Document Object Model (DOM) Level 3 Events Specification

Version 1.0

W3C Working Draft 08 February 2002

This version:
http://www.w3.org/TR/2002/WD-DOM-Level-3-Events-20020208
(PostScript file , PDF file , plain text , ZIP file , single HTML file)
Latest version:
http://www.w3.org/TR/DOM-Level-3-Events
Previous version:
http://www.w3.org/TR/2001/WD-DOM-Level-3-Events-20010823

Editor:
Tom Pixley, *Netscape Communications Corporation*

Copyright ©2002 W3C® (MIT, INRIA, Keio), All Rights Reserved. W3C liability, trademark, document use and software licensing rules apply.

Abstract

This specification defines the Document Object Model Events Level 3, a platform- and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure and style of documents. The Document Object Model Events Level 3 builds on the Document Object Model Events Level 2 [DOM Level 2 Events].

Status of this document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. The latest status of this document series is maintained at the W3C.

This document contains the Document Object Model Level 3 Events specification.

This is a Working Draft for review by W3C members and other interested parties.

It is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to use W3C Working Drafts as reference material or to cite them as other than "work in progress". This is work in progress and does not imply endorsement by, or the consensus of, either W3C or members of the DOM Working Group.

Comments on this document are invited and are to be sent to the public mailing list www-dom@w3.org. An archive is available at http://lists.w3.org/Archives/Public/www-dom/.

This document has been produced as part of the W3C DOM Activity. The authors of this document are the DOM Working Group members.

A list of current W3C Recommendations and other technical documents can be found at http://www.w3.org/TR.

Table of contents

Expanded Table of Contents 3
Copyright Notice 5

1. Document Object Model Events 9

Appendix A: Changes 45
Appendix B: IDL Definitions 47
Appendix C: Java Language Binding 53
Appendix D: ECMAScript Language Binding 59
Glossary 67
References 69
Index 71

Expanded Table of Contents

Expanded Table of Contents 3
Copyright Notice 5
W3C Document Copyright Notice and License 5
W3C Software Copyright Notice and License 6

1. Document Object Model Events 9
1.1. Overview of the DOM Level 3 Event Model 9
1.1.1. Terminology 9
1.2. Description of event flow 10
1.2.1. Event listeners activation 10
1.2.2. Event capture 10
1.2.3. Event bubbling 11
1.2.4. Event cancelation 11
1.2.5. EventListener Grouping 12
1.3. Event listener registration 12
1.3.1. Event registration interfaces 12
1.3.2. Interaction with HTML 4.0 event listeners 18
1.4. Basic interfaces 18
1.4.1. Event creation 21
1.5. Event module definitions 23
1.5.1. User Interface event types 23
1.5.2. Mouse event types 25
1.5.3. Text events 29
1.5.4. Mutation event types 36
1.5.5. HTML event types 39
1.6. Issues 41

Appendix A: Changes 45
A.1. Changes between DOM Level 2 Events and DOM Level 3 Events 45
A.1.1. Changes to DOM Level 2 Events interfaces 45
A.1.2. New Interfaces 45
Appendix B: IDL Definitions 47
Appendix C: Java Language Binding 53
Appendix D: ECMAScript Language Binding 59
Glossary 67
References 69
1. Normative references 69
2. Informative references 69
Index 71

Expanded Table of Contents

Copyright Notice

Copyright © 2002 World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University). All Rights Reserved.

This document is published under the W3C Document Copyright Notice and License [p.5] . The bindings within this document are published under the W3C Software Copyright Notice and License [p.6] . The software license requires "Notice of any changes or modifications to the W3C files, including the date changes were made." Consequently, modified versions of the DOM bindings must document that they do not conform to the W3C standard; in the case of the IDL definitions, the pragma prefix can no longer be 'w3c.org'; in the case of the Java language binding, the package names can no longer be in the 'org.w3c' package.

