Event Manager Legacy Reference

(Legacy)

Carbon > Events & Other Input

2007-05-03

Apple Inc.
© 2007 Apple Inc.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Inc., with the following exceptions: Any person is hereby authorized to store documentation on a single computer for personal use only and to print copies of documentation for personal use provided that the documentation contains Apple's copyright notice.

The Apple logo is a trademark of Apple Inc.

Use of the "keyboard" Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

No licenses, express or implied, are granted with respect to any of the technology described in this document. Apple retains all intellectual property rights associated with the technology described in this document. This document is intended to assist application developers to develop applications only for Apple-labeled computers.

Every effort has been made to ensure that the information in this document is accurate. Apple is not responsible for typographical errors.

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

Apple, the Apple logo, Carbon, Mac, Mac OS, and Macintosh are trademarks of Apple Inc., registered in the United States and other countries.

Simultaneously published in the United States and Canada.

Even though Apple has reviewed this document, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS DOCUMENT IS PROVIDED "AS IS," AND YOU, THE READER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS DOCUMENT, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Contents

Event Manager Legacy Reference (Legacy)

Framework:	Carbon/Carbon.h
Declared in	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. Morevover, 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.

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

Deprecated Functions

`Button` (page 8)
Determines whether the user pressed the mouse button.

`StillDown` (page 20)
: After receiving a mouse-down event, you can use the `StillDown` function to determine if the mouse button is still down.

`WaitMouseUp` (page 22)
: After receiving a mouse-down event, determines if the user subsequently released the mouse.

`GetMouse` (page 31) Deprecated in Mac OS X v10.5
: Obtains the current mouse location.

Unsupported Functions

These functions are not supported in Carbon.

`AcceptHighLevelEvent` (page 7)
: Obtains additional information associated with an event after receiving a high-level event (other than an Apple event).

`DisposeFKEYUPP` (page 9)

`DisposeGetNextEventFilterUPP` (page 9)

`DisposeGetSpecificFilterUPP` (page 10)

`GetPortNameFromProcessSerialNumber` (page 11)
: Obtains the port name of a process.

`GetProcessSerialNumberFromPortName` (page 12)
: Obtains the process serial number of a process.

`GetEvQHdr` (page 10)
: Obtains a pointer to the header of the Operating System event queue.

`GetOSEvent` (page 10)
: Retrieves low-level events stored in the Operating System event queue.

`OSEventAvail` (page 16)
: Retrieves an event from the Operating System event queue without removing it.

`GetSpecificHighLevelEvent` (page 13)
: Selects and optionally retrieves a specific high-level event from your application's high-level event queue.

`InvokeFKEYUPP` (page 14)

`InvokeGetNextEventFilterUPP` (page 14)

`InvokeGetSpecificFilterUPP` (page 15)

`NewFKEYUPP` (page 15)

`NewGetNextEventFilterUPP` (page 15)

Functions

AcceptHighLevelEvent

Obtains additional information associated with an event after receiving a high-level event (other than an Apple event).

Unsupported

```
OSErr AcceptHighLevelEvent (
    TargetID *sender,
    UInt32 *msgRefcon,
    void *msgBuff,
    UInt32 *msgLen
);
```

Parameters

`sender`

A pointer to a structure of type `TargetID` (page 27) whose contents identify the sender of the event. The structure referenced through the `sender` parameter contains the session reference number that identifies the connection with the other application and the port name and location name of the sender.

`msgRefcon`

A pointer to a value that uniquely identifies the communication associated with this event. If you send a response to this event, you should specify the same value as that referenced through the `msgRefcon` parameter so that the sender of the event can associate the reply with the original request.

msgBuff

A pointer to a block of memory where the `AcceptHighLevelEvent` function should return any additional data associated with the event. Your application is responsible for allocating the memory for the additional data pointed to by the `msgBuff` parameter and for setting the `msgLen` parameter to the number of bytes that you have allocated for the data.

If the `msgBuff` parameter points to an area in memory that is not large enough to hold all the data associated with the event, `AcceptHighLevelEvent` returns as much data as the specified memory area can hold, returns the amount of data remaining in the `msgLen` parameter, and returns the result code `bufferIsSmall`.

msgLen

A pointer to a value that specifies the size of the data (in bytes) pointed to by the `msgBuff` parameter. If `AcceptHighLevelEvent` returns the result code `bufferIsSmall`, the value referenced through the `msgLen` parameter contains the number of bytes remaining. You can call `AcceptHighLevelEvent` again to receive the rest of the data.

