Event Manager Reference

(Not Recommended)

Carbon > Events & Other Input

2007-10-31

Contents

Tables

Event Manager Reference (Not Recommended)

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

Overview

Important: The Event Manager is a legacy System 7 technology. You should use the Mac OS X Carbon Event Manager instead. See *Carbon Event Manager Programming Guide*.

The Event Manager is a legacy System 7 technology that was created to support the cooperative, multitasking environment available on Macintosh computers at the time. This environment allowed users to switch between many open applications and allows other applications to receive background processing time.

The Carbon Event Manager, introduced in Mac OS X, offers a simple yet flexible approach to event handling that greatly reduces the amount of code needed to write a basic application. Moreover, the Carbon Event Manager's streamlined event handling enhances system performance on Mac OS X through more efficient allocation of processing time. Applications that use the Carbon Event Manager not only run better on Mac OS X, they help improve overall performance and responsiveness.

Since the introduction of the Macintosh computer, Mac applications have used the Event Manager to receive information about actions performed by the user, to receive notices of changes in their processing, and to communicate with other applications. For example, an application can retrieve information from the Event Manager about whether the user has pressed a key or the mouse button, whether one of the application's windows needs updating, or whether some other hardware-related or software-related action requires a response from the application.

Applications also used the Event Manager to support the cooperative, multitasking environment available on versions of the Mac OS that preceded Mac OS X. This environment allows users to switch between many open applications and allows other applications to receive background processing time. By using Event Manager routines, an application allowed the system software to coordinate the scheduling of processing time between it and other applications.

Carbon supports the majority of the Event Manager.

High-level events APIs (as contained in `EPPC.h`) are not supported. You should use Apple events instead.

Carbon does not support the `diskEvt` event. Support for volume mount and unmount events will be available in the Carbon Event Manager.

Carbon does not set the `convertClipboardFlag` in the `EventRecord` to indicate that the scrap has changed while the application was suspended. You should call the Scrap Manager function `GetCurrentScrap` instead.

Low-level event queue functions, such as `GetEvQHdr` and `PPostEvent`, are no longer supported.

Application-defined function-key procedures are not supported in Carbon.

Functions by Task

Accessors for Low-Memory Globals

`LMGetKeyRepThresh` (page 17)
: Returns the low-memory auto-key rate.

`LMGetKeyThresh` (page 17)
: Returns the low-memory auto-key threshold.

`LMSetKeyRepThresh` (page 19)
: Sets the low-memory auto-key rate.

`LMSetKeyThresh` (page 19)
: Sets the low-memory auto-key threshold.

`LMGetKbdType` (page 17)
: Returns a value that specifies the physical keyboard type.

`LMGetKbdLast` (page 16)
: Returns a value that specifies the last physical keyboard type used.

`LMSetKbdLast` (page 18)
: Sets a value that specifies the last physical keyboard type used.

`LMSetKbdType` (page 18)
: Sets the keyboard type.

Getting Timing Information

`GetCaretTime` (page 12)
: Obtains the suggested difference in ticks that should exist between blinks of the caret (usually a vertical bar marking the insertion point) in editable text.

`GetDblTime` (page 13)
: Determines whether a sequence of mouse events constitutes a double click.

Making Keyboard Settings

`KeyScript` (page 41) Deprecated in Mac OS X v10.5
: Changes the keyboard script (the script system used for keyboard input), changes the keyboard layout (the mapping of keys to characters) or input method within the current keyboard script (a facility for entering 2-byte characters), or makes a setting related to text input, using the supplied value.

Reading the Keyboard

`GetKeys` (page 14)
    Obtains the current state of the keyboard.

`KeyTranslate` (page 15)
    Converts a virtual key code to a character code based on a `'KCHR'` resource.

`GetCurrentKeyModifiers` (page 12)
    Returns the current state of the keyboard modifier keys.

`IsCmdChar` (page 15)
    Tests whether the Command key is pressed in conjunction with another key (or keys) that could generate the specified test character.

Receiving Events

`EventAvail` (page 10)
    Retrieves the next available event from the Event Manager without removing the returned event from your application's event stream.

`FlushEvents` (page 11)
    Removes low-level events from the Operating System event queue.

`GetNextEvent` (page 14)
    Retrieves events one at a time from the Event Manager.

`SetEventMask` (page 20)
    Sets the system event mask of your application to the specified mask.

`WaitNextEvent` (page 21)
    Retrieves events one at a time from the Event Manager.

Sending Events

`PostEvent` (page 19)
    Posts events into the Operating System event queue.

Miscellaneous

`CheckEventQueueForUserCancel` (page 10)
    Checks the event queue for an user-cancel event.

`GetGlobalMouse` (page 13)
    Obtains the position of the mouse, in global coordinates.

Functions

CheckEventQueueForUserCancel

Checks the event queue for an user-cancel event.

```
Boolean CheckEventQueueForUserCancel (
   void
);
```

Return Value
Returns `true` if a user-cancel event is in the queue, `false` otherwise.

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

Declared In
`CarbonEvents.h`

EventAvail

Retrieves the next available event from the Event Manager without removing the returned event from your application's event stream.

```
Boolean EventAvail (
   EventMask eventMask,
   EventRecord *theEvent
);
```

Parameters

eventMask
A value that indicates which kinds of events are to be returned; this parameter is interpreted as a sum of event mask constants. You specify the event mask using one or more of the values defined by the "Event Mask Constants" (page 34). If no event of any of the designated types is available, `EventAvail` returns a null event.

theEvent
A pointer to an event structure for the next available event of the specified type or types. The `EventAvail` function does not remove the returned event from the event stream, but does return the information about the event in an event structure. The event structure includes the type of event received and other information.

Return Value
`EventAvail` returns `false` as its function result if the event being returned is a null event; otherwise, `EventAvail` returns `true`.

Special Considerations

If `EventAvail` returns a low-level event from the Operating System event queue, the event will not be accessible later if, in the meantime, the event queue becomes full and the event is discarded from it; however, this is not a common occurrence.

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

Not available to 64-bit applications.