W3C Document Copyright Notice and License

Note: This section is a copy of the W3C Document Notice and License and could be found at http://www.w3.org/Consortium/Legal/copyright-documents-19990405.

Copyright © 1994-2002 World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University). All Rights Reserved.

http://www.w3.org/Consortium/Legal/

Public documents on the W3C site are provided by the copyright holders under the following license. The software or Document Type Definitions (DTDs) associated with W3C specifications are governed by the Software Notice. By using and/or copying this document, or the W3C document from which this statement is linked, you (the licensee) agree that you have read, understood, and will comply with the following terms and conditions:

Permission to use, copy, and distribute the contents of this document, or the W3C document from which this statement is linked, in any medium for any purpose and without fee or royalty is hereby granted, provided that you include the following on *ALL* copies of the document, or portions thereof, that you use:

1. A link or URL to the original W3C document.
2. The pre-existing copyright notice of the original author, or if it doesn't exist, a notice of the form: "Copyright © [$date-of-document] World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University). All Rights Reserved. http://www.w3.org/Consortium/Legal/" (Hypertext is preferred, but a textual representation is permitted.)
3. *If it exists*, the STATUS of the W3C document.

When space permits, inclusion of the full text of this **NOTICE** should be provided. We request that authorship attribution be provided in any software, documents, or other items or products that you create pursuant to the implementation of the contents of this document, or any portion thereof.

No right to create modifications or derivatives of W3C documents is granted pursuant to this license. However, if additional requirements (documented in the Copyright FAQ) are satisfied, the right to create modifications or derivatives is sometimes granted by the W3C to individuals complying with those requirements.

THIS DOCUMENT IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE DOCUMENT ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE DOCUMENT OR THE PERFORMANCE OR IMPLEMENTATION OF THE CONTENTS THEREOF.

The name and trademarks of copyright holders may NOT be used in advertising or publicity pertaining to this document or its contents without specific, written prior permission. Title to copyright in this document will at all times remain with copyright holders.

W3C Software Copyright Notice and License

Note: This section is a copy of the W3C Software Copyright Notice and License and could be found at http://www.w3.org/Consortium/Legal/copyright-software-19980720

Copyright © 1994-2002 World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University). All Rights Reserved.

http://www.w3.org/Consortium/Legal/

This W3C work (including software, documents, or other related items) is being provided by the copyright holders under the following license. By obtaining, using and/or copying this work, you (the licensee) agree that you have read, understood, and will comply with the following terms and conditions:

Permission to use, copy, and modify this software and its documentation, with or without modification, for any purpose and without fee or royalty is hereby granted, provided that you include the following on ALL copies of the software and documentation or portions thereof, including modifications, that you make:

1. The full text of this NOTICE in a location viewable to users of the redistributed or derivative work.
2. Any pre-existing intellectual property disclaimers. If none exist, then a notice of the following form: "Copyright © [$date-of-software] World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University). All Rights Reserved. http://www.w3.org/Consortium/Legal/."

3. Notice of any changes or modifications to the W3C files, including the date changes were made. (We recommend you provide URIs to the location from which the code is derived.)

THIS SOFTWARE AND DOCUMENTATION IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE OR DOCUMENTATION WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE SOFTWARE OR DOCUMENTATION.

The name and trademarks of copyright holders may NOT be used in advertising or publicity pertaining to the software without specific, written prior permission. Title to copyright in this software and any associated documentation will at all times remain with copyright holders.

W3C Software Copyright Notice and License

1. Document Object Model Events

Editor:
Tom Pixley, Netscape Communications Corporation

1.1. Overview of the DOM Level 3 Event Model

The DOM Level 3 Event Model is designed with two main goals. The first goal is the design of a generic event system which allows registration of event handlers, describes event flow through a tree structure, and provides basic contextual information for each event. 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 the event model is to provide a common subset of the current event systems used in *DOM Level 0* [p.67] 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.