Return Value

A result code. See "Event Manager Result Codes" (page 28).

Discussion

When your application receives a high-level event, you can use the `AcceptHighLevelEvent` function to get additional data associated with the event. The `AcceptHighLevelEvent` function returns information that identifies the sender of the event and the unique message reference constant of the event.

Your application should allocate memory for any additional data associated with the event, then supply a pointer to the data area and also provide the length in bytes of the data area.

Special Considerations

The `AcceptHighLevelEvent` 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.

Carbon Porting Notes

The High Level Event APIs (`EPPC.h`) are not supported in Carbon. Instead, use Apple events.

Declared In

`EPPC.h`

Button

Determines whether the user pressed the mouse button.

Not recommended

```
Boolean Button ();
```

Parameters

Return Value

Discussion

The `Button` function simply checks to see if the mouse button is down. If so, it returns `true`; otherwise, it returns `false`. To determine whether the mouse button is still down after a mouse-down event, use the `StillDown` (page 20) function. To check to see if the mouse was released, use the `WaitMouseUp` (page 22) function.

Availability
Supported in Carbon. Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.
Available in Mac OS X 10.0 and later.

Carbon Porting Notes

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`. On Mac OS X v.10.2 and later, if you need to know the button state, you should call the `GetCurrentEventButtonState` function.

Declared In
`Events.h`

DisposeFKEYUPP

Unsupported

```
void DisposeFKEYUPP (
    FKEYUPP userUPP
);
```

Parameters

userUPP

Return Value

Carbon Porting Notes

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.

Declared In
`Events.h`

DisposeGetNextEventFilterUPP

Unsupported

```
void DisposeGetNextEventFilterUPP (
    GetNextEventFilterUPP userUPP
);
```

Parameters

userUPP

Return Value

Carbon Porting Notes

`GetNextEvent` (GNE) filters patch the `GetNextEvent` function and therefore are not supported in Carbon.

Declared In
`Events.h`

DisposeGetSpecificFilterUPP

Unsupported

```
void DisposeGetSpecificFilterUPP (
    GetSpecificFilterUPP userUPP
);
```

Parameters

userUPP

Return Value

Carbon Porting Notes

The High Level Event APIs (`EPPC.h`) are not supported in Carbon. Instead, use Apple events.

Declared In

`EPPC.h`

GetEvQHdr

Obtains a pointer to the header of the Operating System event queue.

Unsupported

```
QHdrPtr GetEvQHdr ();
```

Parameters

Return Value

See the Memory Management Utilities documentation for a description of the `QHdrPtr` data type.

Discussion

In most cases, your application should not call the `GetEvQHdr` function. The `GetEvQHdr` function returns a pointer to the header of the Operating System event queue.

Carbon Porting Notes

Returns a global system data structure, so it will not be supported in the future.

Declared In

`Events.h`

GetOSEvent

Retrieves low-level events stored in the Operating System event queue.

Unsupported

```
Boolean GetOSEvent (
    EventMask mask,
    EventRecord *theEvent
);
```

Parameters

mask

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 values defined in "Event Mask Constants" in *Inside Mac OS X: Event Manager Reference*. `GetOSEvent` returns only low-level events stored in the Operating System event queue; it does not return activate, update, operating-system, or high-level events. If no low-level event of any of the designated kinds is available, `GetOSEvent` returns a null event.

theEvent

A pointer to an event structure for the next available low-level event of the specified type or types in the Operating System event queue. The `GetOSEvent` function removes the returned event from the Operating System event queue 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

In most cases, your application should not use this function. The `GetOSEvent` function retrieves and removes an event from the Operating System event queue. `GetOSEvent` returns `false` as its function result if the event being returned is a null event; otherwise, `GetOSEvent` returns `true`. `GetOSEvent` does not intercept or respond to the event in any way. It also does not process Command–Shift– number key combinations or process any alarms set by the user through the Alarm Clock desk accessory.

Carbon Porting Notes

`GetOSEvent` is not supported in Carbon. Use the `GetNextEvent` function instead.