Related Sample Code
Simple DrawSprocket

Declared In
`Events.h`

FlushEvents

Removes low-level events from the Operating System event queue.

```
void FlushEvents (
   EventMask whichMask,
   EventMask stopMask
);
```

Parameters

`whichMask`
: A value that indicates which kinds of low-level events are to be removed from the Operating System event queue; this parameter is interpreted as a sum of event mask constants. You specify the event mask using one or more of the values defined in "Event Mask Constants" (page 34). The `whichMask` and `stopMask` parameters together specify which events to remove.

`stopMask`
: A value that limits which low-level events are to be removed from the Operating System event queue; this parameter is interpreted as a sum of event mask constants. You specify the event mask using one or more of the values defined in "Event Mask Constants" (page 34). `FlushEvents` does not remove any low-level events that are specified by the `stopMask` parameter. To remove all events specified by the `whichMask` parameter, specify 0 as the `stopMask` parameter.

Discussion
`FlushEvents` removes only low-level events stored in the Operating System event queue; it does not remove activate, update, operating-system, or high-level events. `FlushEvents` does not remove any types of events not stored in the Operating System event queue.

You can choose to use the `FlushEvents` function when your application first starts to empty the Operating System event queue of any keystrokes or mouse events generated by the user while the Finder loaded your application. In general, however, your application should not empty the queue at any other time as this loses user actions and makes your application and the computer appear unresponsive to the user.

You specify which low-level events to remove using the `whichMask` and `stopMask` parameters. `FlushEvents` removes the low-level events specified by the `whichMask` parameter, up to but not including the first event of any type specified by the `stopMask` parameter.

If the event queue doesn't contain any of the events specified by the `whichMask` parameter, `FlushEvents` does not remove any events from the queue.

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

Related Sample Code
HideMenuBar

ictbSample

Declared In
`Events.h`

GetCaretTime

Obtains the suggested difference in ticks that should exist between blinks of the caret (usually a vertical bar marking the insertion point) in editable text.

```
UInt32 GetCaretTime (
   void
);
```

Return Value
The blink delay, in ticks.

Discussion
If your application supports editable text, your application should use the value returned by `GetCaretTime` to determine how often to blink the caret. If your application uses only TextEdit, you can use TextEdit functions to automatically blink the caret at the time interval that the user specifies in the General Controls panel.

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

Declared In
`Events.h`

GetCurrentKeyModifiers

Returns the current state of the keyboard modifier keys.

```
UInt32 GetCurrentKeyModifiers (
   void
);
```

Parameters

Return Value
A bit mask indicating which keyboard modifier keys are pressed. See "Event Modifier Constants" (page 27) for a list of possible values.

Discussion
`GetCurrentKeyModifiers` provides a more convenient way to get the modifier key state than calling `GetNextEvent`. It returns a value whose individual bits indicate which keyboard modifier keys are currently being pressed. You can test for the Caps Lock, Shift, Control, Option, and Command keys.

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

Declared In
`CarbonEventsCore.h`

GetDblTime

Determines whether a sequence of mouse events constitutes a double click.

```
UInt32 GetDblTime (
   void
);
```

Return Value
The suggested maximum elapsed time, in ticks, between a mouse-up event and a mouse-down event.

Discussion
The `GetDblTime` function returns the suggested maximum elapsed time, in ticks, between a mouse-up event and a mouse-down event. The user can adjust this value using the Mouse control panel.

If your application distinguishes a double click of the mouse from a single click, your application should use the value returned by `GetDblTime` to make this distinction. If your application uses TextEdit, the TextEdit functions automatically recognize and handle double clicks of text within a TextEdit edit structure by appropriately highlighting or unhighlighting the selection.

The ratio of ticks to value in the `DoubleTime` global variable is 1:1. However, the Finder multiplies `DoubleTime` by 2 to determine double click time because it needs to account for user problems that typically occur during icon arrangement. Therefore, the Finder uses `DoubleTime*2` whereas the rest of the system uses `DoubleTime`.

Incidentally, the Finder does not limit the `DoubleTime` to 64 ticks. In most places, it treats it like a byte although in some others it treats it like a longword. The best method would be to provide a one-second double-byte (two seconds in the Finder).

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

Declared In
`Events.h`

GetGlobalMouse

Obtains the position of the mouse, in global coordinates.

```
void GetGlobalMouse (
   Point *globalMouse
);
```

Parameters

`globalMouse`
The position of the mouse, as a global point.

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

Declared In
`CarbonEventsCore.h`

GetKeys

Obtains the current state of the keyboard.

```
void GetKeys (
   KeyMap theKeys
);
```

Parameters

theKeys

On output, the current state of the keyboard, including the keypad, if any.

Discussion

You can use the `GetKeys` function to determine the current state of the keyboard at any time. For example, you can determine whether one of the modifier keys is down by itself or in combination with another key using the `GetKeys` function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Events.h`

GetNextEvent

Retrieves events one at a time from the Event Manager.

```
Boolean GetNextEvent (
   EventMask eventMask,
   EventRecord *theEvent
);
```

Parameters

eventMask

A value that indicates which kinds of events are to be returned; this parameter is interpreted as a sum of event mask constants. You specify the event mask using one or more of the values defined in "Event Mask Constants" (page 34). If no event of any of the designated types is available, `GetNextEvent` returns a null event.

theEvent

A pointer to an event structure for the next available event of the specified type or types. The `GetNextEvent` function removes the returned event from the event stream and returns the information about the event in an event structure. The event structure includes the type of event received and other information.

Return Value

Discussion

`GetNextEvent` returns `false` as its function result if the event being returned is a null event or if `GetNextEvent` has intercepted the event; otherwise, `GetNextEvent` returns `true`. The `GetNextEvent` function calls the Operating System Manager function `SystemEvent` to determine whether the event should be handled by the application or the Operating System.

The `GetNextEvent` function also intercepts Command–Shift–number key sequences and calls the corresponding `'FKEY'` resource to perform the associated action. The Event Manager's processing of Command–Shift–number key sequences with numbers 3 through 9 can be disabled by setting the `ScrDmpEnable` global variable (a byte) to 0.

Special Considerations

For greater support of the multitasking environment, your application should use `WaitNextEvent` instead of `GetNextEvent` whenever possible.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Events.h`

IsCmdChar

Tests whether the Command key is pressed in conjunction with another key (or keys) that could generate the specified test character.

```
Boolean IsCmdChar (
   const EventRecord *event,
   short test
);
```

Parameters

`event`

The event record for a key-down or auto-key event with the Command key down.