The following sections of the Event Model specification define both the specification for the DOM Event Model and a number of conformant event modules designed for use within the model. The Event Model consists of the two sections on event propagation and event listener registration and the Event interface.

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 the event module is supported by the implementation. In order to fully support this module, an implementation must also support the "Core" feature defined in the DOM Level 3 Core specification [DOM Level 3 Core]. Please, refer to additional information about *conformance* in the DOM Level 3 Core specification [DOM Level 3 Core]. The DOM Level 3 Event module is backward compatible with the DOM Level 2 Events [DOM Level 2 Events] module, i.e. a DOM Level 3 Events implementation who returns `true` for "Events" with the `version` number `"3.0"` must also return `true` for this `feature` when the `version` number is `"2.0"`, `""` or, `null`.

Each event module describes its own feature string in the event module listing.

1.1.1. Terminology

UI events
: User interface events. These events are generated by user interaction through an external device (mouse, keyboard, etc.)

UI Logical events
: Device independent user interface events such as focus change messages or element triggering notifications.

Mutation events
: Events caused by any action which modifies the structure of the document.

Capturing
: The process by which an event can be handled by one of the event's target's *ancestors* [p.67] before being handled by the event's target.

Bubbling
: The process by which an event propagates upward through its *ancestors* [p.67] after being handled by the event's target.

Cancelable
: A designation for events which indicates that upon handling the event the client may choose to prevent the DOM implementation from processing any default action associated with the event.

1.2. Description of event flow

Event flow is the process through which the an event originates from the DOM implementation and is passed into the Document Object Model. The methods of event capture and event bubbling, along with various event listener registration techniques, allow the event to then be handled in a number of ways. It can be handled locally at the `EventTarget` [p.12] level or centrally from an `EventTarget` higher in the document tree. This results in three phases in event flow: the event capture (`CAPTURING_PHASE`), at the `EventTarget` (`AT_TARGET`), and the event bubbling (`BUBBLING_PHASE`).

1.2.1. Event listeners activation

Each event has an `EventTarget` [p.12] toward which the event is directed by the DOM implementation. This `EventTarget` is specified in the `Event` [p.18] 's `target` attribute. When the event reaches the target, any event listeners registered on the `EventTarget` are triggered. Although all `EventListeners` [p.17] on the `EventTarget` are guaranteed to be triggered by any event which is received by that `EventTarget`, no specification is made as to the order in which they will receive the event with regards to the other `EventListeners` [p.17] on the `EventTarget`.

Any exceptions thrown inside an `EventListener` [p.17] will not stop propagation of the event. It will continue processing any additional `EventListener` in the described manner.

It is expected that actions taken by `EventListener` [p.17] s may cause additional events to fire. Additional events should be handled in a synchronous manner and may cause reentrancy into the event model.

1.2.2. Event capture

Event capture is the process by which an EventListener registered on an *ancestor* [p.67] of the event's target can intercept events of a given type before they are received by the event's target. Capture operates from the top of the tree, generally the `Document`, downward, making it the symmetrical opposite of bubbling which is described below. The chain of `EventTarget` [p.12] s from the top of the tree to the event's target is determined before the initial dispatch of the event. If modifications occur to the tree during event processing, event flow will proceed based on the initial state of the tree.

An `EventListener` [p.17] being registered on an `EventTarget` [p.12] may choose to have that `EventListener` capture events by specifying the `useCapture` parameter of the `addEventListener` method to be `true`. Thereafter, when an event of the given type is dispatched toward a *descendant* [p.67] of the capturing object, the event will trigger any capturing event listeners of the appropriate type which exist in the direct line between the top of the document and the event's target. This downward propagation continues until the event's target is reached. A capturing `EventListener` will not be triggered by events dispatched directly to the `EventTarget` upon which it is registered. Any type of event can be captured.

