NSEvent

Inherits From:	NSObject
Conforms To:	NSCoding NSCopying NSObject (NSObject)
Declared In:	AppKit/NSEvent.h

Class Description

An NSEvent object, or simply an *event*, contains information about an input action such as a mouse click or a key down. The Application Kit associates each such user action with a window, reporting the event to the application that created the window. The NSEvent object contains pertinent information about each event, such as where the mouse was located or which character was typed. As the application receives events, it temporarily places them in a buffer called the *event queue*. When the application is ready to process an event, it takes one from the queue.

NSEvents are typically passed up the application's *responder chain*, a series of objects that stand in line for event messages and untargeted action messages, as described in the NSResponder class specification. When the NSApplication object retrieves an event from the event queue, it dispatches the event to the appropriate NSWindow by invoking **sendEvent:**. The NSWindow then passes the event to its first responder in an event message such as **mouseDown:** or **keyDown:**, and the event gets passed on up the responder chain until some object handles it. In the case of a mouse-down event, a **mouseDown:** message is sent to the NSView where the user clicked the mouse; if it doesn't handle the event itself, the NSView relays the message to its next responder.

Most events follow this same path: from the windowing system to the application's event queue, and from there to the appropriate objects in the application. Though it rarely need do so, an application can also create an event from scratch and insert it into the event queue for distribution, or send it directly to its destination in an event message. The newly created events can be added to the event queue by invoking NSWindow's (or NSApplication's) **postEvent:atStart:** method.

While most events are distributed automatically through the responder chain, sometimes an object needs to retrieve events explicitly—for example, while handling mouse-dragged events. NSWindow and NSApplication define the method **nextEventMatchingMask:untilDate:inMode:dequeue:**, which allows an object to retrieve events of specific types. The nature of the retrieved event can then be ascertained by invoking NSEvent instance methods—**type**, **window**, and so on. All types of events are associated with an NSWindow; the **window** method returns this object. The location of a mouse event within the window's coordinate system is given by **locationInWindow**, and the time of the event by **timestamp**. The

modifierFlags method returns an indication of which modifier keys (Command, Control, Shift, and so on) the user held down while the event occurred.

The **type** method returns an NSEventType value that identifies the sort of event. The different types of events fall into five groups:

- Keyboard events
- Mouse events
- Tracking-rectangle and cursor-update events
- Periodic events
- Other events

Some of these groups comprise several NSEventType constants, others only one. The following sections discuss the groups, along with the corresponding NSEventType constants.

Keyboard Events

Among the most common events sent to an application are direct reports of the user's keyboard actions, identified by these NSEventType constants:

- NSKeyDown. The user generated a character or characters by pressing a key.
- NSKeyUp. The key was released.
- NSFlagsChanged. The user pressed or released a modifier key, or turned Alpha Lock on or off.

Of these, key-down events are the most useful to an application. When a **type** message returns NSKeyDown, the next step is typically to get the characters generated by the key-down using the **characters** method.

Key-up events are used less frequently since they follow almost automatically when there's been a key-down event. And because NSEvent's **modifierFlags** method returns the state of the modifier keys regardless of the type of event, applications normally don't need to receive flags-changed events; they're useful only for applications that have to keep track of the state of these keys at all times.

For more information on keyboard events, see "Key Events" under the Class Description in the NSResponder class specification.

Mouse Events

Mouse events are generated by changes in the state of the mouse buttons and by changes in the position of the mouse cursor on the screen. This category consists of:

- NSLeftMouseDown, NSLeftMouseUp, NSRightMouseDown, NSRightMouseUp. "Mouse-down" means the user pressed the button; "mouse-up" means the user released it. If the mouse has just one button, only left mouse events are generated. By sending a **clickCount** message to the event, you can determine whether the mouse event was a single click, double click, and so on.

- NSLeftMouseDragged, NSRightMouseDragged. The user moved the mouse with one or more buttons down. NSLeftMouseDragged events are generated when the mouse is moved with its left mouse button down or with both buttons down, and NSRightMouseDragged when it's moved with just the right button down. A mouse with a single button generates only left mouse-dragged events. A series of mouse-dragged events is always preceded by a mouse-down event and followed by a mouse-up event.
- NSMouseMoved. The user moved the mouse without holding down either mouse button. Mouse-moved events are normally not tracked, as they quickly flood the event queue; use NSWindow's **setAcceptsMouseMovedEvents:** to turn on tracking of mouse movements.