`test`

The character you want to test.

Return Value

The function returns `TRUE` if the test character is produced with the current modifier keys, or if it would be produced by changing the current modifier key bits in either or both of the following ways: (1) turning the Command bit off or (2) toggling the Shift bit.

Discussion

This function tests whether the Command key is pressed in conjunction with another key (or keys) that could generate the `test` character for some combination of Command up or down and Shift up or down. This accommodates European keyboards that may have the `test` character as a shifted character, and non-Roman keyboards that will only generate the `test` character if the Command key is pressed. It's most useful for testing for Command-period, but it can test for command-AnyCharacter.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Events.h`

KeyTranslate

Converts a virtual key code to a character code based on a `'KCHR'` resource.

```
UInt32 KeyTranslate (
   const void *transData,
   UInt16 keycode,
   UInt32 *state
);
```

Parameters

transData

A pointer to the 'KCHR' resource that you want the `KeyTranslate` function to use when converting the key code to a character code.

keycode

A 16-bit value that your application should set so that bits 0–6 contain the virtual key code and bit 7 contains either 1 to indicate an up stroke or 0 to indicate a down stroke of the key. Bits 8–15 have the same interpretation as the high byte of the `modifiers` field of the event structure and should be set according to the needs of your application.

state

A pointer to a value that your application should set to 0 the first time it calls `KeyTranslate` or any time your application calls `KeyTranslate` with a different `'KCHR'` resource. Thereafter, your application should pass the same value in the `state` parameter as `KeyTranslate` returned in the previous call.

Return Value

A 32-bit value that gives the character code for the virtual key code specified by the `keycode` parameter.

Discussion

The `KeyTranslate` function returns the values that correspond to one or possibly two characters that are generated by the specified virtual key code. The following diagram shows the structure of the 32-bit number that `KeyTranslate` returns.

```
31             24 23             16 15             8 7               0
-----------------------------------------------------------------
|               |               |               |               |
-----------------------------------------------------------------
    Reserved 1    Character code 1    Reserved 2    Character code 2
```

For example, a given virtual key code might correspond to an alphabetic character with a separate accent character. When the user presses Option-E followed by E, you can map this through the `KeyTranslate` function using the U.S. `'KCHR'` resource to produce é, which `KeyTranslate` returns as two characters in the bytes labeled Character code 1 and Character code 2.

If `KeyTranslate` returns only one character code, it is always in the byte labeled Character code 2. However, your application should always check both bytes labeled Character code 1 and Character code 2 for possible values that map to the virtual key code.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Events.h`

LMGetKbdLast

Returns a value that specifies the last physical keyboard type used.

```
UInt8 LMGetKbdLast (
   void
);
```

Return Value
The last physical keyboard type used.

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

Declared In
`Events.h`

LMGetKbdType

Returns a value that specifies the physical keyboard type.

```
UInt8 LMGetKbdType (
   void
);
```

Return Value
The physical keyboard type.

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

Declared In
`Events.h`

LMGetKeyRepThresh

Returns the low-memory auto-key rate.

```
SInt16 LMGetKeyRepThresh (
   void
);
```

Return Value
The auto-key rate, that is, the amount of time, in ticks, that must elapse before the Event Manager generates a subsequent auto-key event.

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

Declared In
`Events.h`

LMGetKeyThresh

Returns the low-memory auto-key threshold.

```
SInt16 LMGetKeyThresh (
   void
);
```

Return Value
Returns the auto-key threshold, that is, the amount of time, in ticks, that must elapse before the Event Manager generates an auto-key event.

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

Declared In
`Events.h`

LMSetKbdLast

Sets a value that specifies the last physical keyboard type used.

```
void LMSetKbdLast (
   UInt8 value
);
```

Parameters

`value`
    The physical keyboard type you want to set.

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

Declared In
`Events.h`

LMSetKbdType

Sets the keyboard type.

```
void LMSetKbdType (
   UInt8 value
);
```

Parameters

`value`
    The physical keyboard type you want to set.

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

Declared In
`Events.h`

LMSetKeyRepThresh

Sets the low-memory auto-key rate.

```
void LMSetKeyRepThresh (
   SInt16 value
);
```

Parameters

value

The low-memory auto-key rate you want to set.

Discussion

`LMSetKeyRepThresh` sets the low-memory auto-key rate, that is, the amount of time, in ticks, that must elapse before the Event Manager generates a subsequent auto-key event.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Events.h`

LMSetKeyThresh

Sets the low-memory auto-key threshold.

```
void LMSetKeyThresh (
   SInt16 value
);
```

Parameters

value

The low-memory auto-key threshold you want to set.

Discussion

`LMSetKeyThresh` sets the low-memory auto-key threshold, that is, the amount of time, in ticks, that must elapse before the Event Manager generates an auto-key event.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Events.h`

PostEvent

Posts events into the Operating System event queue.

```
OSErr PostEvent (
   EventKind eventNum,
   UInt32 eventMsg
);
```

Parameters

`eventNum`

A value that indicates the type of event to post into the Operating System event queue. You specify the event kind using one or more of these values defined in "Event Mask Constants" (page 34): `mouseDown`, `mouseUp`, `keyDown`, `keyUp`, `autoKey`, and `diskEvt`. Do not attempt to post any other type of event in the Operating System event queue.

`eventMsg`

An unsigned integer that contains the contents of the `message` field for the event that `PostEvent` should post in the queue.

Return Value

A result code. See "Event Manager Result Codes" (page 40).

Discussion

In the `eventNum` and `eventMsg` parameters, you specify the value for the `what` and `message` fields of the event's event structure. The `PostEvent` function fills out the `when`, `where`, and `modifiers` fields of the event structure with the current time, current mouse location, and current state of the modifier keys and mouse button.

The `PostEvent` function posts only events that are enabled by the system event mask. If the event queue is full, `PostEvent` removes the oldest event in the queue and posts the new event.

Note that if you use `PostEvent` to repost an event, the `PostEvent` function fills out the `when`, `where`, and `modifier` fields of the event structure, giving these fields of the reposted event different values from the values contained in the original event.

