

1. Document Object Model Events

Editors:

[Björn Höhrmann](#)

Philippe Le Hégaré, W3C (until November 2003)

Tom Pixley, Netscape Communications Corporation (until July 2002)

Table of Contents

[1.1 Introduction](#)

[1.1.1 Conformance](#)

[1.2 Event dispatch and DOM event flow](#)

[1.3 Default actions and cancelable events](#)

[1.4 Activation requests and behavior](#)

[1.5 Event types](#)

[1.5.1 Complete list of event types](#)

[1.6 Basic interfaces](#)

[Event](#), [CustomEvent](#), [EventTarget](#), [EventListener](#), [EventException](#), [EventExceptionCode](#)

[1.6.1 Event creation](#)

- [DocumentEvent](#)

[1.7 Event module definitions](#)

[1.7.1 User Interface event types](#)

- [UIEvent](#)

[1.7.2 Text events types](#)

- [TextEvent](#)

[1.7.3 Keyboard event types](#)

- [KeyboardEvent](#)

[1.7.4 Mouse event types](#)

- [MouseEvent](#)

[1.7.5 Mouse multi wheel event types](#)

- [MouseMultiWheelEvent](#)

[1.7.6 Mouse wheel event types](#)

- [MouseWheelEvent](#)

[1.7.7 Mutation event types](#)

- [MutationEvent](#)

[1.7.8 Mutation name event types](#)

- [MutationNameEvent](#)

[1.7.9 Basic event types](#)

1.1 Introduction

DOM Events is designed with two main goals. The first goal is the design of an [event](#) system which allows registration of event listeners and describes event flow through a tree structure. Additionally, the specification will provide standard modules of events for user interface control and document mutation notifications, including defined contextual information for each of these event modules.

The second goal of DOM Events is to provide a common subset of the current event systems used in

[DOM Level 0](#) browsers. This is intended to foster interoperability of existing scripts and content. It is not expected that this goal will be met with full backwards compatibility. However, the specification attempts to achieve this when possible.

1.1.1 Conformance

This specification is to be understood in the context of the DOM Level 3 Core specification [[DOM Level 3 Core](#)] and the general considerations for DOM implementations apply. For example, handling of [namespace URIs](#) is discussed in [XML Namespaces](#), and behavior in exceptional circumstances (such as when a `null` argument is passed when `null` was not expected) is discussed under [DOMException](#). For additional information about [conformance](#), please see the DOM Level 3 Core specification [[DOM Level 3 Core](#)].

An implementation is DOM Level 3 Events conformant if it supports the Core module defined in [[DOM Level 2 Core](#)], the [Event dispatch and DOM event flow](#) mechanism and the interfaces with their associated semantics defined in [Basic interfaces](#). An implementation conforms to a DOM Level 3 Events module if it conforms to DOM Level 3 Events, the event types defined in the module, and the modules the module depends upon (if any).

An implementation conforms to an event type if it conforms to its associated semantics and DOM interfaces. This includes that event objects that are generated by the implementation are generated as outlined in the tabular definition of the event type.

An implementation which does not conform to an event module can still implement the DOM interfaces associated with it. The DOM application can then create an event object using the [DocumentEvent.createEvent\(\)](#) method and dispatch an event type associated with this interface using the [EventTarget.dispatchEvent\(\)](#) method.

A DOM application may use the `hasFeature(feature, version)` method of the `DOMImplementation` interface with parameter values "Events" and "3.0" (respectively) to determine whether or not DOM Level 3 Events is supported by the implementation. Since DOM Level 3 Events is built on top of DOM Level 2 Events [[DOM Level 2 Events](#)], an implementation that returns `true` for "Events" and "3.0" will also return `true` for the parameters "Events" and "2.0". The same holds for the feature strings defined for the individual event modules as applicable. Refer to [DOM Features](#) in [[DOM Level 3 Core](#)] for additional information.

1.2 Event dispatch and DOM event flow

This section defines the event dispatch mechanism of the event model defined in this specification. Applications may dispatch event objects using the [EventTarget.dispatchEvent\(\)](#) method, and implementations must dispatch event objects as if through this method. The behavior of this method depends on the *event flow* associated with the underlying object. An event flow describes how event objects *propagate* through a data structure. As an example, when an event object is dispatched to an element in an XML document, the object propagates through parts of the document, as determined by the DOM event flow which is defined at the end of this section.

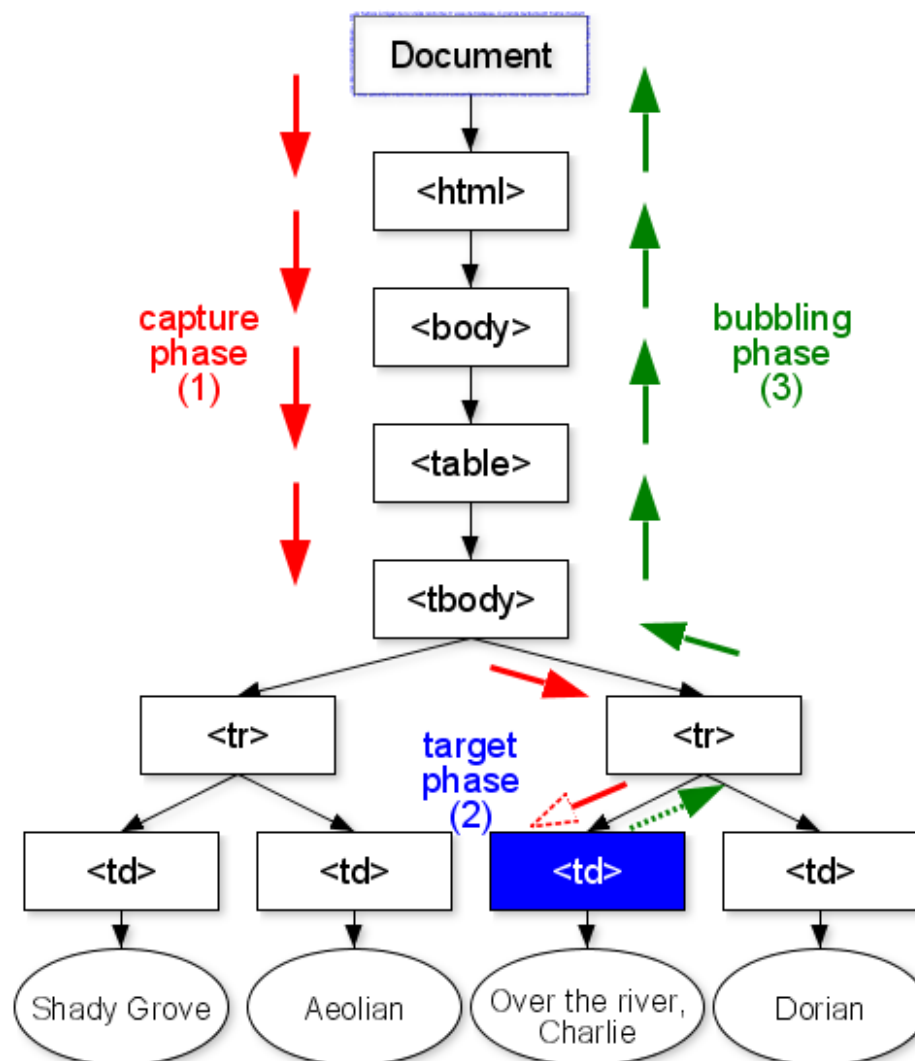


Figure: graphical representation of an event dispatched in a DOM tree using the DOM event flow [SVG 1.0 version]

Event objects are always dispatched to an event target called the *event's target*. At the beginning of the dispatch, implementations must first determine the event object's *propagation path*. This is an ordered list of event targets the object may propagate to. The last item in the list is the event's target; the preceding items in the list are referred to as the *target's ancestors* and the immediately preceding item as the *target's parent*. Once determined, the propagation path cannot be changed. As an example, in the DOM event flow event listeners might change the position of the target node in the document while the event object is being dispatched; such changes do not affect the propagation path.

As the next step the event object *accomplishes* one or more *event phases*. This specification defines the following event phases. Event objects accomplish them in the specified order using the partial propagation paths as defined below. A phase is skipped if it is not supported, or if the event object's propagation has been stopped. For example, if the `Event.bubbles` attribute is set to false, the bubble phase is skipped, and if `Event.stopPropagation()` has been called prior to the dispatch, all phases will be skipped.

1. The *capture phase*: the event object propagates through the target's ancestors to the target's parent. This phase is also known as the *capturing phase*. Event listeners registered for this phase can handle the event before it reaches its target.
2. The *target phase*: the event object has reached the event's target. This phase is also known as the *at-target phase*. Event listeners registered for this phase can handle the event once it has reached its target.

3. The *bubble phase*: the event object propagates through the target's ancestors in reverse order, starting with the target's parent. This phase is also known as the *bubbling phase*. Event listeners registered for this phase can handle the event after it has reached its target.

Implementations let event objects accomplish an event phase by applying the following steps while there are pending event targets in the partial propagation path for this phase and the event object's propagation has not been stopped through [Event.stopPropagation\(\)](#).

Firstly, the implementation must determine the *current target*. This is the next pending event target in the partial propagation path, starting with the first. From the perspective of an event listener this is the event target the listener has been registered on.

Secondly, the implementation must determine the current target's *candidate event listeners*. This is the list of all event listeners that have been registered on the current target in their order of registration. Once determined, the candidate event listeners cannot be changed, adding or removing listeners does not affect the current target's candidate event listeners.

Finally, the implementation must process all candidate event listeners in order and trigger each listener if all the following conditions are met. A listener is triggered by invoking the [EventListener.handleEvent\(\)](#) method or an equivalent binding-specific mechanism.

- The event object's immediate propagation has not been stopped.
- The listener has been registered for this event phase.
- The listener has been registered for this event type.

As the final step of the event dispatch, for reasons of backwards compatibility, the implementation must reset the event object's internal propagation and default action prevention states. This ensures that an event object may be properly dispatched multiple times while also allowing to prevent the event objects propagation or default actions prior to the event dispatch.

The model defined above applies regardless of the specific event flow associated with an event target. Each event flow defines how the propagation path is determined and which event phases are supported. The *DOM event flow* is an application of this model: the propagation path for a `Node` object is determined by its `Node.parentNode` chain, and all events accomplish the capture and target phases. Whether an event accomplishes the bubble phase is defined individually for each event type. An alternate application of this model can be found in [[DOM Level 3 Load and Save](#)].

The DOM event model is reentrant. Event listeners may perform actions that cause additional events to be dispatched. Such events are handled in a synchronous manner, the event propagation that causes the event listener to be triggered will resume only after the event dispatch of the new event is completed.

1.3 Default actions and cancelable events

Implementations may have a default action associated with an event type. An example is the [[HTML 4.01](#)] form element. When the user submits the form (e.g. by pressing on a submit button), the event [submit](#) is dispatched to the element and the default action for this event type is generally to send a request to a Web server with the parameters from the form.

The default actions are not part of the DOM Event flow. Before invoking a default action, the implementation must first dispatch the event as described in the [Event dispatch and DOM event flow](#).

A *cancelable event* is an event associated with a default action which is allowed to be canceled during the DOM event flow. At any phase during the event flow, the triggered event listeners have the option of canceling the default action or allowing the default action to proceed. In the case of the hyperlink in the browser, canceling the action would have the result of not activating the hyperlink. Not all events defined in this specification are cancelable events. See also [Default actions and cancelable keyboard](#)

events.

Different implementations will specify their own default actions, if any, associated with each event. The DOM Events specification does not attempt to specify these actions.

This specification does not provide mechanisms for accessing default actions or adding new ones.