Declared In

`Events.h`

GetPortNameFromProcessSerialNumber

Obtains the port name of a process.

Unsupported

```
OSErr GetPortNameFromProcessSerialNumber (
    PPCPortRec *portName,
    const ProcessSerialNumber *pPSN
);
```

Parameters

portName

Returns a pointer to a PPC port structure, the contents of which specify the port name of the process designated by the `pPSN` parameter. You can use the returned port name to send a high-level event to that process.

pPSN

A pointer to the process serial number of the process whose port name you want.

Return Value

A result code. See "Event Manager Result Codes" (page 28).

Discussion

The `GetPortNameFromProcessSerialNumber` function returns, through the `portName` parameter, the port name registered to a process having a specific process serial number.

Special Considerations

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.

Carbon Porting Notes

The High Level Event APIs (`EPPC.h`) are not supported in Carbon. Instead, use Apple events.

Declared In

`EPPC.h`

GetProcessSerialNumberFromPortName

Obtains the process serial number of a process.

Unsupported

```
OSErr GetProcessSerialNumberFromPortName (
    const PPCPortRec *portName,
    ProcessSerialNumber *pPSN
);
```

Parameters

`portName`

A pointer to a PPC port structure, the contents of which specify the port name registered to a process whose serial number you want.

`pPSN`

Returns a pointer to the process serial number of the process designated through the `portName` parameter. You can use the returned process serial number to send a high-level event to that process. Do not interpret the value of the process serial number.

Return Value

A result code. See "Event Manager Result Codes" (page 28).

Discussion

The `GetProcessSerialNumberFromPortName` function returns, in the `pPSN` parameter, a pointer to the process serial number of the process registered at a specific port.

Special Considerations

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.

Carbon Porting Notes

The High Level Event APIs (`EPPC.h`) are not supported in Carbon. Instead, use Apple events.

Declared In

`EPPC.h`

GetSpecificHighLevelEvent

Selects and optionally retrieves a specific high-level event from your application's high-level event queue.

Unsupported

```
Boolean GetSpecificHighLevelEvent (
    GetSpecificFilterUPP aFilter,
    void *contextPtr,
    OSErr *err
);
```

Parameters

aFilter

A universal function pointer to the application-defined filter function that `GetSpecificHighLevelEvent` should use to search for a specific event; see `GetSpecificFilterProcPtr` (page 24) for details. `GetSpecificHighLevelEvent` calls your filter function once for each event in your application's high-level event queue until the function returns `true` or the end of the queue is reached.

contextPtr

A pointer to a value that specifies the criteria your filter function should use to select a specific event. For example, you can specify the address of a reference constant to search for a particular event, the address of a target ID structure to search for a specific sender of an event, or the address of an event class to search for a specific class of event.

err

`GetSpecificHighLevelEvent` returns, through this parameter, a value that indicates whether any errors occurred. The `err` parameter specifies the `noErr` constant if no errors occurred or `noOutstandingHLE` if no high-level events are pending in your application's high-level event queue.

Return Value

Discussion

You can use the `GetSpecificHighLevelEvent` function to search for a specific high-level event in your application's high-level event queue. You specify a filter function as one of the parameters to `GetSpecificHighLevelEvent`. The `GetSpecificHighLevelEvent` function calls your filter function once for every event in your application's high-level event queue, until your filter function returns `true` or the end of the queue is reached.

The `GetSpecificHighLevelEvent` function passes the value referenced by the `contextPtr` parameter to your filter function. Your filter function also receives as parameters the event structure associated with the high-level event and the target ID structure that identifies the sender of the event. Your filter function can compare the value referenced by the `contextPtr` parameter with any of the other information it receives.

If your filter function finds a match, it can call `AcceptHighLevelEvent` (page 7) if necessary, and then return `true`. If your filter function does not find a match, then it should return `false`.

If your filter function returns `true`, the `GetSpecificHighLevelEvent` function returns `true`. If your filter function returns `false` for all high-level events in your application's event queue, or if there are no high-level events in the queue, `GetSpecificHighLevelEvent` returns `false`.

See `GetSpecificFilterProcPtr` (page 24) for more information about how to define a filter function and the parameters that `GetSpecificHighLevelEvent` passes to your filter function.

Special Considerations

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.