Do not post any events other than mouse-down, mouse-up, key-down, key-up, auto-key, and disk-inserted events in the Operating System event queue. Attempting to post other events into the Operating System event queue interferes with the internal operation of the Event Manager.

In most cases, your application should not call the `PostEvent` function.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Events.h`

SetEventMask

Sets the system event mask of your application to the specified mask.

```
void SetEventMask (
   EventMask value
);
```

Parameters

value

A value that specifies which events should be posted in the Operating System event queue. You specify the event mask using one or more of the values defined in "Event Mask Constants" (page 34).

Discussion

Your application should not call the `SetEventMask` function to disable any event types from being posted. Use `SetEventMask` only to enable key-up events if your application needs to respond to key-up events.

The `SetEventMask` function sets the system event mask of your application according to the parameter `value`. The Operating System Event Manager posts only low-level events (other than update or activate events) corresponding to bits in the system event mask of the current process when posting events in the Operating System event queue. The system event mask of an application is initially set to post mouse-up, mouse-down, key-down, auto-key, and disk-inserted events into the Operating System event queue.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Events.h`

WaitNextEvent

Retrieves events one at a time from the Event Manager.

```
Boolean WaitNextEvent (
   EventMask eventMask,
   EventRecord *theEvent,
   UInt32 sleep,
   RgnHandle mouseRgn
);
```

Parameters

eventMask

A value that indicates which kinds of events are to be returned. This parameter is interpreted as a sum of event mask constants. You specify the event mask using values defined in "Event Mask Constants" (page 34). To accept all events, you can specify the `everyEvent` constant as the event mask.

If no event of any of the designated types is available, `WaitNextEvent` returns a null event. `WaitNextEvent` determines the next available event to return based on the `eventMask` parameter and the priority of the event.

Events not designated by the event mask remain in the event stream until retrieved by an application. Low-level events in the Operating System event queue are kept in the queue until they are retrieved by your application or another application or until the queue becomes full. Once the queue becomes full, the Operating System Event Manager begins discarding the oldest events in the queue.

theEvent

A pointer to an event structure for the next available event of the specified type or types. The `WaitNextEvent` function removes the returned event from the event stream and returns the information about the event in an event structure. The event structure includes the type of event received and other information.

In addition to the event structure, high-level events can contain additional data; use the Apple Event Manager `AEProcessAppleEvent` function to get additional data associated with these events.

sleep

The number of ticks (a tick is approximately 1/60 of a second) indicating the amount of time your application is willing to relinquish the processor if no events are pending for your application. If you specify a value greater than 0 for the `sleep` parameter, your application relinquishes the processor for the specified time or until an event occurs.

You should not set the `sleep` parameter to a value greater than the number of ticks returned by `GetCaretTime` if your application provides text-editing capabilities. When the specified time expires, and if there are no pending events for your application, `WaitNextEvent` returns a `NULL` event in the parameter `theEvent`.

When running on Mac OS X, a Carbon application will block for the entire duration of the `sleep` parameter if there are no events to be delivered. This is slightly different behavior than on Mac OS 9, where the application will often receive `NULL` events before the sleep duration has elapsed.

mouseRgn

A handle to a region that specifies a region inside of which mouse movement does not cause mouse-moved events. In other words, your application receives mouse-moved events only when the cursor is outside the specified region. You should specify the region in global coordinates. If you pass an empty region or a `null` region handle, the Event Manager does not report mouse-moved events to your application. Note that your application should recalculate the `mouseRgn` parameter when it receives a mouse-moved event, or it will continue to receive mouse-moved events as long as the cursor position is outside the original `mouseRgn`.

Return Value

The `WaitNextEvent` function returns `false` as its function result if the event being returned is a null event or if `WaitNextEvent` has intercepted the event; otherwise, `WaitNextEvent` returns `true`.

Discussion

The `WaitNextEvent` function calls the Operating System Event Manager function `SystemEvent` to determine whether the event should be handled by the application or the Operating System.

If no events are pending for your application, `WaitNextEvent` waits for a specified amount of time for an event. (During this time, processing time may be allocated to background processes.) If an event occurs, it is returned through the parameter `theEvent`, and `WaitNextEvent` returns a function result of `true`. If the specified time expires and there are no pending events for your application, `WaitNextEvent` returns a null event in `theEvent` and a function result of `false`.

Before returning an event to your application, `WaitNextEvent` performs other processing and may intercept the event.

The `WaitNextEvent` function intercepts Command–Shift–number key sequences and calls the corresponding `'FKEY '` resource to perform the associated action. The Event Manager's processing of Command–Shift–number key sequences with numbers 3 through 9 can be disabled by setting the `ScrDmpEnable` global variable (a byte) to 0.

If the returned event is a high-level event and your application supports Apple events, use the Apple Event Manager function `AEProcessAppleEvent` to respond to the Apple event and to get additional information associated with the Apple event.

To retrieve an event without removing it from the event stream, use `EventAvail` (page 10).

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

Related Sample Code
HideMenuBar
Simple DrawSprocket

Declared In
`Events.h`

Data Types

EventRecord

Contains information associated with an event.

```
struct EventRecord {
    EventKind what;
    UInt32 message;
    UInt32 when;
    Point where;
    EventModifiers modifiers;
};
typedef struct EventRecord EventRecord;
```

Fields
`what`
: The kind of event received. The Event Manager specifies the kind of event with one of the values defined by the `EventKind` enumeration.

`message`

Additional information associated with the event. The interpretation of this information depends on the event type. The contents of the `message` field for each event type are summarized here:

- For a null, mouse-up, or mouse-down event, the event message is:Undefined.
- For a key-up, key-down, or auto-key event, the event message is:The low-order word contains the character code and virtual key code, which you can access with the constants `charCodeMask` and `keyCodeMask`, respectively. For Apple Desktop Bus (ADB) keyboards, the low byte of the high-order word contains the ADB address of the keyboard where the keyboard event occurred. The high byte of the high-order word is reserved.
- For an update or activate event, the event message is:A pointer to the window to update, activate, or deactivate.
- For a disk-inserted event, the event message is:The drive number in the low-order word, the File Manager result code in the high-order word.
- For a resume event, the event message is:The `suspendResumeMessage` enumerator in bits 24–31 and a 1 (the `resumeFlag` enumerator) in bit 0 indicate the event is a resume event. Bit 1 contains a 1 (the `convertClipBoardFlag` enumerator) if Clipboard conversion is required, and bits 2–23 are reserved.
- For a suspend event, the event message is:The `suspendResumeMessage` enumerator in bits 24–31 and a 0 in bit 0 to indicate the event is a suspend event. Bit 1 is undefined, and bits 2–23 are reserved.
- For a mouse-moved event, the event message is:The `mouseMovedMessage` enumerator in bits 24–31. Bits 2–23 are reserved, and bit 0 and bit 1 are undefined.
- For a high-level event, the event message is:

  The class of events to which the high-level event belongs. The `message` and `where` fields of a high-level event define the specific type of high-level event received.