Implementations could react to an event before dispatching it and make changes to the display and the DOM tree. Preventing the default action of such an event will reverse these changes. A good example is the attribute `HTMLInputElement.checked`: as described in [[DOM Level 2 HTML](#)], the value of this attribute may be changed before the dispatch of the event; the user clicks on the radio button, the radio button is being checked (or unchecked) on the display, the attribute `HTMLInputElement.checked` is changed as well, and then the device event type [click](#) is being dispatched. If the default action of the device event type is prevented, or if the default action attached to the [DOMActivate](#) event type is prevented, the attribute `HTMLInputElement.checked` will need to be changed back to its original value.

1.4 Activation requests and behavior

(This section is currently being rewritten.)

Event targets may have associated *activation behavior* that implementations perform in response to an *activation request*. As an example, the typical activation behavior associated with hyperlinks is to follow the link. Activation requests are typically initiated by users through an input device.

In terms of this specification, the activation behavior of the event target is the default action of the event type [DOMActivate](#). DOM applications should use this event type whenever they wish to make or react to an activation request.

Implementations dispatch the `DOMActivate` event as default action of a [click](#) event. This click event is either part of the activation request (e.g., a user requests activation using a mouse), or synthesized by the implementation to accommodate legacy applications. Context information of such a `click` event is implementation dependent.

When implementations dispatch a synthesized `click` event, the expectation is that they do so as default action of another event type. For example, when a user activates a hyperlink using a keyboard, the `click` event would be dispatched as default action of the respective keyboard event.

Implementations are required to dispatch the synthesized `click` event as described above even if they do not dispatch such an event (e.g., when activation is requested by a voice command, since this specification does not address event types for voice input).

Note: The activation of an event target is device dependent but is also application dependent, e.g. a link in a document can be activated using a mouse click or a mouse double click.

1.5 Event types

Each event is associated with a type, called *event type*. The event type is composed of a [local name](#) and a [namespace URI](#) as used in [[DOM Level 3 Core](#)]. All events defined in this specification are in no namespace.

1.5.1 Complete list of event types

Depending on the level of DOM support, or the devices used for display (e.g. screen) or interaction (e.g., mouse, keyboard, touch screen, or voice), these event types can be generated by the

implementation. When used with an [[XML 1.0](#)] or [[HTML 4.01](#)] application, the specifications of those languages may restrict the semantics and scope (in particular the possible target nodes) associated with an event type. Refer to the specification defining the language used in order to find those restrictions or to find event types that are not defined in this document.

The following table provides a non-normative summary of the event types defined in this specification. All event types are in no namespace and this specification refers to them by their local name only. All events will accomplish the capture and target phases, but not all of them will accomplish the bubbling phase (see also [Event dispatch and DOM event flow](#)). Some events are not [cancelable](#) (see [Default actions and cancelable events](#)). Some events will only be dispatched to a specific set of possible targets, specified using node types. Contextual information related to the event type is accessible using DOM interfaces.

type	Bubbling phase	Cancelable	Target node types	DOM interface
DOMActivate	Yes	Yes	Element	UIEvent
DOMFocusIn	Yes	No	Element	UIEvent
DOMFocusOut	Yes	No	Element	UIEvent
focus	No	No	Element	UIEvent
blur	No	No	Element	UIEvent
textInput	Yes	Yes	Element	TextEvent
click	Yes	Yes	Element	MouseEvent
dblclick	Yes	Yes	Element	MouseEvent
mousedown	Yes	Yes	Element	MouseEvent
mouseup	Yes	Yes	Element	MouseEvent
mouseover	Yes	Yes	Element	MouseEvent
mousemove	Yes	Yes	Element	MouseEvent
mouseout	Yes	Yes	Element	MouseEvent
keydown	Yes	Yes	Element	KeyboardEvent
keyup	Yes	Yes	Element	KeyboardEvent
mousemultiwheel	Yes	Yes	Document, Element	MouseMultiWheelEvent
mousewheel	Yes	Yes	Document, Element	MouseWheelEvent
DOMSubtreeModified	Yes	No	Document, DocumentFragment, Element, Attr	MutationEvent
DOMNodeInserted	Yes	No	Element, Attr, Text, Comment, CDATASection, DocumentType, EntityReference, ProcessingInstruction	MutationEvent
DOMNodeRemoved	Yes	No	Element, Attr, Text, Comment, CDATASection, DocumentType, EntityReference, ProcessingInstruction	MutationEvent
DOMNodeRemovedFromDocument	No	No	Element, Attr, Text, Comment, CDATASection, DocumentType, EntityReference, ProcessingInstruction	MutationEvent
DOMNodeInsertedIntoDocument	No	No	Element, Attr, Text, Comment, CDATASection, DocumentType, EntityReference, ProcessingInstruction	MutationEvent
DOMAttrModified	Yes	No	Element	MutationEvent

DOMCharacterDataModified	Yes	No	Text, Comment, CDATASection, ProcessingInstruction	MutationEvent
DOMElementNameChanged	Yes	No	Element	MutationNameEvent
DOMAttributeNameChanged	Yes	No	Element	MutationNameEvent
load	No	No	Document, Element	Event
unload	No	No	Document, Element	Event
abort	Yes	No	Element	Event
error	Yes	No	Element	Event
select	Yes	No	Element	Event
change	Yes	No	Element	Event
submit	Yes	Yes	Element	Event
reset	Yes	Yes	Element	Event
resize	Yes	No	Document, Element	UIEvent
scroll	Yes	No	Document, Element	UIEvent

As an example, the event [load](#) will trigger event listeners attached on `Element` nodes for that event and on the capture and target phases. This event cannot be cancelled. If an event listener for the [load](#) event is attached to a node other than `Document` or `Element` nodes, or if it is attached to the bubbling phase only, this event listener cannot be triggered.

The event objects associated with the event types described above may contain context information. Refer to the description of the DOM interfaces for further information.

1.6 Basic interfaces

The interfaces described in this section are fundamental to DOM Level 3 Events and must always be supported by the implementation. Together they define the feature Events 3.0.

Interface *Event* (introduced in DOM Level 2)

The `Event` interface is used to provide contextual information about an event to the listener processing the event. An object which implements the `Event` interface is passed as the parameter to an [EventListener](#). The object passed to the event listener may also implement derived interfaces that provide access to information directly relating to the type of event they represent.

To create an instance of the `Event` interface, use the [DocumentEvent.createEvent\("Event"\)](#) method call.

IDL Definition

```
// Introduced in DOM Level 2:
interface Event {

    // PhaseType
    const unsigned short    CAPTURING_PHASE           = 1;
    const unsigned short    AT_TARGET                 = 2;
    const unsigned short    BUBBLING_PHASE            = 3;

    readonly attribute DOMString    type;
    readonly attribute EventTarget target;
    readonly attribute EventTarget currentTarget;
    readonly attribute unsigned short eventPhase;
    readonly attribute boolean      bubbles;
    readonly attribute boolean      cancelable;
```

```

readonly attribute DOMTimeStamp    timeStamp;
void                               stopPropagation();
void                               preventDefault();
void                               initEvent(in DOMString eventTypeArg,
                                             in boolean canBubbleArg,
                                             in boolean cancelableArg);

// Introduced in DOM Level 3:
readonly attribute DOMString      namespaceURI;
// Introduced in DOM Level 3:
void                               stopImmediatePropagation();
// Introduced in DOM Level 3:
readonly attribute boolean        defaultPrevented;
// Introduced in DOM Level 3:
void                               initEventNS(in DOMString namespaceURIArg,
                                             in DOMString eventTypeArg,
                                             in boolean canBubbleArg,
                                             in boolean cancelableArg);

};

```

Definition group *PhaseType*

An integer indicating which phase of the event flow is being processed as defined in [Event dispatch and DOM event flow](#).

Defined Constants

AT_TARGET

The current event is in the [target phase](#), i.e. it is being evaluated at the [event target](#).

BUBBLING_PHASE

The current event phase is the [bubbling phase](#).

CAPTURING_PHASE

The current event phase is the [capture phase](#).

Attributes

bubbles of type **boolean**, **readonly**

Used to indicate whether or not an event is a bubbling event. If the event can bubble the value is `true`, otherwise the value is `false`.

cancelable of type **boolean**, **readonly**

Used to indicate whether or not an event can have its default action prevented (see also [Default actions and cancelable events](#)). If the default action can be prevented the value is `true`, otherwise the value is `false`.

currentTarget of type **EventTarget**, **readonly**

Used to indicate the [EventTarget](#) whose [EventListeners](#) are currently being processed. This is particularly useful during the capture and bubbling phases. This attribute could contain the [target node](#) or a target ancestor when used with the [Event dispatch and DOM event flow](#).

defaultPrevented of type **boolean**, **readonly**, **introduced in DOM Level 3**

Used to indicate whether [Event.preventDefault\(\)](#) has been called for this event.

eventPhase of type **unsigned short**, **readonly**

Used to indicate which phase of event flow is currently being accomplished.

namespaceURI of type **DOMString**, **readonly**, **introduced in DOM Level 3**

The [namespace URI](#) associated with this event at initialization time, or `null` if it is unspecified.

DOM Level 2 Events initialization methods, such as [Event.initEvent\(\)](#), set the value to `null`.

target of type **EventTarget**, **readonly**

Used to indicate the [event target](#). This attribute contains the [target node](#) when used with the [Event dispatch and DOM event flow](#).

timeStamp of type `DOMTimeStamp`, **readonly**

Used to specify the time at which the event was created in milliseconds relative to 1970-01-01T00:00:00Z. Due to the fact that some systems may not provide this information the value of `timeStamp` may be not available for all events. When not available, the value is 0.

type of type `DOMString`, **readonly**

The [local name](#) of the event type. The name must be an [NCName](#) as defined in [[XML Namespaces 1.1](#)] and is case-sensitive.

Methods**initEvent**

Initializes attributes of an `Event` created through the [DocumentEvent.createEvent](#) method. This method may only be called before the `Event` has been dispatched via the [EventTarget.dispatchEvent\(\)](#) method. If the method is called several times before invoking [EventTarget.dispatchEvent](#), only the final invocation takes precedence. This method has no effect if called after the event has been dispatched. If called from a subclass of the `Event` interface only the values specified in this method are modified, all other attributes are left unchanged.

This method sets the [Event.type](#) attribute to `eventTypeArg`, and [Event.namespaceURI](#) to `null`. To initialize an event with a namespace URI, use the [Event.initEventNS\(\)](#) method.

Parameters**eventTypeArg** of type `DOMString`

Specifies [Event.type](#), the [local name](#) of the event type.

canBubbleArg of type `boolean`

Specifies [Event.bubbles](#). This parameter overrides the intrinsic bubbling behavior of the event.

cancelableArg of type `boolean`

Specifies [Event.cancelable](#). This parameter overrides the intrinsic cancelable behavior of the event.

No Return Value**No Exceptions****initEventNS** introduced in **DOM Level 3**

Initializes attributes of an `Event` object. This method has the same behavior as [Event.initEvent\(\)](#).

Parameters**namespaceURIArg** of type `DOMString`

Specifies [Event.namespaceURI](#), the [namespace URI](#) associated with this event, or `null` if no namespace.

eventTypeArg of type `DOMString`

Refer to the [Event.initEvent\(\)](#) method for a description of this parameter.

canBubbleArg of type `boolean`

Refer to the [Event.initEvent\(\)](#) method for a description of this parameter.

cancelableArg of type `boolean`

Refer to the [Event.initEvent\(\)](#) method for a description of this parameter.

No Return Value**No Exceptions****preventDefault**

Signifies that the event is to be canceled, meaning any default action normally taken by the implementation as a result of the event will not occur (see also [Default actions and cancelable events](#)). Calling this method for a non-cancelable event has no effect.

Note: This method does not stop the event propagation; use [Event.stopPropagation\(\)](#) or [Event.stopImmediatePropagation\(\)](#) for that effect.

No Parameters
No Return Value
No Exceptions

stopImmediatePropagation introduced in DOM Level 3

Prevents other event listeners from being triggered and, unlike [Event.stopPropagation\(\)](#) its effect is immediate . Once it has been called, further calls to this method have no additional effect.