Carbon Porting Notes

The High Level Event APIs (`EPPC.h`) are not supported in Carbon. Instead, use Apple events.

Declared In

`EPPC.h`

InvokeFKEYUPP

Unsupported

```
void InvokeFKEYUPP (
    FKEYUPP userUPP
);
```

Parameters

`userUPP`

Return Value

Carbon Porting Notes

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.

Declared In

`Events.h`

InvokeGetNextEventFilterUPP

Unsupported

```
void InvokeGetNextEventFilterUPP (
    EventRecord *theEvent,
    Boolean *result,
    GetNextEventFilterUPP userUPP
);
```

Parameters

`theEvent`

`result`

`userUPP`

Return Value

Carbon Porting Notes

`GetNextEvent` (GNE) filters patch the `GetNextEvent` function and therefore are not supported in Carbon.

Declared In

`Events.h`

InvokeGetSpecificFilterUPP

Unsupported

```
Boolean InvokeGetSpecificFilterUPP (
    void *contextPtr,
    HighLevelEventMsgPtr msgBuff,
    const TargetID *sender,
    GetSpecificFilterUPP userUPP
);
```

Parameters

contextPtr

msgBuff

sender

userUPP

Return Value

Carbon Porting Notes

The High Level Event APIs (`EPPC.h`) are not supported in Carbon. Instead, use Apple events.

Declared In

`EPPC.h`

NewFKEYUPP

Unsupported

```
FKEYUPP NewFKEYUPP (
    FKEYProcPtr userRoutine
);
```

Parameters

userRoutine

Return Value

Carbon Porting Notes

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.

Declared In

`Events.h`

NewGetNextEventFilterUPP

Unsupported

```
GetNextEventFilterUPP NewGetNextEventFilterUPP (
    GetNextEventFilterProcPtr userRoutine
);
```

Parameters

userRoutine

Return Value

Carbon Porting Notes

Declared In

`Events.h`

NewGetSpecificFilterUPP

Unsupported

```
GetSpecificFilterUPP NewGetSpecificFilterUPP (
    GetSpecificFilterProcPtr userRoutine
);
```

Parameters

userRoutine

Return Value

Carbon Porting Notes

The High Level Event APIs (`EPPC.h`) are not supported in Carbon. Instead, use Apple events.

Declared In

`EPPC.h`

OSEventAvail

Retrieves an event from the Operating System event queue without removing it.

Unsupported

```
Boolean OSEventAvail (
    EventMask mask,
    EventRecord *theEvent
);
```

Parameters

mask

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 values defined in "Event Mask Constants" in *Inside Mac OS X: Event Manager Reference*. `OSEventAvail` returns only low-level events stored in the Operating System event queue; it does not return activate, update, operating-system, or high-level events. If no low-level event of any of the designated types is available, `OSEventAvail` returns a null event.

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

Return Value

Discussion

In most cases your application does not need to use this function. The `OSEventAvail` function retrieves an event from the Operating System event queue without removing it from the queue. The `OSEventAvail` function returns `false` as its function result if the event being returned is a null event; otherwise, `OSEventAvail` returns `true`.

`OSEventAvail` does not intercept or respond to the event in any way. It also does not process Command–Shift–number key combinations or process any alarms set by the user through the Alarm Clock desk accessory.

Special Considerations

If the `OSEventAvail` function 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.

Carbon Porting Notes

OSEventAvail is not supported in Carbon. Use the `EventAvail` function instead.

Declared In

`Events.h`

PostHighLevelEvent

Sends a high-level event to another application.

Unsupported

```
OSErr PostHighLevelEvent (
    const EventRecord *theEvent,
    void *receiverID,
    UInt32 msgRefcon,
    void *msgBuff,
    UInt32 msgLen,
    UInt32 postingOptions
);
```

Parameters

theEvent
: A pointer to the event structure for the event to send. Your application should fill out the `what`, `message`, and `where` fields of the event structure. Specify the `kHighLevelEvent` constant in the `what` field, the event class of the high-level event in the `message` field, and the event ID in the `where` field. You do not need to fill out the `when` or `modifiers` fields; the Event Manager automatically assigns the appropriate values to these fields when you send the message.

receiverID

The recipient of the high-level event. When sending an event to another application on the local computer, you can specify the recipient of the event by session reference number, process serial number, signature, or port name and location name. When sending an event to an application on a remote computer, you can specify the recipient only by the session reference number or by the port name and location name.