`when`

The `when` field indicates the time when the event was posted (in ticks since system startup).

`where`

For low-level events and operating-system events, the `where` field contains the location of the cursor at the time the event was posted (in global coordinates).

For high-level events, the `where` field contains a second event specifier, the event ID. The event ID defines the particular type of event within the class of events defined by the `message` field of the high-level event. For high-level events, you should interpret the `where` field as having the data type `OSType`, not `Point`.

`modifiers`

The `modifiers` field contains information about the state of the modifier keys and the mouse button at the time the event was posted. For activate events, this field also indicates whether the window should be activated or deactivated. In System 7 it also indicates whether the mouse-down event caused your application to switch to the foreground.

Each of the modifier keys is represented by a specific bit in the `modifiers` field of the event structure. The modifier keys include the Option, Command, Caps Lock, Control, and Shift keys. If your application attaches special meaning to any of these keys in combination with other keys or when the mouse button is down, you can test the state of the `modifiers` field to determine the action your application should take. For example, you can use this information to determine whether the user pressed the Command key and another key to make a menu choice.

Discussion

When your application uses an Event Manager function to retrieve an event, the Event Manager returns information about the retrieved event in an event structure, which is a structure of type `EventRecord`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Events.h`

EvQEl

Defines an event queue entry.

```
struct EvQEl {
    QElemPtr qLink;
    SInt16 qType;
    EventKind evtQWhat;
    UInt32 evtQMessage;
    UInt32 evtQWhen;
    Point evtQWhere;
    EventModifiers evtQModifiers;
};
typedef struct EvQEl EvQEl;
typedef EvQEl * EvQElPtr;
```

Fields

`qLink`

Next queue entry.

`qType`

Queue type (`evType`).

`evtQWhat`

Event code.

`evtQMessage`

Event message.

`evtQWhen`

Ticks since startup.

`evtQWhere`

Mouse location.

`evtQModifiers`
: Modifier flags.

Discussion

A structure of type `EvQEl` defines an entry in the Operating System event queue. Each entry in the event queue begins with 4 bytes of flags followed by a pointer to the next queue entry. The flags are maintained by and internal to the Operating System Event Manager. The queue entries are linked by pointers, and the first field of the `EvQEl` data type, which represents the structure of a queue entry, begins with a pointer to the next queue entry. Thus, you cannot directly access the flags using the `EvQEl` data type.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Events.h`

KeyMap

Contains information about the current state of the keyboard.

```
typedef BigEndianLong KeyMap[4];
```

Discussion

The type `KeyMap` is used in `GetKeys` (page 14) to return the current state of the keyboard, including the keypad, if any. The `KeyMap` type is interpreted as an array of 128 elements, each having a Boolean value. Each key on the keyboard or keypad corresponds to an element in the `KeyMap` array. A `KeyMap` element is `true` if the corresponding key is down and `false` if it isn't. The maximum number of keys that can be down simultaneously is two character keys plus any combination of the five modifier keys.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Events.h`

KeyMapByteArray

Contains information about the current state of the keyboard.

```
typedef UInt8 KeyMapByteArray[16];
```

Discussion

The type `KeyMapByteArray` is an alternate version of the type `KeyMap` (page 26) for use on little endian platforms.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Events.h`

Constants

Event Modifier Constants

Define modifiers for event types.

```
enum {
    activeFlagBit = 0,
    btnStateBit = 7,
    cmdKeyBit = 8,
    shiftKeyBit = 9,
    alphaLockBit = 10,
    optionKeyBit = 11,
    controlKeyBit = 12,
    rightShiftKeyBit = 13,
    rightOptionKeyBit = 14,
    rightControlKeyBit = 15
};
typedef UInt16 EventModifiers;
```

Constants

`activeFlagBit`

Available in Mac OS X v10.0 and later.

Declared in `Events.h`.

`btnStateBit`

Available in Mac OS X v10.0 and later.

Declared in `Events.h`.

`cmdKeyBit`

Available in Mac OS X v10.0 and later.

Declared in `Events.h`.

`shiftKeyBit`

Available in Mac OS X v10.0 and later.

Declared in `Events.h`.

`alphaLockBit`

Available in Mac OS X v10.0 and later.

Declared in `Events.h`.

`optionKeyBit`

Available in Mac OS X v10.0 and later.

Declared in `Events.h`.

`controlKeyBit`

Available in Mac OS X v10.0 and later.

Declared in `Events.h`.

`rightShiftKeyBit`

Available in Mac OS X v10.0 and later.

Declared in `Events.h`.

`rightOptionKeyBit`
: Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`rightControlKeyBit`
: Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

charCodeMask

```
enum {
    charCodeMask = 0x000000FF,
    keyCodeMask = 0x0000FF00,
    adbAddrMask = 0x00FF0000,
    osEvtMessageMask = 0xFF000000
};
```

Constants

`charCodeMask`
: The enumerator indicating you want your application to receive a character-code keyboard event.

  Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`keyCodeMask`
: The enumerator indicating you want your application to receive a key-code keyboard event.

  Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`adbAddrMask`
: The enumerator indicating you want your application to receive an ADB address if there is an ADB keyboard.

  Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`osEvtMessageMask`
: The enumerator indicating you want your application to receive a keyboard event that can be used to extract a message code.

  Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

convertClipboardFlag

Obsolete in Carbon.

```
enum {
    convertClipboardFlag = 2
};
```

Constants

`convertClipboardFlag`

Available in Mac OS X v10.0 and later.

Declared in `Events.h`.