Note: This method does not prevent the default action from being invoked; use [Event.preventDefault\(\)](#) for that effect.

No Parameters
No Return Value
No Exceptions

stopPropagation

Prevents other event listeners from being triggered but its effect is deferred until all event listeners attached on the [Event.currentTarget](#) have been triggered . Once it has been called, further calls to this method have no additional effect.

Note: This method does not prevent the default action from being invoked; use [Event.preventDefault\(\)](#) for that effect.

No Parameters
No Return Value
No Exceptions

Interface *CustomEvent* (introduced in DOM Level 3)

The CustomEvent interface is the recommended interface for application-specific event types. Unlike the [Event](#) interface, it allows applications to provide contextual information about the event type. Application-specific event types should have an associated namespace to avoid clashes with future general-purpose event types.

To create an instance of the CustomEvent interface, use the [DocumentEvent.createEvent\("CustomEvent"\)](#) method call.

IDL Definition

```
// Introduced in DOM Level 3:
interface CustomEvent : Event {
  readonly attribute DOMObject      detail;
  void      initCustomEventNS(in DOMString namespaceURIArg,
                              in DOMString typeArg,
                              in boolean canBubbleArg,
                              in boolean cancelableArg,
                              in DOMObject detailArg);
};
```

Attributes

detail of type `DOMObject`, **readonly**
 Specifies some detail information about the [Event](#).

Methods

initCustomEventNS
 Initializes attributes of a CustomEvent object. This method has the same behavior as

[Event.initEventNS\(\)](#).

Parameters

namespaceURIArg of type DOMString

Refer to the [Event.initEventNS\(\)](#) method for a description of this parameter.

typeArg of type DOMString

Refer to the [Event.initEventNS\(\)](#) method for a description of this parameter.

canBubbleArg of type boolean

Refer to the [Event.initEventNS\(\)](#) method for a description of this parameter.

cancelableArg of type boolean

Refer to the [Event.initEventNS\(\)](#) method for a description of this parameter.

detailArg of type DOMObject

Specifies [CustomEvent.detail](#). This value may be `null`.

No Return Value

No Exceptions

Interface *EventTarget* (introduced in DOM Level 2)

The `EventTarget` interface is implemented by all the objects which could be [event targets](#) in an implementation which supports an event flow. The interface allows registration and removal of event listeners, and dispatch of events to an event target.

When used with the DOM event flow, this interface is implemented by all [target nodes](#) and target ancestors, i.e. all DOM `Nodes` of the tree support this interface when the implementation conforms to DOM Level 3 Events and, therefore, this interface can be obtained by using binding-specific casting methods on an instance of the `Node` interface.

Invoking `addEventListener` Or `addEventListenerNS` repeatedly on the same `EventTarget` with the same values for the parameters `namespaceURI`, `type`, `listener`, and `useCapture` has no effect. Doing so does not cause the [EventListener](#) to be called more than once and does not cause a change in the triggering order.

IDL Definition

```
// Introduced in DOM Level 2:
interface EventTarget {
    void          addEventListener(in DOMString type,
                                   in EventListener listener,
                                   in boolean useCapture);
    void          removeEventListener(in DOMString type,
                                       in EventListener listener,
                                       in boolean useCapture);

    // Modified in DOM Level 3:
    boolean       dispatchEvent(in Event evt)
                                   raises(EventException,
                                       DOMException);

    // Introduced in DOM Level 3:
    void          addEventListenerNS(in DOMString namespaceURI,
                                       in DOMString type,
                                       in EventListener listener,
                                       in boolean useCapture);

    // Introduced in DOM Level 3:
    void          removeEventListenerNS(in DOMString namespaceURI,
                                       in DOMString type,
                                       in EventListener listener,
                                       in boolean useCapture);
};
```

Methods

addEventListener

Registers an event listener, depending on the `useCapture` parameter, on the capture phase of the DOM event flow or its target and bubbling phases. Invoking this method is equivalent to invoking `addEventListenerNS` with the same values for the parameters `type`, `listener`, and `useCapture`, and the value `null` for the parameter `namespaceURI`.

Parameters

type of type `DOMString`

Specifies the [Event.type](#) associated with the event for which the user is registering.

listener of type `EventListener`

The `listener` parameter takes an object implemented by the user which implements the [EventListener](#) interface and contains the method to be called when the event occurs.

useCapture of type `boolean`

If `true`, `useCapture` indicates that the user wishes to add the event listener for the [capture phase](#) only, i.e. this event listener will not be triggered during the [target](#) and [bubbling](#) phases. If `false`, the event listener will only be triggered during the target and bubbling phases.

No Return Value

No Exceptions

addEventListenerNS introduced in DOM Level 3

Registers an event listener, depending on the `useCapture` parameter, on the capture phase of the DOM event flow or its target and bubbling phases.

Parameters

namespaceURI of type `DOMString`

Specifies the [Event.namespaceURI](#) associated with the event for which the user is registering.

type of type `DOMString`

Refer to the [EventTarget.addEventListener\(\)](#) method for a description of this parameter.

listener of type `EventListener`

Refer to the [EventTarget.addEventListener\(\)](#) method for a description of this parameter.

useCapture of type `boolean`

Refer to the [EventTarget.addEventListener\(\)](#) method for a description of this parameter.

No Return Value

No Exceptions

dispatchEvent modified in DOM Level 3

Dispatches an event into the implementation's event model. The [event target](#) of the event is the `EventTarget` object on which `dispatchEvent` is called.

Parameters

evt of type `Event`

The event to be dispatched.

Return Value

Indicates whether any of the listeners which handled the event called `Event.preventDefault()`. If `Event.preventDefault()` was called the returned value is `false`, else it is `true`.

Exceptions

UNSPECIFIED_EVENT_TYPE_ERR: Raised if the [Event.type](#) was not specified by initializing the event before `dispatchEvent` was called. Specification of the [Event.type](#) as `null` or an empty string will also trigger this exception.

DISPATCH_REQUEST_ERR: Raised if the [Event](#) object is already being dispatched.

NOT_SUPPORTED_ERR: Raised if the [Event](#) object has not been created using [DocumentEvent.createEvent\(\)](#).

DOMException

INVALID_CHARACTER_ERR: Raised if [Event.type](#) is not an [NCName](#) as defined in [[XML Namespaces 1.1](#)].

removeEventListener

Removes an event listener. Calling `removeEventListener` with arguments which do not identify any currently registered [EventListener](#) on the `EventTarget` has no effect. The [Event.namespaceURI](#) for which the user registered the event listener is implied and is `null`.

Parameters

type of type `DOMString`

Specifies the [Event.type](#) for which the user registered the event listener.

listener of type [EventListener](#)

The [EventListener](#) to be removed.

useCapture of type `boolean`

Specifies whether the [EventListener](#) being removed was registered for the capture phase or not. If a listener was registered twice, once for the capture phase and once for the target and bubbling phases, each must be removed separately. Removal of an event listener registered for the capture phase does not affect the same event listener registered for the target and bubbling phases, and vice versa.

No Return Value

No Exceptions

removeEventListenerNS introduced in DOM Level 3

Removes an event listener. Calling `removeEventListenerNS` with arguments which do not identify any currently registered [EventListener](#) on the `EventTarget` has no effect.

Parameters

namespaceURI of type `DOMString`

Specifies the [Event.namespaceURI](#) associated with the event for which the user registered the event listener.

type of type `DOMString`

Refer to the [EventTarget.removeEventListener\(\)](#) method for a description of this parameter.

listener of type [EventListener](#)

Refer to the [EventTarget.removeEventListener\(\)](#) method for a description of this parameter.

useCapture of type `boolean`

Refer to the [EventTarget.removeEventListener\(\)](#) method for a description of this parameter.

No Return Value

No Exceptions

Interface *EventListener* (introduced in DOM Level 2)

The `EventListener` interface is the primary way for handling events. Users implement the `EventListener` interface and register their event listener on an [EventTarget](#). The users should also remove their `EventListener` from its [EventTarget](#) after they have completed using the listener.

Copying a `Node`, with methods such as `Node.cloneNode` Or `Range.cloneContents`, does not copy the event listeners attached to it. Event listeners must be attached to the newly created `Node` afterwards if so desired.

Moving a `Node`, with methods `Document.adoptNode`, `Node.appendChild`, Or `Range.extractContents`, does not affect the event listeners attached to it.

IDL Definition

```
// Introduced in DOM Level 2:
interface EventListener {
    void          handleEvent(in Event evt);
};
```

Methods

handleEvent

This method is called whenever an event occurs of the event type for which the `EventListener` interface was registered.

Parameters

evt of type [Event](#)

The [Event](#) contains contextual information about the [event](#).

No Return Value

No Exceptions

Exception *EventException* introduced in DOM Level 2

Event operations may throw an [EventException](#) as specified in their method descriptions.

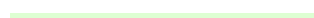
IDL Definition

```
// Introduced in DOM Level 2:
exception EventException {
    unsigned short  code;
};
// EventExceptionCode
const unsigned short    UNSPECIFIED_EVENT_TYPE_ERR    = 0;
// Introduced in DOM Level 3:
const unsigned short    DISPATCH_REQUEST_ERR        = 1;
```

Definition group *EventExceptionCode*

An integer indicating the type of error generated.

Defined Constants



DISPATCH_REQUEST_ERR, introduced in DOM Level 3.

If the [Event](#) object is already dispatched in the tree.

UNSPECIFIED_EVENT_TYPE_ERR

If the [Event.type](#) was not specified by initializing the event before the method was called. Specification of the [Event.type](#) as `null` or an empty string will also trigger this exception.

1.6.1 Event creation

In most cases, the events dispatched by the DOM Events implementation are also created by the implementation. It is however possible to simulate events such as mouse events by creating the [Event](#) objects and dispatch them using the DOM Events implementation.

Creating [Event](#) objects that are known to the DOM Events implementation is done using [DocumentEvent.createEvent\(\)](#). The application must then initialize the object by calling the appropriate initialization method before invoking [EventTarget.dispatchEvent\(\)](#). The [Event](#) objects created must be known by the DOM Events implementation; otherwise an event exception is thrown.

Interface *DocumentEvent* (introduced in DOM Level 2)

The `DocumentEvent` interface provides a mechanism by which the user can create an [Event](#) object of a type supported by the implementation. If the feature "Events" is supported by the `Document` object, the `DocumentEvent` interface must be implemented on the same object. If the feature "+Events" is supported by the `Document` object, an object that supports the `DocumentEvent` interface must be returned by invoking the method `Node.getFeature("+Events", "3.0")` on the `Document` object.

IDL Definition

```
// Introduced in DOM Level 2:
interface DocumentEvent {
    Event          createEvent(in DOMString eventType)
                               raises(DOMException);

    // Introduced in DOM Level 3:
    boolean        canDispatch(in DOMString namespaceURI,
                               in DOMString type);
};
```

Methods

canDispatch introduced in DOM Level 3

Tests if the implementation can generate events of a specified type.

Parameters

namespaceURI of type `DOMString`

Specifies the [Event.namespaceURI](#) of the event.

type of type `DOMString`

Specifies the [Event.type](#) of the event.

Return Value

`boolean` `true` if the implementation can generate and dispatch this event type,
`false` otherwise.

No Exceptions

createEvent

Creates an event object of the type specified.

Parameters**eventType** of type **DOMString**

The `eventType` parameter specifies the name of the DOM Events interface to be supported by the created event object, e.g. "Event", "MouseEvent", "MutationEvent" and so on. If the [Event](#) is to be dispatched via the [EventTarget.dispatchEvent\(\)](#) method the appropriate event initialization method must be called after creation in order to initialize the [Event](#)'s values.

As an example, a user wishing to synthesize some kind of [UIEvent](#) would invoke [DocumentEvent.createEvent\("UIEvent"\)](#). The [UIEvent.initUIEventNS\(\)](#) method could then be called on the newly created [UIEvent](#) object to set the specific type of user interface event to be dispatched, [DOMActivate](#) for example, and set its context information, e.g. [UIEvent.detail](#) in this example.