Mouse-dragged and mouse-moved events are generated repeatedly as long as the user keeps moving the mouse. If the mouse is stationary, neither type of event is generated until the mouse moves again.

Note: Neither the OpenStep specification nor NeXT's OPENSTEP implementation specifies facilities for the third button of a three-button mouse.

See "Mouse Events" under "Event Handling" in the NSView class specification for more information on mouse events.

Tracking-Rectangle and Cursor-Update Events

Because following the mouse's movements precisely is an expensive operation, the Application Kit provides a less intensive mechanism for tracking the location of the mouse. It does this by allowing the application to define regions of the screen, called *tracking rectangles*, that generate events when the cursor enters or leaves them. The event types are NSMouseEntered and NSMouseExited, and they're generated when the application has asked the Window Server to set a tracking rectangle in a window, typically by using NSView's **addTrackingRect:owner:userData:assumeInside:** method. A window can have any number of tracking rectangles; NSEvent's **trackingNumber** method identifies the rectangle that triggered the event.

A special kind of tracking event is the NSCursorUpdate event. This type is used to implement NSView's cursor-rectangle mechanism. An NSCursorUpdate event is generated when the cursor has crossed the boundary of a predefined rectangular area. Applications rarely use NSCursorUpdate events directly, instead using NSView's far more convenient methods.

See "Tracking Rectangles and Cursor Rectangles" under "Event Handling" in the NSView class specification for more information.

Periodic Events

An event of type NSPeriodic simply notifies an application that a certain time interval has elapsed. By using the NSEvent class method **startPeriodicEventsAfterDelay:withPeriod:**, an application can register to receive periodic events and have them placed in its event queue at a certain frequency. When the application no longer needs them, the flow of periodic events can be turned off by invoking **stopPeriodicEvents**. An application can have only one stream of periodic events active for each thread. Unlike keyboard and mouse

events, periodic events aren't dispatched to an NSWindow. The application must retrieve them explicitly using **nextEventMatchingMask:untilDate:inMode:dequeue:**, typically in a modal loop.

Periodic events are particularly useful in situations where input events aren't generated. For example, when the user holds the mouse down over a scroll button but doesn't move it, no events are generated after the mouse-down event. The scrolling mechanism then has to start and use a stream of periodic events to keep the document scrolling at a reasonable pace until the user releases the mouse. When a mouse-up event occurs, the scrolling mechanism terminates the periodic event stream.

Other Events

The remaining event types—NSAppKitDefined, NSSystemDefined, and NSApplicationDefined—are less structured, containing only generic subtype and data fields. These three types are extensions to the OpenStep specification, so you shouldn't use them in portable code (periodic events are also implemented in this manner, but are in the specification). Of the three miscellaneous event types, only NSApplicationDefined is of real use to application programs. It allows the application to generate totally custom events and insert them into the event queue. Each such event can have a subtype and two additional codes to different it from others. **otherEventWithType:...** creates one of these events, and the **subtype**, **data1**, and **data2** methods return the information specific to these events.

Adopted Protocols

NSCoding	– encodeWithCoder: – initWithCoder:
NSCopying	– copyWithZone: – copy

Method Types

Creating events

+ keyEventWithType:location:modifierFlags:timestamp: windowNumber:context:characters: charactersIgnoringModifier:isARepeat:keyCode:
+ mouseEventWithType:location:modifierFlags: timestamp:windowNumber:context:eventNumber: clickCount:pressure:
+ enterExitEventWithType:location:modifierFlags: timestamp:windowNumber:context:eventNumber: trackingNumber:userData:
+ otherEventWithType:location:modifierFlags: timestamp:windowNumber:context:subtype:data1: data2:

Requesting and stopping periodic events

+ startPeriodicEventsAfterDelay:withPeriod:
+ stopPeriodicEvents

Getting general event information

– context
– locationInWindow
– modifierFlags
– timestamp
– type
– window
– windowNumber

Getting key event information

– characters
– charactersIgnoringModifiers
– isARepeat
– keyCode

Getting mouse event information

– clickCount
– eventNumber
– pressure

Getting tracking-rectangle event information

– eventNumber
– trackingNumber
– userData

Getting custom event information

– data1
– data2
– subtype

Class Methods

enterExitEventWithType:location:modifierFlags:timestamp: windowNumber:context:eventNumber:trackingNumber:userData:

+ (NSEvent *)**enterExitEventWithType:**(NSEventType)*type*
location:(NSPoint)*location*
modifierFlags:(unsigned int)*flags*
timestamp:(NSTimeInterval)*time*
windowNumber:(int)*windowNumber*
context:(NSDPSContext *)*context*
eventNumber:(int)*eventNumber*
trackingNumber:(int)*trackingNumber*
userData:(void *)*userData*

Returns a new NSEvent object describing a tracking-rectangle or cursor-update event. *type* must be one of the following, else an NSInvalidArgumentException is raised:

NSMouseEntered
NSMouseExited
NSCursorUpdate

location, *flags*, *time*, *windowNumber*, and *context* are as described under **keyEventWithType:...**. Arguments specific to mouse tracking events are:

eventNumber is an identifier for the new event. It's normally taken from a counter for mouse events, which continually increases as the application runs.

trackingNumber is the number that identifies the tracking rectangle. This identifier is the same returned by NSView's **addTrackingRect:owner:userData:assumeInside:**.

userData is data arbitrarily associated with the tracking rectangle when it was set up using NSView's **addTrackingRect:owner:userData:assumeInside:**.

See also: – **eventNumber**, – **trackingNumber**, – **userData**

keyEventWithType:location:modifierFlags:timestamp:windowNumber: context:characters:charactersIgnoringModifier:isARepeat:keyCode:

+ (NSEvent *)**keyEventWithType:**(NSEventType)*type*
location:(NSPoint)*location*
modifierFlags:(unsigned int)*flags*
timestamp:(NSTimeInterval)*time*
windowNumber:(int)*windowNum*
context:(NSDPSContext *)*context*
characters:(NSString *)*characters*
charactersIgnoringModifiers:(NSString *)*unmodCharacters*
isARepeat:(BOOL)*repeatKey*
keyCode:(unsigned short int)*code*

Returns a new NSEvent object describing a key event. *type* must be one of the following, else an NSInvalidArgumentException is raised:

NSKeyDown
NSKeyUp
NSFlagsChanged

location is the mouse location in the base coordinate system of the window specified by *windowNumber*.

flags is an integer bit field containing any of these modifier key masks, combined using the C bitwise OR operator:

NSAlphaShiftKeyMask
NSShiftKeyMask
NSControlKeyMask
NSAlternateKeyMask
NSCommandKeyMask
NSNumericPadKeyMask
NSHelpKeyMask
NSFunctionKeyMask

time is the time the event occurred in seconds since system startup. How to get this value varies with the platform.

windowNumber identifies the PostScript window device associated with the event, which is associated with the NSWindow that will receive the event.

context is the Display PostScript context of the event.

characters is a string of characters associated with the key event. Though most key events contain only one character, it is possible for a single keypress to generate a series of characters.

unmodCharacters is the string of characters generated by the key event as if no modifier key had been pressed (except for Shift). This is useful for getting the "basic" key value in a hardware-independent manner.

repeatKey is YES if the key event is a repeat caused by the user holding the key down, NO if the key event is new.

code identifies the keyboard key associated with the key event. Its value is hardware-dependent.

See also: – **characters**, – **charactersIgnoringModifiers**, – **isARepeat**, – **keyCode**

mouseEventWithType:location:modifierFlags:timestamp: windowNumber:context:eventNumber:clickCount:pressure:

+ (NSEvent *)**mouseEventWithType:**(NSEventType)*type*
location:(NSPoint)*location*
modifierFlags:(unsigned int)*flags*
timestamp:(NSTimeInterval)*time*
windowNumber:(int)*windowNum*
context:(NSDPSContext *)*context*
eventNumber:(int)*eventNumber*
clickCount:(int)*clickNumber*
pressure:(float)*pressure*

Returns a new NSEvent object describing a mouse-down, -up, -moved, or -dragged event. *type* must be one of the following, else an NSInvalidArgumentException is raised:

NSLeftMouseDown
NSLeftMouseUp
NSRightMouseDown
NSRightMouseUp
NSMouseMoved
NSLeftMouseDragged
NSRightMouseDragged

location, *flags*, *time*, *windowNumber*, and *context* are as described under **keyEventWithType:...**.

eventNumber is an identifier for the new event. It's normally taken from a counter for mouse events, which continually increases as the application runs.

clickNumber is the number of mouse clicks associated with the mouse event.

pressure is a value from 0.0 to 1.0 indicating the pressure applied to the input device on a mouse event, used for an appropriate device such as a graphics tablet. For devices that aren't pressure-sensitive, the value

should be either 0.0 or 1.0. How to determine whether the input device is pressure-sensitive depends on the platform.