Discussion

Obsolete in Carbon. To determine if the clipboard has changed while your application was suspended, use the Scrap Manager function `GetCurrentScrap`.

Carbon Porting Notes

Unsupported. To determine if the clipboard has changed while your application was suspended, use the Scrap Manager function `GetCurrentScrap`.

Event Modifier Bits

Modifer bits for events.

```
enum {
    activeFlag = 1 << activeFlagBit,
    btnState = 1 << btnStateBit,
    cmdKey = 1 << cmdKeyBit,
    shiftKey = 1 << shiftKeyBit,
    alphaLock = 1 << alphaLockBit,
    optionKey = 1 << optionKeyBit,
    controlKey = 1 << controlKeyBit,
    rightShiftKey = 1 << rightShiftKeyBit,
    rightOptionKey = 1 << rightOptionKeyBit,
    rightControlKey = 1 << rightControlKeyBit
};
```

Constants

`activeFlag`

The enumerator that indicates a window is being activated or that a mouse-down event caused a foreground switch.

Available in Mac OS X v10.0 and later.

Declared in `Events.h`.

`btnState`

The enumerator indicating that the mouse button has been released.

Available in Mac OS X v10.0 and later.

Declared in `Events.h`.

`cmdKey`

The enumerator indicating that the Command key is being pressed.

Available in Mac OS X v10.0 and later.

Declared in `Events.h`.

`shiftKey`
: The enumerator indicating that the Shift key is being pressed.

  Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`alphaLock`
: The enumerator indicating that the Caps Lock key is being pressed.

  Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`optionKey`
: The enumerator indicating that the Option key is being pressed.

  Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`controlKey`
: The enumerator indicating that the Control key is being pressed.

  Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`rightShiftKey`
: Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`rightOptionKey`
: Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`rightControlKey`
: Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

HighLevelEventMsgClass

```
enum {
    HighLevelEventMsgClass = 'jaym',
    rtrnReceiptMsgID = 'rtrn'
};
```

Constants

`HighLevelEventMsgClass`
: The enumerator indicating a high-level event message class for return receipt.

`rtrnReceiptMsgID`
: The posting enumerator indicating the return receipt message ID.

Character Codes

Define character codes for events.

```
enum {
    kNullCharCode = 0,
    kHomeCharCode = 1,
    kEnterCharCode = 3,
    kEndCharCode = 4,
    kHelpCharCode = 5,
    kBellCharCode = 7,
    kBackspaceCharCode = 8,
    kTabCharCode = 9,
    kLineFeedCharCode = 10,
    kVerticalTabCharCode = 11,
    kPageUpCharCode = 11,
    kFormFeedCharCode = 12,
    kPageDownCharCode = 12,
    kReturnCharCode = 13,
    kFunctionKeyCharCode = 16,
    kCommandCharCode = 17,
    kCheckCharCode = 18,
    kDiamondCharCode = 19,
    kAppleLogoCharCode = 20,
    kEscapeCharCode = 27,
    kClearCharCode = 27,
    kLeftArrowCharCode = 28,
    kRightArrowCharCode = 29,
    kUpArrowCharCode = 30,
    kDownArrowCharCode = 31,
    kSpaceCharCode = 32,
    kDeleteCharCode = 127,
    kBulletCharCode = 165,
    kNonBreakingSpaceCharCode = 202
};
```

Constants

`kNullCharCode`

Available in Mac OS X v10.0 and later.

Declared in `Events.h`.

`kHomeCharCode`

Available in Mac OS X v10.0 and later.

Declared in `Events.h`.

`kEnterCharCode`

Available in Mac OS X v10.0 and later.

Declared in `Events.h`.

`kEndCharCode`

Available in Mac OS X v10.0 and later.

Declared in `Events.h`.

`kHelpCharCode`

Available in Mac OS X v10.0 and later.

Declared in `Events.h`.

`kBellCharCode`

Available in Mac OS X v10.0 and later.

Declared in `Events.h`.

`kBackspaceCharCode`
: Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`kTabCharCode`
: Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`kLineFeedCharCode`
: Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`kVerticalTabCharCode`
: Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`kPageUpCharCode`
: Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`kFormFeedCharCode`
: Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`kPageDownCharCode`
: Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`kReturnCharCode`
: Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`kFunctionKeyCharCode`
: Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`kCommandCharCode`
: Available in Mac OS X v10.1 and later.

  Declared in `Events.h`.

`kCheckCharCode`
: Available in Mac OS X v10.1 and later.

  Declared in `Events.h`.

`kDiamondCharCode`
: Available in Mac OS X v10.1 and later.

  Declared in `Events.h`.

`kAppleLogoCharCode`
: Available in Mac OS X v10.1 and later.

  Declared in `Events.h`.

`kEscapeCharCode`
: Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`kClearCharCode`
: Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`kLeftArrowCharCode`
: Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`kRightArrowCharCode`
: Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`kUpArrowCharCode`
: Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`kDownArrowCharCode`
: Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`kSpaceCharCode`
: Available in Mac OS X v10.1 and later.

  Declared in `Events.h`.

`kDeleteCharCode`
: Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`kBulletCharCode`
: Available in Mac OS X v10.1 and later.

  Declared in `Events.h`.

`kNonBreakingSpaceCharCode`
: Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

kShiftUnicode

```
enum {
    kShiftUnicode = 0x21E7,
    kControlUnicode = 0x2303,
    kOptionUnicode = 0x2325,
    kCommandUnicode = 0x2318,
    kPencilUnicode = 0x270E,
    kCheckUnicode = 0x2713,
    kDiamondUnicode = 0x25C6,
    kBulletUnicode = 0x2022,
    kAppleLogoUnicode = 0xF8FF
};
```

Constants

`kShiftUnicode`
: Available in Mac OS X v10.1 and later.

  Declared in `Events.h`.

`kControlUnicode`
: Available in Mac OS X v10.1 and later.

  Declared in `Events.h`.

`kOptionUnicode`
: Available in Mac OS X v10.1 and later.

  Declared in `Events.h`.

`kCommandUnicode`
: Available in Mac OS X v10.1 and later.

  Declared in `Events.h`.

`kPencilUnicode`
: Available in Mac OS X v10.1 and later.

  Declared in `Events.h`.

`kCheckUnicode`
: Available in Mac OS X v10.1 and later.

  Declared in `Events.h`.