For backward compatibility reason, "UIEvents", "MouseEvents", "MutationEvents", and "HTMLEvents" feature names are valid values for the parameter `eventType` and represent respectively the interfaces "UIEvent", "MouseEvent", "MutationEvent", and "Event", and the characters 'a'..'z' are considered equivalent to the characters 'A'..'Z'.

Return Value

[Event](#) The newly created event object.

Exceptions

`DOMException` NOT_SUPPORTED_ERR: Raised if the implementation does not support the [Event](#) interface requested.

1.7 Event module definitions

The DOM Event Model allows a DOM implementation to support multiple modules of events. The model has been designed to allow addition of new event modules if required. The DOM will not attempt to define all possible events. For purposes of interoperability, the DOM defines a module of user interface events including lower level device dependent events and a module of document mutation events.

1.7.1 User Interface event types

This module defines the feature `UIEvents 3.0` and depends on the features `Events 3.0` and `Views 2.0`.

The User Interface event module contains basic event types associated with user interfaces.

Interface *UIEvent* (introduced in DOM Level 2)

The `UIEvent` interface provides specific contextual information associated with User Interface events.

To create an instance of the `UIEvent` interface, use the [DocumentEvent.createEvent\("UIEvent"\)](#)

method call.

IDL Definition

```
// Introduced in DOM Level 2:
interface UIEvent : Event {
  readonly attribute views::AbstractView view;
  readonly attribute long detail;
  void initUIEvent(in DOMString typeArg,
                  in boolean canBubbleArg,
                  in boolean cancelableArg,
                  in views::AbstractView viewArg,
                  in long detailArg);

  // Introduced in DOM Level 3:
  void initUIEventNS(in DOMString namespaceURIArg,
                    in DOMString typeArg,
                    in boolean canBubbleArg,
                    in boolean cancelableArg,
                    in views::AbstractView viewArg,
                    in long detailArg);
};
```

Attributes

detail of type **long**, **readonly**

Specifies some detail information about the [Event](#), depending on the type of event.

view of type **views::AbstractView**, **readonly**

The `view` attribute identifies the `AbstractView` from which the event was generated.

Methods

initUIEvent

Initializes attributes of an `UIEvent` object. This method has the same behavior as [Event.initEvent\(\)](#).

Parameters

typeArg of type **DOMString**

Refer to the [Event.initEvent\(\)](#) method for a description of this parameter.

canBubbleArg of type **boolean**

Refer to the [Event.initEvent\(\)](#) method for a description of this parameter.

cancelableArg of type **boolean**

Refer to the [Event.initEvent\(\)](#) method for a description of this parameter.

viewArg of type **views::AbstractView**

Specifies `UIEvent.view`. This value may be `null`.

detailArg of type **long**

Specifies `UIEvent.detail`.

No Return Value

No Exceptions

initUIEventNS introduced in **DOM Level 3**

Initializes attributes of an `UIEvent` object. This method has the same behavior as [Event.initEventNS\(\)](#).

Parameters

namespaceURIArg of type **DOMString**

Refer to the [Event.initEventNS\(\)](#) method for a description of this parameter.

typeArg of type **DOMString**

Refer to the [Event.initEventNS\(\)](#) method for a description of this parameter.

canBubbleArg of type boolean

Refer to the [Event.initEventNS\(\)](#) method for a description of this parameter.

cancelableArg of type boolean

Refer to the [Event.initEventNS\(\)](#) method for a description of this parameter.

viewArg of type `views::AbstractView`

Refer to the [UIEvent.initUIEvent\(\)](#) method for a description of this parameter.

detailArg of type long

Refer to the [UIEvent.initUIEvent\(\)](#) method for a description of this parameter.

No Return Value

No Exceptions

The User Interface event types are listed below.

DOMActivate

Type	DOMActivate
Namespace	None
Interface	UIEvent
Cancelable	Yes
Bubbles	Yes
Target	Element
Context info	UIEvent.view is in use.

Refer to [Activation requests and behavior](#).

DOMFocusIn

Type	DOMFocusIn
Namespace	None
Interface	UIEvent
Cancelable	No
Bubbles	Yes
Target	Element
Context info	UIEvent.view is in use.

An [event target](#) receives focus. The focus is given to the element before the dispatch of this event type. This event type is dispatched after the event type [focus](#).

DOMFocusOut

Type	DOMFocusOut
Namespace	None
Interface	UIEvent
Cancelable	No
Bubbles	Yes
Target	Element
Context info	UIEvent.view is in use.

An [event target](#) loses focus. The focus is taken from the element before the dispatch of this event type. This event type is dispatched after the event type [blur](#).

focus

Type	focus
Namespace	None
Interface	UIEvent
Cancelable	No
Bubbles	No
Target	Element
Context info	UIEvent.view is in use.

An [event target](#) receives focus. The focus is given to the element before the dispatch of this event type.

blur

Type	blur
Namespace	None
Interface	UIEvent
Cancelable	No
Bubbles	No
Target	Element
Context info	UIEvent.view is in use.

An [event target](#) loses focus. The focus is taken from the element before the dispatch of this event type.

1.7.2 Text events types

This module defines the feature TextEvents 3.0 and depends on the feature UIEvents 3.0.

The text event module originates from the [[HTML 4.01](#)] `onkeypress` attribute. Unlike this attribute, the event type [textInput](#) applies only to characters and is designed for use with any text input devices, not just keyboards. Refer to Appendix A, "[Keyboard events and key identifiers](#)", for examples on how text events are used in combination with keyboard events.

Interface *TextEvent* (introduced in DOM Level 3)

The `TextEvent` interface provides specific contextual information associated with Text Events.

To create an instance of the `TextEvent` interface, use the [DocumentEvent.createEvent\("TextEvent"\)](#) method call.

IDL Definition

```
// Introduced in DOM Level 3:
interface TextEvent : UIEvent {
  readonly attribute DOMString      data;
  void          initTextEvent(in DOMString typeArg,
                               in boolean canBubbleArg,
                               in boolean cancelableArg,
                               in views::AbstractView viewArg,
                               in DOMString dataArg);
  void          initTextEventNS(in DOMString namespaceURIArg,
                                in DOMString typeArg,
                                in boolean canBubbleArg,
                                in boolean cancelableArg,
```

```

};
        in views::AbstractView viewArg,
        in DOMString dataArg);

```

Attributes

data of type **DOMString**, **readonly**

`data` holds the value of the characters generated by the character device. This may be a single Unicode character or a non-empty sequence of Unicode characters [[Unicode](#)]. Characters should be normalized as defined by the Unicode normalization form *NFC*, defined in [[UAX #15](#)]. This attribute cannot be null or contain the empty string.

Methods

initTextEvent

Initializes attributes of a `TextEvent` object. This method has the same behavior as [UIEvent.initUIEvent\(\)](#). The value of [UIEvent.detail](#) remains undefined.

Parameters

typeArg of type **DOMString**

Refer to the [UIEvent.initUIEvent\(\)](#) method for a description of this parameter.

canBubbleArg of type **boolean**

Refer to the [UIEvent.initUIEvent\(\)](#) method for a description of this parameter.

cancelableArg of type **boolean**

Refer to the [UIEvent.initUIEvent\(\)](#) method for a description of this parameter.

viewArg of type **views::AbstractView**

Refer to the [UIEvent.initUIEvent\(\)](#) method for a description of this parameter.

dataArg of type **DOMString**

Specifies [TextEvent.data](#).

No Return Value

No Exceptions

initTextEventNS

Initializes attributes of a `TextEvent` object. This method has the same behavior as [UIEvent.initUIEventNS\(\)](#). The value of [UIEvent.detail](#) remains undefined.

Parameters

namespaceURIArg of type **DOMString**

Refer to the [UIEvent.initUIEventNS\(\)](#) method for a description of this parameter.

typeArg of type **DOMString**

Refer to the [UIEvent.initUIEventNS\(\)](#) method for a description of this parameter.

canBubbleArg of type **boolean**

Refer to the [UIEvent.initUIEventNS\(\)](#) method for a description of this parameter.

cancelableArg of type **boolean**

Refer to the [UIEvent.initUIEventNS\(\)](#) method for a description of this parameter.

viewArg of type **views::AbstractView**

Refer to the [UIEvent.initUIEventNS\(\)](#) method for a description of this parameter.

dataArg of type **DOMString**

Refer to the [TextEvent.initTextEvent\(\)](#) method for a description of this

parameter.

No Return Value
No Exceptions

The text event type is listed below.

textInput

Type	textInput
Namespace	None
Interface	TextEvent
Cancelable	Yes
Bubbles	Yes
Target	Element
Context info	UIEvent.view and TextEvent.data are in use.

One or more characters have been entered. The characters can originate from a variety of sources. For example, it could be characters resulting from a key being pressed or released on a keyboard device, characters resulting from the processing of an [input method editor](#), or resulting from a voice command. Where a "paste" operation generates a simple sequence of characters, i.e. a text without any structure or style information, this event type should be generated as well.

1.7.3 Keyboard event types

(This part of the specification is likely to be moved to a separate specification on keyboard events.)

This module defines the feature `KeyboardEvents 3.0` and depends on the feature `UIEvents 3.0`.

Keyboard events are device dependent, i.e. they rely on the capabilities of the input devices and how they are mapped in the operating systems. It is therefore highly recommended to rely on [Text events types](#) when dealing with character input.

Interface *KeyboardEvent* (introduced in DOM Level 3)

The `KeyboardEvent` interface provides specific contextual information associated with keyboard devices. Each keyboard event references a key using an identifier. Keyboard events are commonly directed at the element that has the focus.

The `KeyboardEvent` interface provides convenient attributes for some common modifiers keys: [KeyboardEvent.ctrlKey](#), [KeyboardEvent.shiftKey](#), [KeyboardEvent.altKey](#), [KeyboardEvent.metaKey](#). These attributes are equivalent to using the method [KeyboardEvent.getModifierState\(keyIdentifierArg\)](#) with "Control", "Shift", "Alt", or "Meta" respectively.

To create an instance of the `KeyboardEvent` interface, use the [DocumentEvent.createEvent\("KeyboardEvent"\)](#) method call.

IDL Definition

```
// Introduced in DOM Level 3:
interface KeyboardEvent : UIEvent {

    // KeyLocationCode
    const unsigned long          DOM_KEY_LOCATION_STANDARD          = 0x00;
```

```

const unsigned long    DOM_KEY_LOCATION_LEFT           = 0x01;
const unsigned long    DOM_KEY_LOCATION_RIGHT          = 0x02;
const unsigned long    DOM_KEY_LOCATION_NUMPAD         = 0x03;

readonly attribute DOMString    keyIdentifier;
readonly attribute unsigned long keyLocation;
readonly attribute boolean      ctrlKey;
readonly attribute boolean      shiftKey;
readonly attribute boolean      altKey;
readonly attribute boolean      metaKey;
boolean                  getModifierState(in DOMString keyIdentifierArg);
void                    initKeyboardEvent(in DOMString typeArg,
                                           in boolean canBubbleArg,
                                           in boolean cancelableArg,
                                           in views::AbstractView viewArg,
                                           in DOMString keyIdentifierArg,
                                           in unsigned long keyLocationArg,
                                           in DOMString modifiersListArg);

void                    initKeyboardEventNS(in DOMString namespaceURIArg,
                                           in DOMString typeArg,
                                           in boolean canBubbleArg,
                                           in boolean cancelableArg,
                                           in views::AbstractView viewArg,
                                           in DOMString keyIdentifierArg,
                                           in unsigned long keyLocationArg,
                                           in DOMString modifiersListArg);

};

```

Definition group *KeyLocationCode*

This set of constants is used to indicate the location of a key on the device. In case a DOM implementation wishes to provide a new location information, a value different from the following constant values must be used.