See also: – **clickCount**, – **eventNumber**, – **pressure**

otherEventWithType:location:modifierFlags:timestamp: windowNumber:context:subtype:data1:data2:

+ (NSEvent *)**otherEventWithType:**(NSEventType)*type*
location:(NSPoint)*location*
modifierFlags:(unsigned int)*flags*
timestamp:(NSTimeInterval)*time*
windowNumber:(int)*windowNum*
context:(NSDPSContext *)*context*
subtype:(short int)*subtype*
data1:(int)*data1*
data2:(int)*data2*

Returns a new NSEvent object describing a custom event. *type* must be one of the values below, else an NSInvalidArgumentException is raised. Your code should only create events of type NSApplicationDefined.

NSAppKitDefined (NeXT extension to the OpenStep specification)
NSSystemDefined (NeXT extension to the OpenStep specification)
NSApplicationDefined (NeXT extension to the OpenStep specification)
NSPeriodic

location, *flags*, *time*, *windowNumber*, and *context* are as described under **keyEventWithType:...**. Arguments specific to mouse tracking events are:

subtype further differentiates custom events of type NSAppKitDefined, NSSystemDefined, and NSApplicationDefined. NSPeriodic events don't use this attribute.

data1 and *data2* contain additional data associated with the event. NSPeriodic events don't use these attributes.

See also: – **subtype**, – **data1**, – **data2**

startPeriodicEventsAfterDelay:withPeriod:

+ (void)**startPeriodicEventsAfterDelay:**(NSTimeInterval)*delaySeconds*
withPeriod:(NSTimeInterval)*periodSeconds*

Begins generating periodic events for the current thread every *periodSeconds*, after a delay of *delaySeconds*. Raises an NSInternalInconsistencyException if periodic events are already being generated for the current thread. This method is typically used in a modal loop while tracking mouse-dragged events.

See also: + **stopPeriodicEvents**

stopPeriodicEvents

+ (void)**stopPeriodicEvents**

Stops generating periodic events for the current thread and discards any periodic events remaining in the queue. This message is ignored if periodic events aren't currently being generated.

See also: + **startPeriodicEventsAfterDelay:withPeriod:**

Instance Methods

characters

– (NSString *)**characters**

Returns the characters associated with the receiving key-up or key-down event. These characters are derived from a keyboard mapping that associates various key combinations with Unicode characters. Raises an NSInternalInconsistencyException if sent to any other kind of event.

See also: – **charactersIgnoringModifiers**, + **keyEventWithType:...**

charactersIgnoringModifiers

– (NSString *)**charactersIgnoringModifiers**

Returns the characters generated by the receiving key event as if no modifier key (except for Shift) applies. Raises an NSInternalInconsistencyException if sent to a non-key event. The return value of this method is meaningless for an NSFlagsChanged event.

This method is useful for determining "basic" key values in a hardware-independent manner, enabling such features as keyboard equivalents and mnemonics defined in terms of modifier keys plus character keys. For example, to determine if the user typed Alt-s, you don't have to know whether Alt-s generates a German

double ess, an integral sign, or a section symbol. You simply examine the string returned by this method along with the event's modifier flags, checking for "s" and NSAlternateKeyMask.

See also: – **characters**, – **modifierFlags**, + **keyEventWithType:...**

clickCount

– (int)**clickCount**

Returns the number of mouse clicks associated with the receiver, a mouse-down or -up event. Raises an NSInternalInconsistencyException if sent to a non-mouse event.

The return value of this method is meaningless for events other than mouse-down or -up events.

See also: + **mouseEventWithType:...**

context

– (NSDPSContext *)**context**

Returns the Display PostScript context of the receiving event.

data1

– (int)**data1**

Returns additional data associated with the receiving event. Raises an NSInternalInconsistencyException if sent to an event not of type NSAppKitDefined, NSSystemDefined, NSApplicationDefined, or NSPeriodic.