`kDiamondUnicode`
: Available in Mac OS X v10.1 and later.

  Declared in `Events.h`.

`kBulletUnicode`
: Available in Mac OS X v10.1 and later.

  Declared in `Events.h`.

`kAppleLogoUnicode`
: Available in Mac OS X v10.1 and later.

  Declared in `Events.h`.

Event Mask Constants

Define constants you can use in the event mask.

```
enum {
    mDownMask = 1 << mouseDown,
    mUpMask = 1 << mouseUp,
    keyDownMask = 1 << keyDown,
    keyUpMask = 1 << keyUp,
    autoKeyMask = 1 << autoKey,
    updateMask = 1 << updateEvt,
    diskMask = 1 << diskEvt,
    activMask = 1 << activateEvt,
    highLevelEventMask = 0x0400,
    osMask = 1 << osEvt,
    everyEvent = 0xFFFF
};
typedef UInt16 EventMask;
```

Constants

`mDownMask`

The enumerator indicating you want your application to receive a mouse-down event.

Available in Mac OS X v10.0 and later.

Declared in `Events.h`.

`mUpMask`

The enumerator indicating you want your application to receive a mouse-up event.

Available in Mac OS X v10.0 and later.

Declared in `Events.h`.

`keyDownMask`

The enumerator indicating you want your application to receive a key-down event.

Available in Mac OS X v10.0 and later.

Declared in `Events.h`.

`keyUpMask`

The enumerator indicating you want your application to receive a key-up event.

Available in Mac OS X v10.0 and later.

Declared in `Events.h`.

`autoKeyMask`

The enumerator indicating you want your application to receive an auto-key event.

Available in Mac OS X v10.0 and later.

Declared in `Events.h`.

`updateMask`

The enumerator indicating you want your application to receive an update event.

Available in Mac OS X v10.0 and later.

Declared in `Events.h`.

`diskMask`

The enumerator indicating you want your application to receive a disk-inserted event.

Available in Mac OS X v10.0 and later.

Declared in `Events.h`.

`activMask`
: The enumerator indicating you want your application to receive an activate event.

  Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`highLevelEventMask`
: The enumerator indicating you want your application to receive a high-level event.

  Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`osMask`
: The enumerator indicating you want your application to receive an operating-system event

  Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`everyEvent`
: The enumerator indicating you want your application to receive every event.

  Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

mouseMovedMessage

```
enum {
    mouseMovedMessage = 0x00FA,
    suspendResumeMessage = 0x0001
};
```

Constants

`mouseMovedMessage`
: The message code indicating the mouse-moved operating-system event.

  Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`suspendResumeMessage`
: The message code indicating a suspend or resume operating-system event.

  Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

msgWasPartiallyAccepted

```
enum {
    msgWasPartiallyAccepted = 2,
    msgWasFullyAccepted = 1,
    msgWasNotAccepted = 0
};
```

Constants

`msgWasPartiallyAccepted`
: The posting enumerator value in the return receipt that indicates the message was partially accepted.

`msgWasFullyAccepted`
: The posting enumerator value in the return receipt that indicates the message was fully accepted.

`msgWasNotAccepted`
: The posting enumerator value in the return receipt that indicates the message was fully accepted.

networkEvt

```
enum {
    networkEvt = 10,
    driverEvt = 11,
    app1Evt = 12,
    app2Evt = 13,
    app3Evt = 14,
    app4Evt = 15,
    networkMask = 0x0400,
    driverMask = 0x0800,
    app1Mask = 0x1000,
    app2Mask = 0x2000,
    app3Mask = 0x4000,
    app4Mask = 0x8000
};
```

Constants

`networkEvt`
: Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`driverEvt`
: Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`app1Evt`
: Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`app2Evt`
: Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`app3Evt`
: Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`app4Evt`
: Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`networkMask`
: Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`driverMask`
: Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`app1Mask`
: Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`app2Mask`
: Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`app3Mask`
: Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`app4Mask`
: Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

Event Kind Constants

Define event kinds.

```
enum {
    nullEvent = 0,
    mouseDown = 1,
    mouseUp = 2,
    keyDown = 3,
    keyUp = 4,
    autoKey = 5,
    updateEvt = 6,
    diskEvt = 7,
    activateEvt = 8,
    osEvt = 15,
    kHighLevelEvent = 23
};
typedef UInt16 EventKind;
```

Constants

`nullEvent`
: The event code indicating that there are no other pending events.

  Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`mouseDown`
: The event code indicating that the mouse button has been pressed.

  Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`mouseUp`
: The event code indicating that the mouse button has been released.

  Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`keyDown`
: The event code indicating that a key has been pressed.

  Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`keyUp`
: The event code indicating that a key has been released.

  Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`autoKey`
: The event code indicating that a key has been repeatedly held down.

  Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`updateEvt`
: The event code indicating that a window needs updating.

  Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`diskEvt`
: The event code indicating that a disk has been inserted.

  Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`activateEvt`
: The event code indicating that a window has been activated or deactivated.

  Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`osEvt`
: The event code indicating a suspend, resume, or mouse-moved operating-system event.

  Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

`kHighLevelEvent`
: A high-level event.

  Available in Mac OS X v10.0 and later.

  Declared in `Events.h`.

resumeFlag

Indicates a resume event.

```
enum {
    resumeFlag = 1
};
```

Constants

`resumeFlag`

Flag for a resume event.

Available in Mac OS X v10.0 and later.

Declared in `Events.h`.

Result Codes

Result codes defined for the Event Manager are listed below.

Result Code	Value	Description
`evtNotEnb`	1	Event not enabled for `PostEvent`. Available in Mac OS X v10.0 and later.

Deprecated Event Manager Reference (Not Recommended) Functions

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

Deprecated in Mac OS X v10.5

KeyScript

Changes the keyboard script (the script system used for keyboard input), changes the keyboard layout (the mapping of keys to characters) or input method within the current keyboard script (a facility for entering 2-byte characters), or makes a setting related to text input, using the supplied value. (Deprecated in Mac OS X v10.5.)

```
void KeyScript (
   short code
);
```

Parameters

code

If 0 or positive, directly specifies a script system (that is, it is read as a script code). Negative values have special meanings.

The `code` parameter is a selector that can explicitly specify a keyboard script by script code. See the Script Manager for a list of script codes. If the selector specifies a script, then the current default keyboard layout (`'KCHR'` resource) for that script, as specified in the script's international bundle resource, becomes the current keyboard layout.