Defined Constants

DOM_KEY_LOCATION_LEFT

The key activated is in the left key location (there is more than one possible location for this key). Example: the left Shift key on a PC 101 Key US keyboard.

DOM_KEY_LOCATION_NUMPAD

The key activation originated on the numeric keypad or with a virtual key corresponding to the numeric keypad. Example: the '1' key on a PC 101 Key US keyboard located on the numeric pad.

DOM_KEY_LOCATION_RIGHT

The key activation is in the right key location (there is more than one possible location for this key). Example: the right Shift key on a PC 101 Key US keyboard.

DOM_KEY_LOCATION_STANDARD

The key activation is not distinguished as the left or right version of the key, and did not originate from the numeric keypad (or did not originate with a virtual key corresponding to the numeric keypad). Example: the 'Q' key on a PC 101 Key US keyboard.

Attributes

altKey of type **boolean**, **readonly**

`true` if the alternative (Alt) key modifier is activated.

Note: The Option key modifier on Macintosh systems must be represented using this key modifier.

ctrlKey of type **boolean**, **readonly**

`true` if the control (Ctrl) key modifier is activated.

keyIdentifier of type `DOMString`, **readonly**

`keyIdentifier` holds the identifier of the key. The key identifiers are defined in Appendix A.2 "[Key identifiers set](#)". Implementations that are unable to identify a key must use the key identifier "Unidentified".

keyLocation of type `unsigned long`, **readonly**

The `keyLocation` attribute contains an indication of the location of the key on the device, as described in [Keyboard event types](#).

metaKey of type `boolean`, **readonly**

`true` if the meta (Meta) key modifier is activated.

Note: The Command key modifier on Macintosh systems must be represented using this key modifier.

shiftKey of type `boolean`, **readonly**

`true` if the shift (Shift) key modifier is activated.

Methods

getModifierState

Queries the state of a modifier using a key identifier. See also [Modifier keys](#).

Parameters

keyIdentifierArg of type `DOMString`

A modifier key identifier. Common modifier keys are "Alt", "AltGraph", "CapsLock", "Control", "Meta", "NumLock", "Scroll", Or "Shift".

Note: If an application wishes to distinguish between right and left modifiers, this information could be deduced using keyboard events and [KeyboardEvent.keyLocation](#).

Return Value

`boolean true` if it is a modifier key and the modifier is activated, `false` otherwise.

No Exceptions

initKeyboardEvent

Initializes attributes of a `KeyboardEvent` object. This method has the same behavior as [UIEvent.initUIEvent\(\)](#). The value of [UIEvent.detail](#) remains undefined.

Parameters

typeArg of type `DOMString`

Refer to the [UIEvent.initUIEvent\(\)](#) method for a description of this parameter.

canBubbleArg of type `boolean`

Refer to the [UIEvent.initUIEvent\(\)](#) method for a description of this parameter.

cancelableArg of type `boolean`

Refer to the [UIEvent.initUIEvent\(\)](#) method for a description of this parameter.

viewArg of type `views::AbstractView`

Refer to the [UIEvent.initUIEvent\(\)](#) method for a description of this parameter.

keyIdentifierArg of type `DOMString`

Specifies [KeyboardEvent.keyIdentifier](#).

keyLocationArg of type `unsigned long`

Specifies [KeyboardEvent.keyLocation](#).

modifiersListArg of type `DOMString`

A *white space* separated list of modifier key identifiers to be activated on this object. As an example, "Control Alt" will mark the control and alt modifiers as activated.

No Return Value No Exceptions

initKeyboardEventNS

Initializes attributes of a `KeyboardEvent` object. This method has the same behavior as [UIEvent.initUIEventNS\(\)](#). The value of [UIEvent.detail](#) remains undefined.

Parameters

namespaceURIArg of type `DOMString`

Refer to the [UIEvent.initUIEventNS\(\)](#) method for a description of this parameter.

typeArg of type `DOMString`

Refer to the [UIEvent.initUIEventNS\(\)](#) method for a description of this parameter.

canBubbleArg of type `boolean`

Refer to the [UIEvent.initUIEventNS\(\)](#) method for a description of this parameter.

cancelableArg of type `boolean`

Refer to the [UIEvent.initUIEventNS\(\)](#) method for a description of this parameter.

viewArg of type `views::AbstractView`

Refer to the [UIEvent.initUIEventNS\(\)](#) method for a description of this parameter.

keyIdentifierArg of type `DOMString`

Refer to the [KeyboardEvent.initKeyboardEvent\(\)](#) method for a description of this parameter.

keyLocationArg of type `unsigned long`

Refer to the [KeyboardEvent.initKeyboardEvent\(\)](#) method for a description of this parameter.

modifiersListArg of type `DOMString`

Refer to the [KeyboardEvent.initKeyboardEvent\(\)](#) method for a description of this parameter.

No Return Value No Exceptions

Depending on the character generation device, keyboard events may or may not be generated.

The keyboard event types are listed below.

keydown

Type	keydown
Namespace	None
Interface	KeyboardEvent
Cancelable	Yes
Bubbles	Yes
Target	Element
Context info	UIEvent.view , KeyboardEvent.keyIdentifier , KeyboardEvent.keyLocation , KeyboardEvent.altKey , KeyboardEvent.shiftKey , KeyboardEvent.ctrlKey , and KeyboardEvent.metaKey are in use.

A key is pressed down. This event type is device dependent and relies on the capabilities of the input devices and how they are mapped in the operating system. This event type is generated after the keyboard mapping but before the processing of an [input method editor](#). This event

should logically happen before the event [keyup](#) is produced. Whether a [keydown](#) contributes or not to the generation of a text event is implementation dependent.

keyup

Type	keyup
Namespace	None
Interface	KeyboardEvent
Cancelable	Yes
Bubbles	Yes
Target	Element
Context info	UIEvent.view , KeyboardEvent.keyIdentifier , and KeyboardEvent.keyLocation are in use. KeyboardEvent.altKey , KeyboardEvent.shiftKey , KeyboardEvent.ctrlKey , and KeyboardEvent.metaKey are in use unless the KeyboardEvent.keyIdentifier corresponds to the key modifier itself.

A key is released. This event type is device dependent and relies on the capabilities of the input devices and how they are mapped in the operating system. This event type is generated after the keyboard mapping but before the processing of an [input method editor](#). This event should logically happen after the event [keydown](#) is produced. Whether a [keyup](#) contributes or not to the generation of a text event is implementation dependent.

1.7.4 Mouse event types

This module defines the feature [MouseEvent](#) 3.0 and depends on the feature [UIEvents](#) 3.0.

The [MouseEvent](#) module originates from the [[HTML 4.01](#)] [onclick](#), [ondblclick](#), [onmousedown](#), [onmouseup](#), [onmouseover](#), [onmousemove](#), and [onmouseout](#) attributes. This event module is specifically designed for use with pointing input devices, such as a mouse or a trackball.

Interface *MouseEvent* (introduced in DOM Level 2)

The `MouseEvent` interface provides specific contextual information associated with Mouse events.

In the case of nested elements mouse events are always targeted at the most deeply nested element. Ancestors of the targeted element may use bubbling to obtain notification of mouse events which occur within their descendent elements.

To create an instance of the `MouseEvent` interface, use the [DocumentEvent.createEvent\("MouseEvent"\)](#) method call.

Note: When initializing `MouseEvent` objects using `initMouseEvent` or `initMouseEventNS`, implementations should use the client coordinates `clientX` and `clientY` for calculation of other coordinates (such as target coordinates exposed by [DOM Level 0](#) implementations).

IDL Definition

```
// Introduced in DOM Level 2:
interface MouseEvent : UIEvent {
  readonly attribute long           screenX;
  readonly attribute long           screenY;
  readonly attribute long           clientX;
  readonly attribute long           clientY;
  readonly attribute boolean        ctrlKey;
```

```

readonly attribute boolean      shiftKey;
readonly attribute boolean      altKey;
readonly attribute boolean      metaKey;
readonly attribute unsigned short button;
readonly attribute EventTarget  relatedTarget;
void                          initMouseEvent(in DOMString typeArg,
                                                in boolean canBubbleArg,
                                                in boolean cancelableArg,
                                                in views::AbstractView viewArg,
                                                in long detailArg,
                                                in long screenXArg,
                                                in long screenYArg,
                                                in long clientXArg,
                                                in long clientYArg,
                                                in boolean ctrlKeyArg,
                                                in boolean altKeyArg,
                                                in boolean shiftKeyArg,
                                                in boolean metaKeyArg,
                                                in unsigned short buttonArg,
                                                in EventTarget relatedTargetArg);

// Introduced in DOM Level 3:
boolean      getModifierState(in DOMString keyIdentifierArg);
// Introduced in DOM Level 3:
void        initMouseEventNS(in DOMString namespaceURIArg,
                                in DOMString typeArg,
                                in boolean canBubbleArg,
                                in boolean cancelableArg,
                                in views::AbstractView viewArg,
                                in long detailArg,
                                in long screenXArg,
                                in long screenYArg,
                                in long clientXArg,
                                in long clientYArg,
                                in unsigned short buttonArg,
                                in EventTarget relatedTargetArg,
                                in DOMString modifiersListArg);

};

```

Attributes

altKey of type **boolean**, **readonly**

Refer to the [KeyboardEvent.altKey](#) attribute.

button of type **unsigned short**, **readonly**

During mouse events caused by the depression or release of a mouse button, `button` is used to indicate which mouse button changed state. 0 indicates the normal button of the mouse (in general on the left or the one button on Macintosh mice, used to activate a button or select text). 2 indicates the contextual property (in general on the right, used to display a context menu) button of the mouse if present. 1 indicates the extra (in general in the middle and often combined with the mouse wheel) button. Some mice may provide or simulate more buttons, and values higher than 2 can be used to represent such buttons.

clientX of type **long**, **readonly**

The horizontal coordinate at which the event occurred relative to the viewport associated with the event.

clientY of type **long**, **readonly**

The vertical coordinate at which the event occurred relative to the viewport associated with the event.

ctrlKey of type **boolean**, **readonly**

Refer to the [KeyboardEvent.ctrlKey](#) attribute.

metaKey of type **boolean**, **readonly**

Refer to the [KeyboardEvent.metaKey](#) attribute.

relatedTarget of type **EventTarget**, **readonly**

Used to identify a secondary [EventTarget](#) related to a UI event, depending on the type of event.

screenX of type `long`, **readonly**

The horizontal coordinate at which the event occurred relative to the origin of the screen coordinate system.

screenY of type `long`, **readonly**

The vertical coordinate at which the event occurred relative to the origin of the screen coordinate system.

shiftKey of type `boolean`, **readonly**

Refer to the [KeyboardEvent.shiftKey](#) attribute.

Methods

getModifierState introduced in DOM Level 3

Queries the state of a modifier using a key identifier. See also [Modifier keys](#).

Parameters

keyIdentifierArg of type `DOMString`

Refer to the [KeyboardEvent.getModifierState\(\)](#) method for a description of this parameter.

Return Value

`boolean` `true` if it is a modifier key and the modifier is activated, `false` otherwise.

No Exceptions

initMouseEvent

Initializes attributes of a `MouseEvent` object. This method has the same behavior as [UIEvent.initUIEvent\(\)](#).