If the capturing `EventListener` [p.17] wishes to prevent further processing of the event from occurring it may call the `stopProgagation` method of the `Event` [p.18] interface. This will prevent further dispatch of the event, although additional `EventListeners` registered at the same hierarchy level will still receive the event. Once an event's `stopPropagation` method has been called, further calls to that method have no additional effect. If no additional capturers exist and `stopPropagation` has not been called, the event triggers the appropriate `EventListeners` on the target itself.

Although event capture is similar to the delegation based event model in which all interested parties register their listeners directly on the target about which they wish to receive notifications, it is different in two important respects. First, event capture only allows interception of events which are targeted at *descendants* [p.67] of the capturing `EventTarget` [p.12] . It does not allow interception of events targeted to the capturer's *ancestors* [p.67] , its *siblings* [p.67] , or its sibling's *descendants* [p.67] . Secondly, event capture is not specified for a single `EventTarget`, it is specified for a specific type of event. Once specified, event capture intercepts all events of the specified type targeted toward any of the capturer's *descendants* [p.67] .

1.2.3. Event bubbling

Events which are designated as bubbling will initially proceed with the same event flow as non-bubbling events. The event is dispatched to its target `EventTarget` [p.12] and any event listeners found there are triggered. Bubbling events will then trigger any additional event listeners found by following the `EventTarget`'s parent chain upward, checking for any event listeners registered on each successive `EventTarget`. This upward propagation will continue up to and including the `Document`. `EventListener` [p.17] s registered as capturers will not be triggered during this phase. The chain of `EventTargets` from the event target to the top of the tree is determined before the initial dispatch of the event. If modifications occur to the tree during event processing, event flow will proceed based on the initial state of the tree.

Any event handler may choose to prevent further event propagation by calling the `stopPropagation` method of the `Event` [p.18] interface. If any `EventListener` [p.17] calls this method, all additional `EventListeners` on the current `EventTarget` [p.12] will be triggered but bubbling will cease at that level. Only one call to `stopPropagation` is required to prevent further bubbling.

1.2.4. Event cancelation

Some events are specified as cancelable. For these events, the DOM implementation generally has a default action associated with the event. An example of this is a hyperlink in a Web browser. When the user clicks on the hyperlink the default action is generally to activate that hyperlink. Before processing these events, the implementation must check for event listeners registered to receive the event and dispatch the event to those listeners. These listeners then have the option of canceling the implementation's 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.

Cancelation is accomplished by calling the `Event` [p.18] 's `preventDefault` method. If one or more `EventListeners` [p.17] call `preventDefault` during any phase of event flow the default action will be canceled.

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

1.2.5. EventListener Grouping

EventListener grouping is intended to allow groups of `EventListener` [p.17] s to be registered which will each have independent event flow within them which is not affected by changes to event flow in any other group. This may be used to control events separately in multiple views on a document. It may also be used to develop an application which uses events without the problem of possible interference by other applications running within the same document.

The new methods added for EventListener grouping should not interfere with the non-groups methods. For purposes of interoperability between the groups and non-groups methods, the implementation can be assumed to define a default `EventGroup` [p.16] . This default `EventGroup` is implicitly used in the registration of all `EventListener` [p.17] s registered via methods which do not specify an `EventGroup` (`addEventListener`, `removeEventListener`).

1.3. Event listener registration

1.3.1. Event registration interfaces

Interface *EventTarget* (introduced in **DOM Level 2**)