To specify a port name and location name, provide the address of a target ID structure in the `receiverID` parameter. To specify a process serial number, provide its address in the `receiverID` parameter. To specify a session reference number, or signature, provide the data in the `receiverID` parameter.

msgRefcon

A unique number that identifies the communication associated with this event. Your application can set this field to any value it chooses. If you are replying to a high-level event, you should use the same value in the `msgRefcon` parameter as specified in the high-level event that originated the request.

msgBuff

A pointer to a data buffer that contains any additional data for the event.

msgLen

The size (in bytes) of the data buffer pointed to by the `msgBuff` parameter.

postingOptions

Options associated with the `receiverID` parameter and delivery options associated with the event. You can specify one or more delivery options to indicate whether you want the other application to receive the event at the next opportunity and to indicate whether you want acknowledgment that the event was received by the other application. You use the options associated with the `receiverID` parameter to indicate how you are specifying the recipient of the event—whether by port name and location name in a target ID structure, by session reference number, by process serial number, or by signature.

For descriptions of the enumerators you can use to specify posting options, see "Posting Options Constants" (page 28).

Return Value

A result code. See "Event Manager Result Codes" (page 28).

Discussion

The `PostHighLevelEvent` function posts the high-level event to the specified process.

If the application to which you are sending a high-level event terminates, you receive the result code `sessionClosedErr` the next time your application calls `PostHighLevelEvent` to send another high-level event to the terminated application. If you do not care about any state information about that session, you can just resend your event. Otherwise, you must restart another session and resend your event.

If your application is running in the background and posts a high-level event that requires the network authentication dialog box to be displayed, `PostHighLevelEvent` returns the `noUserInteractionAllowed` result code, does not display the network authentication dialog box, and does not send the event. If your application receives the `noUserInteractionAllowed` result code, you can use the Notification Manager to inform the user that your application needs attention. When the user brings your application to the foreground, you can repost the event. If the reposting is successful, your application can continue to post high-level events without further user interaction. Note that `PostHighLevelEvent` can return `noUserInteractionAllowed` only on the first posting of a high-level event to a remote target.

Special Considerations

The `PostHighLevelEvent` 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.

Carbon Porting Notes

The High Level Event APIs (`EPPC.h`) are not supported in Carbon. Instead, use Apple events.

Declared In

`EPPC.h`

PPostEvent

Posts events in the Operating System event queue.

Unsupported

```
OSErr PPostEvent (
    EventKind eventCode,
    UInt32 eventMsg,
    EvQElPtr *qEl
);
```

Parameters

`eventCode`

A value of type `EventKind` 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 event mask values: `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 32-bit integer that contains the contents of the `message` field for the event that `PPostEvent` should post in the queue.

`qEl`

You specify the address of a pointer to an event queue entry in this parameter. `PPostEvent` returns the event queue entry of the posted event through this parameter.

Return Value

A result code. See "Event Manager Result Codes" (page 28).

Discussion

In the `eventCode` and `eventMsg` parameters, you specify the value for the `what` and `message` fields of the event's event structure. The `PPostEvent` 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 `PPostEvent` function returns, through the `qEl` parameter, a pointer to the event queue entry of the posted event. You can change any fields of the posted event by changing the fields of its event queue entry. For example, you can change the posted event's modifier keys by changing the value of the `evtQModifiers` field of the event queue entry.

The `PPostEvent` function posts only events that are enabled by the system event mask. If the event queue is full, `PPostEvent` removes the oldest event in the queue and posts the new 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 does not need to post events in the Operating System event queue.

Carbon Porting Notes

Posting events in the Operating System event queue is not supported in Carbon.

Declared In
`Events.h`

StillDown

After receiving a mouse-down event, you can use the `StillDown` function to determine if the mouse button is still down.

Not recommended

```
Boolean StillDown ();
```

Parameters

Return Value

Discussion
The `StillDown` function looks in the Operating System event queue for a mouse event. If it finds one, the `StillDown` function returns `false`. If it does not find any mouse events pending in the Operating System event queue, the `StillDown` function returns `true`.

Availability
Supported in Carbon. Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.
Available in Mac OS X 10.0 and later.
Not available to 64-bit applications.