Parameters

typeArg of type `DOMString`

Refer to the [UIEvent.initUIEvent\(\)](#) method for a description of this parameter.

canBubbleArg of type `boolean`

Refer to the [UIEvent.initUIEvent\(\)](#) method for a description of this parameter.

cancelableArg of type `boolean`

Refer to the [UIEvent.initUIEvent\(\)](#) method for a description of this parameter.

viewArg of type `views::AbstractView`

Refer to the [UIEvent.initUIEvent\(\)](#) method for a description of this parameter.

detailArg of type `long`

Refer to the [UIEvent.initUIEvent\(\)](#) method for a description of this parameter.

screenXArg of type `long`

Specifies [MouseEvent.screenX](#).

screenYArg of type `long`

Specifies [MouseEvent.screenY](#).

clientXArg of type `long`

Specifies [MouseEvent.clientX](#).

clientYArg of type `long`

Specifies [MouseEvent.clientY](#).

ctrlKeyArg of type `boolean`

Specifies [MouseEvent.ctrlKey](#).

altKeyArg of type `boolean`

Specifies [MouseEvent.altKey](#).

shiftKeyArg of type boolean

Specifies [MouseEvent.shiftKey](#).

metaKeyArg of type boolean

Specifies [MouseEvent.metaKey](#).

buttonArg of type unsigned short

Specifies [MouseEvent.button](#).

relatedTargetArg of type [EventTarget](#)

Specifies [MouseEvent.relatedTarget](#). This value may be `null`.

No Return Value

No Exceptions

[initMouseEventNS](#) introduced in DOM Level 3

Initializes attributes of a [MouseEvent](#) object. This method has the same behavior as [UIEvent.initUIEventNS\(\)](#).

Parameters

namespaceURIArg of type [DOMString](#)

Refer to the [UIEvent.initUIEventNS\(\)](#) method for a description of this parameter.

typeArg of type [DOMString](#)

Refer to the [UIEvent.initUIEventNS\(\)](#) method for a description of this parameter.

canBubbleArg of type boolean

Refer to the [UIEvent.initUIEventNS\(\)](#) method for a description of this parameter.

cancelableArg of type boolean

Refer to the [UIEvent.initUIEventNS\(\)](#) method for a description of this parameter.

viewArg of type [views::AbstractView](#)

Refer to the [UIEvent.initUIEventNS\(\)](#) method for a description of this parameter.

detailArg of type long

Refer to the [UIEvent.initUIEventNS\(\)](#) method for a description of this parameter.

screenXArg of type long

Refer to the [MouseEvent.initMouseEvent\(\)](#) method for a description of this parameter.

screenYArg of type long

Refer to the [MouseEvent.initMouseEvent\(\)](#) method for a description of this parameter.

clientXArg of type long

Refer to the [MouseEvent.initMouseEvent\(\)](#) method for a description of this parameter.

clientYArg of type long

Refer to the [MouseEvent.initMouseEvent\(\)](#) method for a description of this parameter.

buttonArg of type unsigned short

Refer to the [MouseEvent.initMouseEvent\(\)](#) method for a description of this parameter.

relatedTargetArg of type [EventTarget](#)

Refer to the [MouseEvent.initMouseEvent\(\)](#) method for a description of this parameter.

modifiersListArg of type [DOMString](#)

Refer to the [KeyboardEvent.initKeyboardEventNS\(\)](#) method for a description of this parameter.

No Return Value

No Exceptions

The Mouse event types are listed below. In the case of nested elements, mouse event types are always targeted at the most deeply nested element. Ancestors of the targeted element may use bubbling to obtain notification of mouse events which occur within its descendent elements.

Implementations must maintain the *current click count* when generating mouse events. This is a non-negative integer indicating the number of consecutive clicks of a pointing device button during a user action. The notion of consecutive clicks depends on the environment configuration. For example, a "double click" might not happen if there is a long delay between the two clicks.

click

Type	click
Namespace	None
Interface	MouseEvent
Cancelable	Yes
Bubbles	Yes
Target	Element
Context info	MouseEvent.screenX , MouseEvent.screenY , MouseEvent.clientX , MouseEvent.clientY , MouseEvent.altKey , MouseEvent.ctrlKey , MouseEvent.shiftKey , MouseEvent.metaKey , MouseEvent.button , and UIEvent.view are in use. The UIEvent.detail attribute indicates the current click count . The attribute value is 1 when the user begins this action and increments by 1 for each click.

A pointing device button is clicked over an element. The definition of a click depends on the environment configuration; i.e. it may depend on the screen location or the delay between the press and release of the pointing device button. In any case, the event target must be the same between the [mousedown](#), [mouseup](#), and [click](#). The sequence of these events is: [mousedown](#), [mouseup](#), and [click](#). It depends on the environment configuration whether the event type [click](#) can occur if one or more of the event types [mouseover](#), [mousemove](#), and [mouseout](#) occur between the press and release of the pointing device button. In addition, the event type is dispatched as described in [Activation requests and behavior](#).

dblclick

Type	dblclick
Namespace	None
Interface	MouseEvent
Cancelable	Yes
Bubbles	Yes
Target	Element
Context info	MouseEvent.screenX , MouseEvent.screenY , MouseEvent.clientX , MouseEvent.clientY , MouseEvent.altKey , MouseEvent.ctrlKey , MouseEvent.shiftKey , MouseEvent.metaKey , MouseEvent.button , and UIEvent.view are in use. The UIEvent.detail attribute indicates the current click count .

A pointing device button is clicked twice over an element. The definition of a double click depends on the environment configuration, except that the event target must be the same between [mousedown](#), [mouseup](#), and [dblclick](#). This event type is dispatched after the event type [click](#) if a click and double click occur simultaneously, and after the event type [mouseup](#) otherwise.

mousedown

Type	mousedown
------	-----------

Namespace	None
Interface	MouseEvent
Cancelable	Yes
Bubbles	Yes
Target	Element
Context info	MouseEvent.screenX , MouseEvent.screenY , MouseEvent.clientX , MouseEvent.clientY , MouseEvent.altKey , MouseEvent.ctrlKey , MouseEvent.shiftKey , MouseEvent.metaKey , MouseEvent.button , and UIEvent.view are in use. The UIEvent.detail attribute indicates the current click count incremented by one. For example, if no click happened before the mousedown, UIEvent.detail will contain the value 1.

A pointing device button is pressed over an element.

mouseup

Type	mouseup
Namespace	None
Interface	MouseEvent
Cancelable	Yes
Bubbles	Yes
Target	Element
Context info	MouseEvent.screenX , MouseEvent.screenY , MouseEvent.clientX , MouseEvent.clientY , MouseEvent.altKey , MouseEvent.ctrlKey , MouseEvent.shiftKey , MouseEvent.metaKey , MouseEvent.button , and UIEvent.view are in use. The UIEvent.detail attribute indicates the current click count incremented by one.

A pointing device button is released over an element.

mouseover

Type	mouseover
Namespace	None
Interface	MouseEvent
Cancelable	Yes
Bubbles	Yes
Target	Element
Context info	MouseEvent.screenX , MouseEvent.screenY , MouseEvent.clientX , MouseEvent.clientY , MouseEvent.altKey , MouseEvent.ctrlKey , MouseEvent.shiftKey , MouseEvent.metaKey , and UIEvent.view are in use. MouseEvent.relatedTarget indicates the event target a pointing device is exiting, if any.

A pointing device is moved onto an element.

mousemove

Type	mousemove
Namespace	None
Interface	MouseEvent
Cancelable	Yes
Bubbles	Yes

Target	Element
Context info	MouseEvent.screenX , MouseEvent.screenY , MouseEvent.clientX , MouseEvent.clientY , MouseEvent.altKey , MouseEvent.ctrlKey , MouseEvent.shiftKey , MouseEvent.metaKey , and UIEvent.view are in use.

A pointing device is moved while it is over an element.

mouseout

Type	mouseout
Namespace	None
Interface	MouseEvent
Cancelable	Yes
Bubbles	Yes
Target	Element
Context info	MouseEvent.screenX , MouseEvent.screenY , MouseEvent.clientX , MouseEvent.clientY , MouseEvent.altKey , MouseEvent.ctrlKey , MouseEvent.shiftKey , MouseEvent.metaKey , and UIEvent.view are in use. MouseEvent.relatedTarget indicates the event target a pointing device is entering, if any.

A pointing device is moved away from an element.

1.7.5 Mouse multi wheel event types

This module defines the feature `MouseMultiWheelEvents 3.0` and depends on the feature `MouseEvents 3.0`.

Multi wheels are devices that can be rotated in one or more spatial dimensions. The coordinate system depends on the environment configuration. As an example, the environment may be configured to associate vertical scrolling with rotation along the y-axis, horizontal scrolling with rotation along the x-axis, and zooming with rotation along the z-axis. The delta attributes of [MouseMultiWheelEvent](#) objects indicate the distance of the rotation. The measurement unit depends on the environment configuration. The sign of the delta value should indicate the direction of the rotation.

Interface *MouseMultiWheelEvent* (introduced in DOM Level 3)

The `MouseMultiWheelEvent` interface provides specific contextual information associated with mouse multi wheel events.

To create an instance of the `MouseMultiWheelEvent` interface, use the [DocumentEvent.createEvent\("MouseMultiWheelEvent"\)](#) method call.

IDL Definition

```
// Introduced in DOM Level 3:
interface MouseMultiWheelEvent : MouseEvent {
  readonly attribute long           wheelDeltaX;
  readonly attribute long           wheelDeltaY;
  readonly attribute long           wheelDeltaZ;
  void                             initMouseMultiWheelEventNS(in DOMString namespaceURIArg,
                                                                in DOMString typeArg,
                                                                in boolean canBubbleArg,
                                                                in boolean cancelableArg,
                                                                in views::AbstractView
```

```

viewArg,
                                in long detailArg,
                                in long screenXArg,
                                in long screenYArg,
                                in long clientXArg,
                                in long clientYArg,
                                in unsigned short buttonArg,
                                in EventTarget
relatedTargetArg,
                                in DOMString
modifiersListArg,
                                in long wheelDeltaXArg,
                                in long wheelDeltaYArg,
                                in long wheelDeltaZArg);
};

```

Attributes

wheelDeltaX of type long, readonly

The distance the wheel has rotated around the x-axis.

wheelDeltaY of type long, readonly

The distance the wheel has rotated around the y-axis.

wheelDeltaZ of type long, readonly

The distance the wheel has rotated around the z-axis.

Methods

[initMouseEventNS](#)

Initializes attributes of a `MouseEvent` object. This method has the same behavior as [MouseEvent.initMouseEventNS\(\)](#).

Parameters

namespaceURIArg of type DOMString

Refer to the [UIEvent.initUIEventNS\(\)](#) method for a description of this parameter.

typeArg of type DOMString

Refer to the [UIEvent.initUIEventNS\(\)](#) method for a description of this parameter.

canBubbleArg of type boolean

Refer to the [UIEvent.initUIEventNS\(\)](#) method for a description of this parameter.

cancelableArg of type boolean

Refer to the [UIEvent.initUIEventNS\(\)](#) method for a description of this parameter.

viewArg of type `views::AbstractView`

Refer to the [UIEvent.initUIEventNS\(\)](#) method for a description of this parameter.

detailArg of type long

Refer to the [UIEvent.initUIEventNS\(\)](#) method for a description of this parameter.

screenXArg of type long

Refer to the [MouseEvent.initMouseEventNS\(\)](#) method for a description of this parameter.

screenYArg of type long

Refer to the [MouseEvent.initMouseEventNS\(\)](#) method for a description of this parameter.

clientXArg of type long

Refer to the [MouseEvent.initMouseEventNS\(\)](#) method for a description of this parameter.

clientYArg of type long

Refer to the [MouseEvent.initMouseEventNS\(\)](#) method for a description of this parameter.

buttonArg of type **unsigned short**

Refer to the [MouseEvent.initMouseEventNS\(\)](#) method for a description of this parameter.

relatedTargetArg of type **EventTarget**

Refer to the [MouseEvent.initMouseEventNS\(\)](#) method for a description of this parameter.

modifiersListArg of type **DOMString**

Refer to the [MouseEvent.initMouseEventNS\(\)](#) method for a description of this parameter.

wheelDeltaXArg of type **long**

Specifies [MouseMultiWheelEvent.wheelDeltaX](#).

wheelDeltaYArg of type **long**

Specifies [MouseMultiWheelEvent.wheelDeltaY](#).

wheelDeltaZArg of type **long**

Specifies [MouseMultiWheelEvent.wheelDeltaZ](#).