> The `EventTarget` interface is implemented by all `Nodes` in an implementation which supports the DOM Event Model. Therefore, this interface can be obtained by using binding-specific casting methods on an instance of the `Node` interface. The interface allows registration and removal of `EventListeners` [p.17] on an `EventTarget` and dispatch of events to that `EventTarget`.
>
> **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);
  boolean            dispatchEvent(in Event evt)
                                   raises(EventException);
  // Introduced in DOM Level 3:
  void               addGroupedEventListener(in DOMString type,
                                             in EventListener listener,
                                             in boolean useCapture,
                                             in EventGroup evtGroup);
  // Introduced in DOM Level 3:
  void               removeGroupedEventListener(in DOMString type,
                                                in EventListener listener,
                                                in boolean useCapture,
                                                in EventGroup evtGroup);
  // Introduced in DOM Level 3:
  boolean            canTrigger(in DOMString type);
  // Introduced in DOM Level 3:
  boolean            isRegisteredHere(in DOMString type);
};
```

Methods

`addEventListener`

This method allows the registration of event listeners on the event target. If an `EventListener` [p.17] is added to an `EventTarget` while it is processing an event, the `EventListener` will not be triggered by the current actions but may be triggered during a later stage of event flow, such as the bubbling phase.

If multiple identical `EventListener` [p.17] s are registered on the same `EventTarget` with the same parameters the duplicate instances are discarded. They do not cause the `EventListener` to be called twice and since they are discarded they do not need to be removed with the `removeEventListener` method.

Parameters

`type` of type `DOMString`

The event type for which the user is registering

`listener` of type `EventListener` [p.17]

The `listener` parameter takes an interface implemented by the user which contains the methods to be called when the event occurs.

`useCapture` of type `boolean`

If true, `useCapture` indicates that the user wishes to initiate capture. After initiating capture, all events of the specified type will be dispatched to the registered `EventListener` before being dispatched to any `EventTargets` beneath them in the tree. Events which are bubbling upward through the tree will not trigger an `EventListener` designated to use capture.

No Return Value

No Exceptions

`addGroupedEventListener` introduced in **DOM Level 3**

This method allows the registration of event listeners on the event target in the specified group. If an `EventListener` [p.17] is added to an `EventTarget` while it is processing an event, the `EventListener` will not be triggered by the current actions, independently of the event groups, but may be triggered during a later stage of event flow, such as the bubbling phase.

If multiple identical `EventListener` [p.17] s are registered on the same `EventTarget` with the same parameters the duplicate instances are discarded. They do not cause the `EventListener` to be called twice and since they are discarded they do not need to be removed with the `removeGroupedEventListener` method.

Parameters

`type` of type `DOMString`

The event type for which the user is registering

`listener` of type `EventListener` [p.17]

The `listener` parameter takes an interface implemented by the user which contains the methods to be called when the event occurs.

`useCapture` of type `boolean`

If `true`, `useCapture` indicates that the user wishes to initiate capture. After initiating capture, all events of the specified type will be dispatched to the registered `EventListener` before being dispatched to any `EventTargets` beneath them in the tree. Events which are bubbling upward through the tree will not trigger an `EventListener` designated to use capture.

`evtGroup` of type `EventGroup` [p.16]

The `EventGroup` to associate with the `EventListener`.

No Return Value

No Exceptions

`canTrigger` introduced in **DOM Level 3**

This method allows the DOM application to know if an event listener, attached to this `EventTarget` or one of its ancestors, will be triggered by the specified event type during the dispatch of the event to this event target or one of its descendants.

Parameters

`type` of type `DOMString`

The event type for which the `EventListener` [p.17] is registered.

Return Value

`boolean`	`true` if an event listener will be triggered on the event target with the specified event type, `false` otherwise.

No Exceptions

`dispatchEvent`

This method allows the dispatch of events into the implementations event model. Events dispatched in this manner will have the same capturing and bubbling behavior as events dispatched directly by the implementation. The target of the event is the `EventTarget` on which `dispatchEvent` is called.

Parameters

`evt` of type `Event` [p.18]
Specifies the event type, behavior, and contextual information to be used in processing the event.

Return Value

`boolean`	The return value of `dispatchEvent` indicates whether any of the listeners which handled the event called `preventDefault`. If `preventDefault` was called the value is `false`, else the value is `true`.

Exceptions

`EventException` [p.20]	UNSPECIFIED_EVENT_TYPE_ERR: Raised if the `Event` [p.18] 's type was not specified by initializing the event before `dispatchEvent` was called. Specification of the `Event`'s type as `null` or an empty string will also trigger this exception.

`isRegisteredHere` introduced in **DOM Level 3**
This method allows the DOM application to know if this event target contains an event listener registered for the specified event type. This is useful for determining at which nodes within a hierarchy altered handling of specific event types has been introduced, but should not be used to determine whether the specified event type triggers a listener (see `canTrigger`).
Issue canTriggerOnTarget-useCapture:
do we need a useCapture parameter?
Resolution: No use case for that.

Parameters

`type` of type `DOMString`
The event type for which the `EventListener` [p.17] is registered.

Return Value

`boolean`	`true` if an event listener is registered on this EventTarget for the specified event type, `false` otherwise.

No Exceptions

`removeEventListener`
This method allows the removal of event listeners from the event target. If an `EventListener` [p.17] is removed from an `EventTarget` while it is processing an event, it will not be triggered by the current actions. `EventListeners` can never be invoked after being removed.
Calling `removeEventListener` with arguments which do not identify any currently registered `EventListener` [p.17] on the `EventTarget` has no effect.

Parameters

`type` of type `DOMString`
Specifies the event type of the `EventListener` [p.17] being removed.
`listener` of type `EventListener` [p.17]
The `EventListener` parameter indicates the `EventListener` to be removed.
`useCapture` of type `boolean`
Specifies whether the `EventListener` being removed was registered as a capturing listener or not. If a listener was registered twice, once with capture and once without, each must be removed separately. Removal of a capturing listener does not affect a non-capturing version of the same listener, and vice versa.
No Return Value
No Exceptions

`removeGroupedEventListener` introduced in **DOM Level 3**
This method allows the removal of event listeners from the event target. If an `EventListener` [p.17] is removed from an `EventTarget` while it is processing an event, it will not be triggered by the current actions, independently of the event groups. `EventListeners` can never be invoked after being removed.
Calling `removeGroupedEventListener` with arguments which do not identify any currently registered `EventListener` [p.17] on the `EventTarget` has no effect.
Parameters
`type` of type `DOMString`
Specifies the event type of the `EventListener` [p.17] being removed.
`listener` of type `EventListener` [p.17]
The `EventListener` parameter indicates the `EventListener` to be removed.
`useCapture` of type `boolean`
Specifies whether the `EventListener` being removed was registered as a capturing listener or not. If a listener was registered twice, once with capture and once without, each must be removed separately. Removal of a capturing listener does not affect a non-capturing version of the same listener, and vice versa.
`evtGroup` of type `EventGroup` [p.16]
The `EventGroup` to associate with the `EventListener`.
No Return Value
No Exceptions

Interface *EventGroup* (introduced in **DOM Level 3**)

The EventGroup interface functions primarily as a placeholder for separating the event flows when there are multiple groups of listeners for a DOM tree.

`EventListener` [p.17] s can be registered without an `EventGroup` using the existing `EventTarget` [p.12] interface, or with an associated `EventGroup` using the new `EventTargetGroup` interface. When an event is dispatched, it is dispatched independently to each `EventGroup`. In particular, the `stopPropagation` method of the `Event` [p.18] interface only stops propagation within an `EventListener`'s associated `EventGroup`.

IDL Definition

