```
enum _CGMouseButton {
    kCGMouseButtonLeft = 0,
    kCGMouseButtonRight = 1,
    kCGMouseButtonCenter = 2
};
typedef uint32_t CGMouseButton;
```

Constants

`kCGMouseButtonLeft`

Specifies the only mouse button on a one-button mouse, or the left mouse button on a two-button or three-button mouse.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGMouseButtonRight`

Specifies the right mouse button on a two-button or three-button mouse.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGMouseButtonCenter`

Specifies the center mouse button on a three-button mouse.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

Discussion

Quartz supports up to 32 mouse buttons. The first three buttons are specified using these three constants. Additional buttons are specified in USB order using the integers 3 to 31.

These constants are used:

- In the function `CGEventCreateMouseEvent` (page 14) to specify the button that's changing state.
- In the function `CGEventSourceButtonState` (page 26) to specify the button that's being tested.
- To specify the value of the `kCGMouseEventButtonNumber` event field when modifying an event.

Mouse Subtypes

Constants used with the `kCGMouseEventSubtype` event field.

```
enum _CGEventMouseSubtype {
    kCGEventMouseSubtypeDefault = 0,
    kCGEventMouseSubtypeTabletPoint = 1,
    kCGEventMouseSubtypeTabletProximity = 2
};
typedef uint32_t CGEventMouseSubtype;
```

Constants

`kCGEventMouseSubtypeDefault`

Specifies that the event is an ordinary mouse event, and does not contain additional tablet device information.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGEventMouseSubtypeTabletPoint`

Specifies that the mouse event originated from a tablet device, and that the various `kCGTabletEvent` field selectors may be used to obtain tablet-specific data from the mouse event.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

`kCGEventMouseSubtypeTabletProximity`

Specifies that the mouse event originated from a tablet device with the pen in proximity but not necessarily touching the tablet, and that the various `kCGTabletProximity` field selectors may be used to obtain tablet-specific data from the mouse event. This is often used with mouse move events originating from a tablet.

Available in Mac OS X v10.4 and later.

Declared in `CGEventTypes.h`.

Discussion

Tablets may generate specially annotated mouse events that contain values associated with the `kCGMouseEventSubtype` event field. To learn how to set these values, see the function `CGEventSetIntegerValueField` (page 24).

Scrolling Event Units

Constants that specify the unit of measurement for a scrolling event.

```
enum {
    kCGScrollEventUnitPixel = 0,
    kCGScrollEventUnitLine = 1
};
typedef uint32_t CGScrollEventUnit;
```

Constants

`kCGScrollEventUnitPixel`

Specifies that the unit of measurement is pixels.

Available in Mac OS X v10.5 and later.

Declared in `CGEventTypes.h`.

`kCGScrollEventUnitLine`

Specifies that the unit of measurement is lines.

Available in Mac OS X v10.5 and later.

Declared in `CGEventTypes.h`.

Discussion

You may pass one of these constants to the function `CGEventCreateScrollWheelEvent` (page 15) to specify the unit of measurement for the event. The constant `kCGScrollEventUnitPixel` produces an event that most applications interpret as a smooth scrolling event. By default, the scale is about ten pixels per line. You can alter the scale with the function `CGEventSourceSetPixelsPerLine` (page 34).

Deprecated Quartz Event Services Functions

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

Deprecated in Mac OS X v10.4

CGEventGetSource

Returns a Quartz event source created from an existing Quartz event. (Deprecated in Mac OS X v10.4. Use `CGEventCreateSourceFromEvent` (page 16) instead.)

```
CGEventSourceRef CGEventGetSource (
    CGEventRef event
);
```

Availability
Available in Mac OS X v10.4 through Mac OS X v10.4.

Deprecated in Mac OS X v10.4.

Declared In
`CGEvent.h`

Document Revision History

This table describes the changes to *Quartz Event Services Reference*.

Date	**Notes**
2007-10-31	Updated for Mac OS X v10.5.
2006-10-03	Updated the description of the CGEventMask data type. Added the CGEventMaskBit function.
2006-04-04	New document that describes the C API for event taps, filters used to observe and alter the stream of low-level user input events.

Index