No Return Value
No Exceptions

mousemultiwheel

Type	mousemultiwheel
Namespace	None
Interface	MouseMultiWheelEvent
Cancelable	Yes
Bubbles	Yes
Target	Document, Element
Context info	MouseMultiWheelEvent.wheelDeltaX , MouseMultiWheelEvent.wheelDeltaY , MouseMultiWheelEvent.wheelDeltaZ , MouseEvent.altKey , MouseEvent.ctrlKey , MouseEvent.shiftKey , MouseEvent.metaKey , and UIEvent.view are in use. MouseEvent.screenX , MouseEvent.screenY , MouseEvent.clientX , MouseEvent.clientY , and MouseEvent.button are in use if the wheel is associated to a pointing device. MouseEvent.relatedTarget indicates the event target the pointing device is pointing at, if any. UIEvent.detail is not in use.

A mouse wheel has been rotated. A default action of user agent generated event objects of this type causes implementations to dispatch a `mousewheel` event iff it supports that event type and [MouseMultiWheelEvent.wheelDeltaY](#) is non-zero.

1.7.6 Mouse wheel event types

This module defines the feature `MouseWheelEvents 3.0` and depends on the feature `MouseEvents 3.0`.

Mouse wheel events are a subset of mouse multi wheel events. See [Mouse multi wheel event types](#) for additional information.

Interface *MouseWheelEvent* (introduced in DOM Level 3)

The `MouseWheelEvent` interface provides specific contextual information associated with mouse wheel events.

To create an instance of the `MouseWheelEvent` interface, use the

[DocumentEvent.createEvent\("MouseEvent"\)](#) method call.

IDL Definition

```
// Introduced in DOM Level 3:
interface MouseEvent : MouseEvent {
  readonly attribute long          wheelDelta;
  void          initMouseEventNS(in DOMString namespaceURIArg,
                                   in DOMString typeArg,
                                   in boolean canBubbleArg,
                                   in boolean cancelableArg,
                                   in views::AbstractView viewArg,
                                   in long detailArg,
                                   in long screenXArg,
                                   in long screenYArg,
                                   in long clientXArg,
                                   in long clientYArg,
                                   in unsigned short buttonArg,
                                   in EventTarget relatedTargetArg,
                                   in DOMString modifiersListArg,
                                   in long wheelDeltaArg);
};
```

Attributes

wheelDelta of type **long**, **readonly**

The distance the wheel has rotated around the y-axis.

Methods

initMouseEventNS

Initializes attributes of a **MouseEvent** object. This method has the same behavior as [MouseEvent.initMouseEventNS\(\)](#).

Parameters

namespaceURIArg of type **DOMString**

Refer to the [UIEvent.initUIEventNS\(\)](#) method for a description of this parameter.

typeArg of type **DOMString**

Refer to the [UIEvent.initUIEventNS\(\)](#) method for a description of this parameter.

canBubbleArg of type **boolean**

Refer to the [UIEvent.initUIEventNS\(\)](#) method for a description of this parameter.

cancelableArg of type **boolean**

Refer to the [UIEvent.initUIEventNS\(\)](#) method for a description of this parameter.

viewArg of type **views::AbstractView**

Refer to the [UIEvent.initUIEventNS\(\)](#) method for a description of this parameter.

detailArg of type **long**

Refer to the [UIEvent.initUIEventNS\(\)](#) method for a description of this parameter.

screenXArg of type **long**

Refer to the [MouseEvent.initMouseEventNS\(\)](#) method for a description of this parameter.

screenYArg of type **long**

Refer to the [MouseEvent.initMouseEventNS\(\)](#) method for a description of this parameter.

parameter.

clientXArg of type long

Refer to the [MouseEvent.initMouseEventNS\(\)](#) method for a description of this parameter.

clientYArg of type long

Refer to the [MouseEvent.initMouseEventNS\(\)](#) method for a description of this parameter.

buttonArg of type unsigned short

Refer to the [MouseEvent.initMouseEventNS\(\)](#) method for a description of this parameter.

relatedTargetArg of type EventTarget

Refer to the [MouseEvent.initMouseEventNS\(\)](#) method for a description of this parameter.

modifiersListArg of type DOMString

Refer to the [MouseEvent.initMouseEventNS\(\)](#) method for a description of this parameter.

wheelDeltaArg of type long

Specifies [MouseEvent.wheelDelta](#).

No Return Value

No Exceptions

mousewheel

Type	mousewheel
Namespace	None
Interface	MouseEvent
Cancelable	Yes
Bubbles	Yes
Target	Document, Element
Context info	MouseEvent.wheelDelta , MouseEvent.altKey , MouseEvent.ctrlKey , MouseEvent.shiftKey , MouseEvent.metaKey , and UIEvent.view are in use. MouseEvent.screenX , MouseEvent.screenY , MouseEvent.clientX , MouseEvent.clientY , and MouseEvent.button are in use if the wheel is associated to a pointing device. MouseEvent.relatedTarget indicates the event target the pointing device is pointing at, if any. UIEvent.detail is not in use.

A mouse wheel has been rotated around the y-axis.

1.7.7 Mutation event types

This module defines the feature MutationEvents 3.0 and depends on the feature Events 3.0.

The mutation and mutation name event modules are designed to allow notification of any changes to the structure of a document, including attribute, text, or name modifications. It may be noted that none of the event types associated with the modules are designated as cancelable. This stems from the fact that it is very difficult to make use of existing DOM interfaces which cause document modifications if any change to the document might or might not take place due to cancelation of the resulting event. Although this is still a desired capability, it was decided that it would be better left until the addition of transactions into the DOM.

Many single modifications of the tree can cause multiple mutation events to be dispatched. Rather than attempt to specify the ordering of mutation events due to every possible modification of the tree, the ordering of these events is left to the implementation.

Interface *MutationEvent* (introduced in DOM Level 2)

The `MutationEvent` interface provides specific contextual information associated with Mutation events.

To create an instance of the `MutationEvent` interface, use the [DocumentEvent.createEvent\("MutationEvent"\)](#) method call.

IDL Definition

```
// Introduced in DOM Level 2:
interface MutationEvent : Event {

    // attrChangeType
    const unsigned short      MODIFICATION          = 1;
    const unsigned short      ADDITION              = 2;
    const unsigned short      REMOVAL                = 3;

    readonly attribute Node    relatedNode;
    readonly attribute DOMString prevValue;
    readonly attribute DOMString newValue;
    readonly attribute DOMString attrName;
    readonly attribute unsigned short attrChange;
    void initMutationEvent(in DOMString typeArg,
                           in boolean canBubbleArg,
                           in boolean cancelableArg,
                           in Node relatedNodeArg,
                           in DOMString prevValueArg,
                           in DOMString newValueArg,
                           in DOMString attrNameArg,
                           in unsigned short attrChangeArg);

    // Introduced in DOM Level 3:
    void initMutationEventNS(in DOMString namespaceURIArg,
                              in DOMString typeArg,
                              in boolean canBubbleArg,
                              in boolean cancelableArg,
                              in Node relatedNodeArg,
                              in DOMString prevValueArg,
                              in DOMString newValueArg,
                              in DOMString attrNameArg,
                              in unsigned short attrChangeArg);

};
```

Definition group *attrChangeType*

An integer indicating in which way the `Attr` was changed.

Defined Constants

ADDITION

The `Attr` was just added.

MODIFICATION

The `Attr` was modified in place.

REMOVAL

The `Attr` was just removed.

Attributes

attrChange of type `unsigned short`, `readonly`

`attrChange` indicates the type of change which triggered the [DOMAttrModified](#) event.

The values can be `MODIFICATION`, `ADDITION`, or `REMOVAL`.

attrName of type `DOMString`, **readonly**

`attrName` indicates the name of the changed `Attr` node in a [DOMAttrModified](#) event.

newValue of type `DOMString`, **readonly**

`newValue` indicates the new value of the `Attr` node in [DOMAttrModified](#) events, and of the `CharacterData` node in [DOMCharacterDataModified](#) events.

prevValue of type `DOMString`, **readonly**

`prevValue` indicates the previous value of the `Attr` node in [DOMAttrModified](#) events, and of the `CharacterData` node in [DOMCharacterDataModified](#) events.

relatedNode of type `Node`, **readonly**

`relatedNode` is used to identify a secondary node related to a mutation event. For example, if a mutation event is dispatched to a node indicating that its parent has changed, the `relatedNode` is the changed parent. If an event is instead dispatched to a subtree indicating a node was changed within it, the `relatedNode` is the changed node. In the case of the [DOMAttrModified](#) event it indicates the `Attr` node which was modified, added, or removed.

Methods

initMutationEvent

Initializes attributes of a `MutationEvent` object. This method has the same behavior as [Event.initEvent\(\)](#).

Parameters

typeArg of type `DOMString`

Refer to the [Event.initEvent\(\)](#) method for a description of this parameter.

canBubbleArg of type `boolean`

Refer to the [Event.initEvent\(\)](#) method for a description of this parameter.

cancelableArg of type `boolean`

Refer to the [Event.initEvent\(\)](#) method for a description of this parameter.

relatedNodeArg of type `Node`

Specifies [MutationEvent.relatedNode](#).

prevValueArg of type `DOMString`

Specifies [MutationEvent.prevValue](#). This value may be null.

newValueArg of type `DOMString`

Specifies [MutationEvent.newValue](#). This value may be null.

attrNameArg of type `DOMString`

Specifies [MutationEvent.attrName](#). This value may be null.

attrChangeArg of type `unsigned short`

Specifies [MutationEvent.attrChange](#). This value may be null.

No Return Value

No Exceptions

initMutationEventNS introduced in DOM Level 3

Initializes attributes of a `MutationEvent` object. This method has the same behavior as [Event.initEventNS\(\)](#).

Parameters

namespaceURIArg of type `DOMString`

Refer to the [Event.initEventNS\(\)](#) method for a description of this parameter.

typeArg of type `DOMString`

Refer to the [Event.initEventNS\(\)](#) method for a description of this parameter.

canBubbleArg of type `boolean`

Refer to the [Event.initEventNS\(\)](#) method for a description of this parameter.

cancelableArg of type `boolean`

Refer to the [Event.initEventNS\(\)](#) method for a description of this parameter.

relatedNodeArg of type `Node`

Refer to the [MutationEvent.initMutationEvent\(\)](#) method for a description of this parameter.

prevValueArg of type DOMString

Refer to the [MutationEvent.initMutationEvent\(\)](#) method for a description of this parameter.

newValueArg of type DOMString

Refer to the [MutationEvent.initMutationEvent\(\)](#) method for a description of this parameter.

attrNameArg of type DOMString

Refer to the [MutationEvent.initMutationEvent\(\)](#) method for a description of this parameter.

attrChangeArg of type unsigned short

Refer to the [MutationEvent.initMutationEvent\(\)](#) method for a description of this parameter.

No Return Value**No Exceptions**

The mutation event types are listed below.

DOMSubtreeModified

Type	DOMSubtreeModified
Namespace	None
Interface	MutationEvent
Cancelable	No
Bubbles	Yes
Target	Document, DocumentFragment, Element, Attr
Context info	None

This is a general event for notification of all changes to the document. It can be used instead of the more specific mutation and mutation name events listed below. It may be dispatched after a single modification to the document or, at the implementation's discretion, after multiple changes have occurred. The latter use should generally be used to accommodate multiple changes which occur either simultaneously or in rapid succession. The target of this event is the lowest common parent of the changes which have taken place. This event is dispatched after any other events caused by the mutation(s) have occurred.

DOMNodeInserted

Type	DOMNodeInserted
Namespace	None
Interface	MutationEvent
Cancelable	No
Bubbles	Yes
Target	Element, Attr, Text, Comment, CDATASection, DocumentType, EntityReference, ProcessingInstruction
Context info	MutationEvent.relatedNode holds the parent node of the node that has been inserted or, in case of <code>Attr</code> nodes, the <code>ownerElement</code> of the <code>Attr</code> node.