```
// Introduced in DOM Level 3:
interface EventGroup {
  boolean            isSameEventGroup(in EventGroup other);
};
```

Methods

`isSameEventGroup`

This method checks if the supplied `EventGroup` is the same as the `EventGroup` upon which the method is called.

Parameters

`other` of type `EventGroup` [p.16]

The `EventGroup` with which to check equality.

Return Value

`boolean` Returns true if the `EventGroups` are equal, else returns false.

No Exceptions

Interface *EventListener* (introduced in **DOM Level 2**)

The `EventListener` interface is the primary method for handling events. Users implement the `EventListener` interface and register their listener on an `EventTarget` [p.12] using the `AddEventListener` method. The users should also remove their `EventListener` from its `EventTarget` after they have completed using the listener.

When a `Node` is copied using the `cloneNode` method the `EventListeners` attached to the source `Node` are not attached to the copied `Node`. If the user wishes the same `EventListeners` to be added to the newly created copy the user must add them manually.

When a `Node` is adopted using the `adoptNode` method the `EventListeners` attached to the source `Node` stay attached to the adopted `Node`.

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 type for which the `EventListener` interface was registered.

Parameters

`evt` of type `Event` [p.18]

The `Event` contains contextual information about the event. It also contains the `stopPropagation` and `preventDefault` methods which are used in determining the event's flow and default action.

No Return Value

No Exceptions

1.3.2. Interaction with HTML 4.0 event listeners

In HTML 4.0, event listeners were specified as attributes of an element. As such, registration of a second event listener of the same type would replace the first listener. The DOM Event Model allows registration of multiple event listeners on a single `EventTarget` [p.12] . To achieve this, event listeners are no longer stored as attribute values.

In order to achieve compatibility with HTML 4.0, implementors may view the setting of attributes which represent event handlers as the creation and registration of an `EventListener` on the `EventTarget` [p.12] . The value of `useCapture` defaults to `false`. This `EventListener` [p.17] behaves in the same manner as any other `EventListeners` which may be registered on the `EventTarget`. If the attribute representing the event listener is changed, this may be viewed as the removal of the previously registered `EventListener` and the registration of a new one. No technique is provided to allow HTML 4.0 event listeners access to the context information defined for each event.

1.4. Basic interfaces

Interface *Event* (introduced in **DOM Level 2**)

The `Event` interface is used to provide contextual information about an event to the handler processing the event. An object which implements the `Event` interface is generally passed as the first parameter to an event handler. More specific context information is passed to event handlers by deriving additional interfaces from `Event` which contain information directly relating to the type of event they accompany. These derived interfaces are also implemented by the object passed to the event listener.

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);
};
```