The selector can also implicitly specify a keyboard script (for example, the next script), a keyboard layout (for example, the previously used keyboard layout in the current script), or an input method (for example, inline input versus window-based input). It can also specify settings that enable or disable keyboard layouts and keyboard scripts, and toggle among input options or line direction.

Discussion

For the purposes of `KeyScript`, keyboard layout means a keyboard-layout (`'KCHR'`) resource, plus optionally a key-remap (`'itlk'`) resource. To change keyboard layouts means to change the current keyboard-layout resource.

If the Keyboard menu is displayed, `KeyScript` also updates the Keyboard menu.

If you call `KeyScript` and explicitly specify a script system that is not available, `KeyScript` does nothing. The current keyboard script remains unchanged.

Note that `Keyscript` also does nothing when passed a positive or zero script code if the user has Font and Keyboard Synchronization turned off. You can find the Font and Synchronization checkbox under Options in the Input Menu tab of the International System Preference in Mac OS X.

Special Considerations

`KeyScript` operates only on those keyboard-layout and key-remap resources that are present in the System file.

Your application's keyboard-menu setting is not maintained by the Process Manager if the state of the keyboard menu is changed while you are switched out, the Process Manager does not restore your setting when you are switched back in. However, the Process Manager does maintain the keyboard disable state (Script Manager variable `smKeyDisableState`) for your application. See the Script Manager for a description of the `smKeyDisableState` variable. `KeyScript` may move memory; your application should not call this function at interrupt time.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`Events.h`

Legacy Functions

This appendix describes functions that are not recommended or are not available in Carbon and Mac OS X. The API documentation for these functions has been moved to *Event Manager Legacy Reference*.

Not Recommended

Table B-1 lists Event Manager functions that are not recommended in Carbon and Mac OS X. You should avoid using any calls that poll the state of the mouse button, as they use excessive processor time and slow down the system. In most cases you are more interested in the transitions of the mouse button rather than its instantaneous state, so you should adopt Carbon events and take action on mouse-up and mouse-down events. If you need to track the mouse while down, consider using the Carbon Event Manager functions `TrackMouseLocation` or `TrackMouseRegion`. If you need to know the button state, you should call the `GetCurrentEventButtonState` function.

Table B-1 Functions that are not recommended

Name	Porting Notes
`Button`	You should avoid using any calls that poll the state of the mouse button.
`GetMouse`	You should avoid using any calls that poll the state of the mouse button.
`StillDown`	You should avoid using any calls that poll the state of the mouse button.
`WaitMouseUp`	You should avoid using any calls that poll the state of the mouse button.

Not Available

Table B-2 lists Event Manager functions that are not available in Carbon and Mac OS X.

Table B-2 Functions that are not available

Name	Porting Notes
`AcceptHighLevelEvent`	The High Level Event APIs (`EPPC.h`) are not supported in Carbon. Instead, use Apple events.
`DisposeFKEYUPP`	FKEYs are not supported in Carbon because they involve loading code from resources, which isn't supported under Carbon, and because very few applications use them.

Name	Porting Notes
DisposeGetNextEvent-FilterUPP	GetNextEvent (GNE) filters patch the GetNextEvent function and therefore are not supported in Carbon.
DisposeGetSpecific-FilterUPP	The High Level Event APIs (EPPC.h) are not supported in Carbon. Instead, use Apple events.
GetPortNameFrom-ProcessSerialNumber	The GetPortNameFromProcessSerialNumber function does not move or purge memory but for other reasons should not be called from within an interrupt, such as in a completion function or VBL task.
GetProcessSerial-NumberFromPortName	The GetProcessSerialNumberFromPortName function does not move or purge memory but for other reasons should not be called from within an interrupt, such as in a completion function or VBL task.
GetEvQHdr	Returns a global system data structure, so it will not be supported in the future.
GetOSEvent	GetOSEvent is not supported in Carbon. Use the GetNextEvent function instead
OSEventAvail	OSEventAvail is not supported in Carbon. Use the EventAvail function instead.
GetSpecificHighLevelEvent	The GetSpecificHighLevelEvent function may move or purge memory. You should not call this function from within an interrupt, such as in a completion function or VBL task.
InvokeFKEYUPP	FKEYs are not supported in Carbon because they involve loading code from resources, which isn't supported under Carbon, and because very few applications use them.
InvokeGetNextEvent-FilterUPP	GetNextEvent (GNE) filters patch the GetNextEvent function and therefore are not supported in Carbon.
InvokeGetSpecific-FilterUPP	The High Level Event APIs (EPPC.h) are not supported in Carbon. Instead, use Apple events.
NewFKEYUPP	FKEYs are not supported in Carbon because they involve loading code from resources, which isn't supported under Carbon, and because very few applications use them.
NewGetNextEventFilterUPP	GetNextEvent (GNE) filters patch the GetNextEvent function and therefore are not supported in Carbon.
NewGetSpecificFilterUPP	The High Level Event APIs (EPPC.h) are not supported in Carbon. Instead, use Apple events.
PostHighLevelEvent	The High Level Event APIs (EPPC.h) are not supported in Carbon. Instead, use Apple events.
PPostEvent	Posting events in the Operating System event queue is not supported in Carbon.

Name	Porting Notes
`SystemClick`	Desk accessories are not supported in Carbon.
`SystemEvent`	Desk accessories are not supported in Carbon.
`SystemTask`	In Carbon, the Event Manager automatically handles all task scheduling.

Document Revision History

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

Date	Notes
2007-10-31	Made minor technical and format changes.
2007-05-03	Added cross-references to "Carbon Event Manager Programming Guide."
2003-04-01	Added documentation for the following functions: `LMGetKbdType` (page 17), `LMGetKbdLast` (page 16), `LMSetKbdLast` (page 18), `LMSetKbdType` (page 18), and `IsCmdChar` (page 15).
2003-02-01	Updated formatting and linking.
	Moved not recommended and unsupported functions into *Event Manager Legacy Reference*.
	Updated `KeyScript` (page 41) discussion, indicating that Keyscript does nothing when passed a positive or zero script code if the user has Font and Keyboard Synchronization turned off.

Index