A node has been added as a child of another node or, in case of `Attr` nodes, has been added to an `Element`. This event is dispatched after the insertion has taken place. The [target node](#) of this event is the node being inserted.

DOMNodeRemoved

Type	DOMNodeRemoved
------	----------------

Namespace	None
Interface	MutationEvent
Cancelable	No
Bubbles	Yes
Target	Element, Attr, Text, Comment, CDATASection, DocumentType, EntityReference, ProcessingInstruction
Context info	MutationEvent.relatedNode holds the parent node of the node being removed or, in case of <code>Attr</code> nodes, the <code>ownerElement</code> of the <code>Attr</code> node.

A node is being removed from its parent node or, in case of `Attr` nodes, removed from its `ownerElement`. This event is dispatched before the removal takes place. The [target node](#) of this event is the node being removed.

DOMNodeRemovedFromDocument

Type	DOMNodeRemovedFromDocument
Namespace	None
Interface	MutationEvent
Cancelable	No
Bubbles	Yes
Target	Element, Attr, Text, Comment, CDATASection, DocumentType, EntityReference, ProcessingInstruction
Context info	None

A node is being removed from a document, either through direct removal of the node or removal of a subtree in which it is contained; `Attr` nodes are considered part of an `Element`'s subtree. This event is dispatched before the removal takes place. The [target node](#) of this event type is the node being removed. If the node is being directly removed, the event type [DOMNodeRemoved](#) occurs before this event type.

DOMNodeInsertedIntoDocument

Type	DOMNodeInsertedIntoDocument
Namespace	None
Interface	MutationEvent
Cancelable	No
Bubbles	Yes
Target	Element, Attr, Text, Comment, CDATASection, DocumentType, EntityReference, ProcessingInstruction
Context info	None

A node has been inserted into a document, either through direct insertion of the node or insertion of a subtree in which it is contained; `Attr` nodes are considered part of an `Element`'s subtree. This event is dispatched after the insertion has taken place. The [target node](#) of this event is the node being inserted. If the node is being directly inserted, the event type [DOMNodeInserted](#) occurs before this event type.

DOMAttrModified

Type	DOMAttrModified
Namespace	None
Interface	MutationEvent
Cancelable	No
Bubbles	Yes

Target	Element
Context info	<code>MutationEvent.attrName</code> and <code>MutationEvent.attrChange</code> are in use. The value of <code>MutationEvent.relatedNode</code> indicates the <code>Attr</code> node that has been modified, added, or removed. If the <code>Attr</code> node has been added, <code>MutationEvent.newValue</code> is in use. If the <code>Attr</code> node has been removed, <code>MutationEvent.prevValue</code> is in use. If the <code>Attr</code> node has been modified, <code>MutationEvent.newValue</code> and <code>MutationEvent.prevValue</code> are in use.

Occurs after `Attr.value` has been modified and after an `Attr` node has been added to or removed from an `Element`. The **target node** of this event is the `Element` node where the change occurred. It is implementation dependent whether this event type occurs when the children of the `Attr` node are changed in ways that do not affect the value of `Attr.value`.

DOMCharacterDataModified

Type	DOMCharacterDataModified
Namespace	None
Interface	<code>MutationEvent</code>
Cancelable	No
Bubbles	Yes
Target	Text, Comment, CDATASection, ProcessingInstruction
Context info	<code>MutationEvent.prevValue</code> , and <code>MutationEvent.newValue</code> are in use.

Occurs after `CharacterData.data` OR `ProcessingInstruction.data` have been modified but the node itself has not been inserted or deleted. The **target node** of this event is the `CharacterData` node or the `ProcessingInstruction` node.

1.7.8 Mutation name event types

This module defines the feature `MutationNameEvents 3.0` and depends on the features `MutationEvents 3.0` and `Core 3.0`.

Interface *MutationNameEvent* (introduced in DOM Level 3)

The `MutationNameEvent` interface provides specific contextual information associated with Mutation name event types.

To create an instance of the `MutationNameEvent` interface, use the `Document.createEvent("MutationNameEvent")` method call.

IDL Definition

```
// Introduced in DOM Level 3:
interface MutationNameEvent : MutationEvent {
    readonly attribute DOMString      prevNamespaceURI;
    readonly attribute DOMString      prevNodeName;
    // Introduced in DOM Level 3:
    void      initMutationNameEvent(in DOMString typeArg,
                                     in boolean canBubbleArg,
                                     in boolean cancelableArg,
                                     in Node relatedNodeArg,
                                     in DOMString prevNamespaceURIArg,
                                     in DOMString prevNodeNameArg);

    // Introduced in DOM Level 3:
    void      initMutationNameEventNS(in DOMString namespaceURIArg,
```

```

prevNamespaceURIArg,
};
in DOMString typeArg,
in boolean canBubbleArg,
in boolean cancelableArg,
in Node relatedNodeArg,
in DOMString
in DOMString prevNodeNameArg);

```

Attributes

prevNamespaceURI of type **DOMString**, **readonly**

The previous value of the `relatedNode`'S `namespaceURI`.

prevNodeName of type **DOMString**, **readonly**

The previous value of the `relatedNode`'S `nodeName`.

Methods

initMutationNameEvent introduced in **DOM Level 3**

Initializes attributes of a `MutationNameEvent` object. This method has the same behavior as [MutationEvent.initMutationEvent\(\)](#).

Parameters

typeArg of type **DOMString**

Refer to the [MutationEvent.initMutationEvent\(\)](#) method for a description of this parameter.

canBubbleArg of type **boolean**

Refer to the [MutationEvent.initMutationEvent\(\)](#) method for a description of this parameter.

cancelableArg of type **boolean**

Refer to the [MutationEvent.initMutationEvent\(\)](#) method for a description of this parameter.

relatedNodeArg of type **Node**

Refer to the [MutationEvent.initMutationEvent\(\)](#) method for a description of this parameter.

prevNamespaceURIArg of type **DOMString**

Specifies [MutationNameEvent.prevNamespaceURI](#). This value may be `null`.

prevNodeNameArg of type **DOMString**

Specifies [MutationNameEvent.prevNodeName](#).

No Return Value

No Exceptions

initMutationNameEventNS introduced in **DOM Level 3**

Initializes attributes of a `MutationNameEvent` object. This method has the same behavior as [MutationEvent.initMutationEventNS\(\)](#).

Parameters

namespaceURIArg of type **DOMString**

Refer to the [MutationEvent.initMutationEventNS\(\)](#) method for a description of this parameter.

typeArg of type **DOMString**

Refer to the [MutationEvent.initMutationEventNS\(\)](#) method for a description of this parameter.

canBubbleArg of type **boolean**

Refer to the [MutationEvent.initMutationEventNS\(\)](#) method for a description of this parameter.

cancelableArg of type **boolean**

Refer to the [MutationEvent.initMutationEventNS\(\)](#) method for a description of this parameter.

relatedNodeArg of type Node

Refer to the [MutationEvent.initMutationEventNS\(\)](#) method for a description of this parameter.

prevNamespaceURIArg of type DOMString

Refer to the [MutationEvent.initMutationEvent\(\)](#) method for a description of this parameter.

prevNodeNameArg of type DOMString

Refer to the [MutationEvent.initMutationEvent\(\)](#) method for a description of this parameter.

No Return Value**No Exceptions**

The mutation name event types are listed below.

DOMElementNameChanged

Type	DOMElementNameChanged
Namespace	None
Interface	MutationNameEvent
Cancelable	No
Bubbles	Yes
Target	Element
Context info	MutationNameEvent.prevNamespaceURI , and MutationNameEvent.prevNodeName are in use.

Occurs after the `namespaceURI` and/or the `nodeName` of an `Element` node have been modified (e.g., the element was renamed using `Document.renameNode()`). The **target node** of this event is the renamed `Element` node.

DOMAttributeNameChanged

Type	DOMAttributeNameChanged
Namespace	None
Interface	MutationNameEvent
Cancelable	No
Bubbles	Yes
Target	Element
Context info	MutationNameEvent.prevNamespaceURI , and MutationNameEvent.prevNodeName are in use. The value of MutationEvent.relatedNode contains the renamed <code>Attr</code> node.

Occurs after the `namespaceURI` and/or the `nodeName` of a `Attr` node have been modified (e.g., the attribute was renamed using `Document.renameNode()`). The **target node** of this event is the `Element` node whose `Attr` has been renamed.

1.7.9 Basic event types

This event module contains basic event types associated with document manipulation. It defines the feature `BasicEvents 3.0` and depends on the feature `Events 3.0`. The basic event types are listed below.

load

Type	load
------	------

Namespace	None
Interface	UIEvent if generated from a user interface, Event otherwise.
Cancelable	No
Bubbles	No
Target	Document, Element
Context info	UIEvent.view may be in use.

The DOM Implementation finishes loading the resource (such as the document) and any dependent resources (such as images, style sheets, or scripts). Dependent resources that fail to load will not prevent this event from firing if the resource that loaded them is still accessible via the DOM. If this event type is dispatched, implementations are required to dispatch this event at least on the `Document` node.

unload

Type	unload
Namespace	None
Interface	UIEvent if generated from a user interface, Event otherwise.
Cancelable	No
Bubbles	No
Target	Document, Element
Context info	UIEvent.view may be in use.

The DOM implementation removes from the environment the resource (such as the document) or any dependent resources (such as images, style sheets, scripts). The document is unloaded after the dispatch of this event type. If this event type is dispatched, implementations are required to dispatch this event at least on the `Document` node.

abort

Type	abort
Namespace	None
Interface	UIEvent if generated from a user interface, Event otherwise.
Cancelable	No
Bubbles	Yes
Target	Element
Context info	UIEvent.view may be in use.

Loading of a resource has been aborted.

error

Type	error
Namespace	None
Interface	UIEvent if generated from a user interface, Event otherwise.
Cancelable	No
Bubbles	Yes
Target	Element
Context info	UIEvent.view may be in use.

A resource failed to load, or has been loaded but cannot be interpreted according to its semantics such as an invalid image, a script execution error, or non-well-formed XML.

select

Type	select
Namespace	None
Interface	UIEvent if generated from a user interface, Event otherwise.
Cancelable	No
Bubbles	Yes
Target	Element
Context info	UIEvent.view may be in use.

A user selects some text. DOM Level 3 Events does not provide contextual information to access the selected text. The selection occurred before the dispatch of this event type.

change

Type	change
Namespace	None
Interface	UIEvent if generated from a user interface, Event otherwise.
Cancelable	No
Bubbles	Yes
Target	Element
Context info	UIEvent.view may be in use.

A control loses the input focus and its value has been modified since gaining focus. This event type is dispatched before the event type [blur](#).

submit

Type	submit
Namespace	None
Interface	UIEvent if generated from a user interface, Event otherwise.
Cancelable	Yes
Bubbles	Yes
Target	Element
Context info	UIEvent.view may be in use.

A form, such as a [[HTML 4.01](#)] or [[XHTML 1.0](#)] form, is submitted.

reset

Type	reset
Namespace	None
Interface	UIEvent if generated from a user interface, Event otherwise.
Cancelable	Yes
Bubbles	Yes
Target	Element
Context info	UIEvent.view may be in use.

A form, such as a [[HTML 4.01](#)] or [[XHTML 1.0](#)] form, is reset.

resize

Type	resize
Namespace	None

Interface	UIEvent
Cancelable	No
Bubbles	Yes
Target	Document, Element
Context info	UIEvent.view is in use.

A document view or an element has been resized. The resize occurred before the dispatch of this event type.

scroll

Type	scroll
Namespace	None
Interface	UIEvent
Cancelable	No
Bubbles	Yes
Target	Document, Element
Context info	UIEvent.view is in use.

A document view or an element has been scrolled. The scroll occurred before the dispatch of this event type.

[\[previous\]](#) [\[next\]](#) [\[contents\]](#) [\[index\]](#)