Carbon Porting Notes

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`. On Mac OS X v.10.2 and later, if you need to know the button state, you should call the `GetCurrentEventButtonState` function.

Declared In
`Events.h`

SystemClick

Handles an event after `FindWindow` returns the `inSysWindow` constant.

Unsupported

```
void SystemClick (
    const EventRecord *theEvent,
    WindowRef theWindow
);
```

Parameters

`theEvent`
A pointer to the event structure for the event.

theWindow

A reference to the window in which the mouse-down event occurred. Pass the window pointer returned by `FindWindow` in this parameter.

Return Value

Discussion

If a mouse-down event occurred in a desk accessory's window, the `SystemClick` function determines which part of the desk accessory's window the cursor was in when the mouse button was pressed and routes the event to the appropriate desk accessory as necessary.

If the mouse button was pressed while the cursor was in the content region of the desk accessory's window and the window is active, `SystemClick` sends the mouse-down event to the desk accessory to process. If the mouse-down event occurred in the content region of the window and the window is inactive, `SystemClick` makes it the active window. It does this by sending your application an activate event to deactivate its front window and directing an event to the desk accessory to activate its window.

If the mouse button was pressed while the cursor was in the drag region or go-away region, `SystemClick` calls the Window Manager function `DragWindow` or `TrackGoAway`, as appropriate. If `TrackGoAway` reports that the user closed the desk accessory, `SystemClick` sends a close message to the desk accessory.

Carbon Porting Notes

Desk accessories are not supported in Carbon.

Declared In

`Events.h`

SystemEvent

Determines if a specific event should be handled by the application or the Operating System.

Unsupported

```
Boolean SystemEvent (
    const EventRecord *theEvent
);
```

Parameters

theEvent

A pointer to the event structure for the event.

Return Value

Discussion

The `WaitNextEvent` and the `GetNextEvent` functions call the `SystemEvent` function. In most cases your application should not call the `SystemEvent` function.

`SystemEvent` returns `false` as its function result if the event should be handled by the application; otherwise, `SystemEvent` takes any appropriate actions and returns `true`.

For activate, update, mouse-up, and keyboard events (including keyboard equivalents of commands), `SystemEvent` checks to see whether the active window belongs to a desk accessory and whether that desk accessory can handle that type of event. If so, `SystemEvent` sends the event to the desk accessory and returns `true`. Otherwise, `SystemEvent` returns `false`.

For mouse-down events and null events, `SystemEvent` returns `false`.

For disk-inserted events, `SystemEvent` attempts to mount the disk using the `PBMountVol` function but returns `false` so that the application can perform further processing if necessary.

See "Event Kind Constants" in *Inside Mac OS X: Event Manager Reference* for a discussion of the fields in the event structure.

Carbon Porting Notes

Desk accessories are not supported in Carbon.

Declared In

`Events.h`

SystemTask

Gives time to each open desk accessory or driver to perform any periodic action.

Unsupported

```
void SystemTask ();
```

Parameters

Return Value

Discussion

The `SystemTask` function gives time to each open desk accessory or driver to perform the periodic action defined for it. A desk accessory or device driver specifies how often the periodic action should occur, and `SystemTask` gives time to the desk accessory or device driver at the appropriate interval.

If your application calls `GetNextEvent` , your application should call `SystemTask` at least every sixtieth of a second. This usually corresponds to calling `SystemTask` once each time through your event loop. If your application does a large amount of processing, you may need to call `SystemTask` more than once in your event loop.

Carbon Porting Notes

In Carbon, the Event Manager automatically handles all task scheduling.

Declared In

`Events.h`

WaitMouseUp

After receiving a mouse-down event, determines if the user subsequently released the mouse.

Not recommended

```
Boolean WaitMouseUp ();
```

Parameters

Return Value

Discussion

The `WaitMouseUp` function looks in the Operating System event queue for a mouse-up event. If it finds one, the `WaitMouseUp` function removes the mouse-up event from the queue and returns `false`. If it does not find any mouse up events pending in the Operating System event queue, the `WaitMouseUp` function returns `true`.

Availability

Supported in Carbon. Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.

Available in Mac OS X 10.0 and later.

Not available to 64-bit applications.

Carbon Porting Notes

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`. On Mac OS X v.10.2 and later, if you need to know the button state, you should call the `GetCurrentEventButtonState` function.