Definition group *PhaseType*

An integer indicating which phase of event flow is being processed.

Defined Constants

`AT_TARGET`
: The event is currently being evaluated at the target `EventTarget` [p.12] .

`BUBBLING_PHASE`
: The current event phase is the bubbling phase.

`CAPTURING_PHASE`
: The current event phase is the capturing 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, else the value is false.

`cancelable` of type `boolean`, readonly
: Used to indicate whether or not an event can have its default action prevented. If the default action can be prevented the value is true, else the value is false.

`currentTarget` of type `EventTarget` [p.12] , readonly
: Used to indicate the `EventTarget` [p.12] whose `EventListeners` [p.17] are currently being processed. This is particularly useful during capturing and bubbling.

`eventPhase` of type `unsigned short`, readonly
: Used to indicate which phase of event flow is currently being evaluated.

`target` of type `EventTarget` [p.12] , readonly
: Used to indicate the `EventTarget` [p.12] to which the event was originally dispatched.

`timeStamp` of type `DOMTimeStamp`, readonly
: Used to specify the time (in milliseconds relative to the epoch) at which the event was created. 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, a value of 0 will be returned. Examples of epoch time are the time of the system start or 0:0:0 UTC 1st January 1970.

`type` of type `DOMString`, readonly
: The name of the event (case-insensitive). The name must be an *XML name* [p.67] . On retrieval, the name of the event is in lowercase.

  Issue lowercase-1:
  : check implementation for lowercase.

  Issue naming-1:
  : XML events might (will?) use {namespaceURI,localName} to indicate the type of events in the future. What should we do about it?

Methods

`initEvent`
: The `initEvent` method is used to initialize the value of an `Event` created through the `DocumentEvent` [p.21] interface. This method may only be called before the `Event` has been dispatched via the `dispatchEvent` method, though it may be called multiple times during that phase if necessary. If called multiple times the final invocation takes precedence. If called from a subclass of `Event` interface only the values specified in the `initEvent` method are modified, all other attributes are left unchanged.