NSPeriodic events don't use this attribute.

See also: – **data2**, – **subtype**, + **otherEventWithType:...**

data2

– (int)**data2**

Returns additional data associated with the receiving event. Raises an NSInternalInconsistencyException if sent to an event not of type NSAppKitDefined, NSSystemDefined, NSApplicationDefined, or NSPeriodic.

NSPeriodic events don't use this attribute.

See also: – **data1**, – **subtype**, + **otherEventWithType:...**

eventNumber

– (int)**eventNumber**

Returns the counter value of the latest mouse or tracking-rectangle event; every system-generated mouse and tracking-rectangle event increments this counter. Raises an NSInternalInconsistencyException if sent to any other type of event.

See also: + **enterExitEventWithType:...**,
+ **mouseEventWithType:...**

isARepeat

– (BOOL)**isARepeat**

Returns YES if the receiving key event is a repeat caused by the user holding the key down, NO if the key event is new. Raises an NSInternalInconsistencyException if sent to a non-key event.

The return value of this method is meaningless for NSFlagsChanged events.

See also: + **keyEventWithType:...**

keyCode

– (unsigned short int)**keyCode**

Returns the code for the keyboard key associated with the receiving key event. Its value is hardware-dependent. Raises an NSInternalInconsistencyException if sent to a non-key event.

See also: + **keyEventWithType:...**

locationInWindow

– (NSPoint)**locationInWindow**

Returns the receiving event's location in the base coordinate system of the associated window.

See also: – **window**

modifierFlags

– (unsigned int)**modifierFlags**

Returns an integer bit field indicating the modifier keys in effect for the receiving event. You can examine individual flag settings using the C bitwise AND operator with these predefined masks:

NSAlphaShiftKeyMask
NSShiftKeyMask
NSControlKeyMask
NSAlternateKeyMask
NSCommandKeyMask
NSNumericPadKeyMask
NSHelpKeyMask
NSFunctionKeyMask

pressure

– (float)**pressure**

Returns a value between 0.0 and 1.0 indicating the pressure applied to the input device (used for appropriate devices). For devices that aren't pressure-sensitive, the value is either 0.0 or 1.0. How to determine whether the input device is pressure-sensitive depends on the platform. Raises an NSInternalInconsistencyException if sent to a non-mouse event.

See also: + **mouseEventWithType:...**

subtype

– (short int)**subtype**

Returns the subtype of the receiving custom event. Raises an NSInternalInconsistencyException if sent to an event not of type NSAppKitDefined, NSSystemDefined, NSApplicationDefined, or NSPeriodic.

NSPeriodic events don't use this attribute.

See also: – **data1**, – **data2**, + **otherEventWithType:...**

timestamp

– (NSTimeInterval)**timestamp**

Returns the time the event occurred in seconds since system startup.

trackingNumber

– (int)**trackingNumber**

Returns the identifier of the tracking rectangle for a tracking-rectangle event. Raises an NSInternalInconsistencyException if sent to any other type of event.

See also: + **enterExitEventWithType:...**

type

– (NSEventType)**type**

Returns the type of the receiving event, one of:

NSLeftMouseDown	NSKeyDown
NSLeftMouseUp	NSKeyUp
NSRightMouseDown	NSFlagsChanged
NSRightMouseUp	NSAppKitDefined (NeXT extension to the OpenStep specification)
NSMouseMoved	NSSystemDefined (NeXT extension to the OpenStep specification)
NSLeftMouseDragged	NSApplicationDefined (NeXT extension to the OpenStep specification)
NSRightMouseDragged	NSPeriodic
NSMouseEntered	NSCursorUpdate
NSMouseExited	

userData

– (void *)**userData**

Returns data associated with a tracking-rectangle event, assigned to the tracking rectangle when it was set up using NSView's **addTrackingRect:owner:userData:assumeInside:**. Raises an NSInternalInconsistencyException if sent to any other type of event.

See also: + **enterExitEventWithType:...**

window

– (NSWindow *)**window**

Returns the window object associated with the event. A periodic event, however, has no window; in this case the return value is undefined.

See also: – **windowNumber**

windowNumber

– (int)**windowNumber**

Returns the identifier for the PostScript window device associated with the event. A periodic event, however, has no window; in this case the return value is undefined.

See also: – **window**