Declared In

`Events.h`

Callbacks

FKEYProcPtr

Defines a function-key callback.

Unsupported.

```
typedef void (*FKEYProcPtr) (
);
```

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

```
void FKEYProcPtr ();
```

Parameters

Return Value

Carbon Porting Notes

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

Availability

Available in Mac OS X v10.0 and later.

Declared In
`Events.h`

GetNextEventFilterProcPtr

Defines an event filter callback.

Unsupported

```
typedef void (*GetNextEventFilterProcPtr)
(
    EventRecord * theEvent,
    Boolean * result
);
```

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

```
void GetNextEventFilterProcPtr (
    EventRecord * theEvent,
    Boolean * result
);
```

Parameters

`theEvent`

`result`

Return Value

Carbon Porting Notes

`GetNextEvent` (GNE) filters patch the `GetNextEvent` function and therefore are not supported in Carbon.

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

Declared In
`Events.h`

GetSpecificFilterProcPtr

Defines a pointer to a filter callback function. The Event Manager calls your filter function once for each event in the high-level event queue until your filter function returns `true` or the end of the queue is reached.

Unsupported

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

```
Boolean MyGetSpecificFilterCallback (
    void *contextPtr,
    HighLevelEventMsgPtr msgBuff,
    const TargetID *sender
);
```

Parameters

`contextPtr`
: The address of data that specifies the criteria your filter function should use to select a specific event. For example, you can specify in the `contextPtr` parameter the address of a reference constant to search for a particular event, the address of a target ID structure to search for a specific sender of an event, or the address of an event class to search for a specific class of event.

`msgBuff`
: A pointer to a structure of type `HighLevelEventMsg`, which provides: the event structure for the high-level event and the reference constant of the event.

`sender`
: The address of the target ID structure of the application that sent the event. The target ID structure is described in `TargetID` (page 27).

Return Value

Discussion

When you use `GetSpecificHighLevelEvent` (page 13) to search the high-level event queue of your application for a specific event, you supply a pointer to a filter function. Your filter function can examine each event and determine whether that event is the desired event. If so, your filter function should return `true`.

Your filter function can compare the contents of the `contextPtr` parameter with the contents of the `msgBuff` and `sender` parameters. If your filter function finds a match, it can call `AcceptHighLevelEvent`, if necessary, and your filter function should return `true`. If your filter function does not find a match, it should return `false`.

The Event Manager defines the universal function pointer `GetSpecificFilterUPP` for an application-defined filter function.

typedef UniversalProcPtr GetSpecificFilterUPP;

The Event Manager also defines the macro `NewGetSpecificFilterProc` for obtaining a `GetSpecificFilterUPP` :

#define NewGetSpecificFilterProc(userRoutine) \(GetSpecificFilterUPP)

NewRoutineDescriptor((ProcPtr)(userRoutine), uppGetSpecificFilterProcInfo, GetCurrentArchitecture())

You typically use the `NewGetSpecificFilterProc` macro like this:

GetSpecificFilterUPP myFilterFunctionProc;

myFilterFunctionProc = NewGetSpecificFilterProc(MyGetSpecificFilterCallback);

The Event Manager also defines the macro `CallGetSpecificFilterProc` for calling a `GetSpecificFilterUPP`. You normally don't need to call this macro directly:

#define CallGetSpecificFilterProc(userRoutine, contextPtr, msgBuff, sender)\
CallUniversalProc((UniversalProcPtr)(userRoutine), uppGetSpecificFilterProcInfo, (contextPtr), (msgBuff), (sender))

Carbon Porting Notes

The High Level Event APIs (`EPPC.h`) are not supported in Carbon. Instead, use Apple events.

Data Types

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`

GNEFilterUPP

```
typedef GetNextEventFilterUPP GNEFilterUPP;
```

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

Declared In
`Events.h`

HighLevelEventMsg

Defines a high-level event message structure.

```
struct HighLevelEventMsg {
    UInt16 HighLevelEventMsgHeaderLength;
    UInt16 version;
    UInt32 reserved1;
    EventRecord theMsgEvent;
    UInt32 userRefcon;
    UInt32 postingOptions;
    UInt32 msgLength;
};
typedef HighLevelEventMsg* HighLevelEventMsgPtr;
typedef HighLevelEventMsgPtr* HighLevelEventMsgHandle;
typedef HighLevelEventMsgHandle HighLevelEventMsgHdl;
```

Discussion
You can search your application's high-level event queue for a specific high-level event by using the `GetSpecificHighLevelEvent` (page 13) and providing a filter function. Your filter function receives a pointer to a high-level event message structure that contains information about a high-level event.

For information on getting a function descriptor for your filter function, see"Resources". For information on how to define a filter function, see `GetSpecificFilterProcPtr` (page 24).

TargetID

```
struct TargetID {
    SInt32 sessionID;
    PPCPortRec name;
    LocationNameRec location;
    PPCPortRec recvrName;
};
typedef TargetID SenderID;
typedef TargetID* TargetIDPtr;
typedef TargetIDPtr* TargetIDHandle;
typedef TargetIDHandle TargetIDHdl;
```

Discussion
When you send a high-level event to another application, you can use a target ID structure to specify the recipient of the event. When you receive a high-level event, the `AcceptHighLevelEvent` (page 7) uses a target ID structure to return information about the sender of the event.

Constants

Posting Options Constants

Options for posting an event to the queue.

```
enum {
    receiverIDMask = 61440,
    receiverIDisPSN = 32768,
    receiverIDisSignature = 28672,
    receiverIDisSessionID = 24576,
    receiverIDisTargetID = 20480,
    systemOptionsMask = 3840,
    nReturnReceipt = 512,
    priorityMask = 255,
    nAttnMsg = 1
};
```

Constants

`receiverIDMask`
: The posting enumerator indicating that the receiver ID consists of the bits for the event to be delivered.

`receiverIDisPSN`
: The posting enumerator indicating that the receiver ID is the process serial number for the event to be delivered.

`receiverIDisSignature`
: The posting enumerator indicating that the receiver ID is a creator signature.

`receiverIDisSessionID`
: The posting enumerator indicating that the receiver ID is a PPC session reference number.

`receiverIDisTargetID`
: The posting enumerator indicating that the receiver ID is a port name and location name.

`systemOptionsMask`
: The posting enumerator indicating that the system options mask.

`nReturnReceipt`
: The posting enumerator indicating that a return receipt has been requested.

`priorityMask`
: The posting enumerator indicating priority.

`nAttnMsg`
: The posting enumerator indicating that this message should be given priority.

Discussion

You use posting option enumerators with `PostHighLevelEvent` (page 17) to indicate how you are specifying the receiver of the high-level event and how you want the event to be delivered.

Result Codes

The most common result codes returned by Event Manager are listed below.

Result Code	Value	Description
`evtNotEnb`	1	Available in Mac OS X v10.0 and later.
`noErr`	0	No error Available in Mac OS X v10.0 and later.
`procNotFound`	-600	Available in Mac OS X v10.0 and later.
`bufferIsSmall`	-607	Available in Mac OS X v10.0 and later.
`noOutstandingHLE`	-608	Available in Mac OS X v10.0 and later.
`connectionInvalid`	-609	Available in Mac OS X v10.0 and later.
`noUserInteractionAllowed`	-610	Available in Mac OS X v10.0 and later.
`noPortErr`	-903	Available in Mac OS X v10.0 and later.

Deprecated Event Manager Legacy Reference (Legacy) Functions

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

Deprecated in Mac OS X v10.5

GetMouse

Obtains the current mouse location. (Deprecated in Mac OS X v10.5.)

Not recommended

```
void GetMouse (
    Point * mouseLoc
);
```

Parameters

mouseLoc

Returns a pointer to a point describing the current mouse location in local coordinates of the current graphics port (for example, the active window). Note that this value differs from the value of the `where` field of the event structure, which specifies the mouse location in global coordinates.

Return Value

Availability

Supported in Carbon. Available in CarbonLib 1.0 and later when running Mac OS 8.1 or later.

Available in Mac OS X 10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Carbon Porting Notes

You should adopt Carbon events and take action on mouse-moved events rather than poll the mouse position directly.

Declared In

`Events.h`

Document Revision History

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

Date	Notes
2007-05-03	Moved to legacy area of ADC Reference Library.
2003-02-01	Split this document out from Inside Mac OS X: Event Manager Reference.

Index