Parameters

`eventTypeArg` of type `DOMString`

Specifies the event type. This type may be any event type currently defined in this specification or a new event type.. The string must be an *XML name* [p.67] .
Any new event type must not begin with any upper, lower, or mixed case version of the string "DOM". This prefix is reserved for future DOM event sets. It is also strongly recommended that third parties adding their own events use their own prefix to avoid confusion and lessen the probability of conflicts with other new events.

`canBubbleArg` of type `boolean`

Specifies whether or not the event can bubble.

`cancelableArg` of type `boolean`

Specifies whether or not the event's default action can be prevented.

No Return Value

No Exceptions

`preventDefault`

If an event is cancelable, the `preventDefault` method is used to signify 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. If, during any stage of event flow, the `preventDefault` method is called the event is canceled. Any default action associated with the event will not occur. Calling this method for a non-cancelable event has no effect. Once `preventDefault` has been called it will remain in effect throughout the remainder of the event's propagation. This method may be used during any stage of event flow.

No Parameters

No Return Value

No Exceptions

`stopPropagation`

The `stopPropagation` method is used to prevent further propagation of an event in the current group during event flow (see also EventListener Grouping [p.12]). If this method is called by any `EventListener` [p.17] the event will cease propagating in the current group through the tree. The event will complete dispatch to all listeners on the current `EventTarget` [p.12] before event flow stops. This method may be used during any stage of event flow.

No Parameters

No Return Value

No Exceptions

Exception *EventException* introduced in **DOM Level 2**

Event operations may throw an `EventException` [p.20] 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;
```

Definition group *EventExceptionCode*

An integer indicating the type of error generated.

Defined Constants

`UNSPECIFIED_EVENT_TYPE_ERR`

If the `Event` [p.18] 's type was not specified by initializing the event before the method was called. Specification of the Event's type as `null` or an empty string will also trigger this exception.

1.4.1. Event creation

Interface *DocumentEvent* (introduced in **DOM Level 2**)

The `DocumentEvent` interface provides a mechanism by which the user can create an Event of a type supported by the implementation. It is expected that the `DocumentEvent` interface will be implemented on the same object which implements the `Document` interface in an implementation which supports the Event model.

IDL Definition

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

Methods

`createEvent`

Parameters

`eventType` of type `DOMString`

The `eventType` parameter specifies the type of `Event` [p.18] interface to be created. If the `Event` interface specified is supported by the implementation this method will return a new `Event` of the interface type requested. If the `Event` is to be dispatched via the `dispatchEvent` method the appropriate event init 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` [p.23] would call `createEvent` with the parameter "UIEvents". The `initUIEvent` method could then be called on the newly created `UIEvent` to set the specific type of UIEvent to be dispatched and set its context information.

The `createEvent` method is used in creating `Event` [p.18] s when it is either inconvenient or unnecessary for the user to create an `Event` themselves. In cases where the implementation provided `Event` is insufficient, users may supply their own `Event` implementations for use with the `dispatchEvent` method. However, the DOM implementation needs access to the attributes `currentTarget` and `eventPhase` of the `Event` interface to propagate appropriately the event in the DOM tree. Therefore users `Event` implementation might need to support the `CustomEvent` [p.22] for that effect.

Return Value

`Event` [p.18]	The newly created `Event`

Exceptions

`DOMException`	NOT_SUPPORTED_ERR: Raised if the implementation does not support the type of `Event` [p.18] interface requested

`createEventGroup`

This method creates a new `EventGroup` [p.16] for use in the `addGroupedEventListener` and `removeGroupedEventListener` methods of the `EventTarget` [p.12] interface.

Return Value

`EventGroup` [p.16]	The newly created `EventGroup`.

No Parameters

No Exceptions

Interface *CustomEvent* (introduced in **DOM Level 3**)

The `CustomEvent` interface provides user defined events. It is intented to be used by the DOM implementation to access the underlying while propagating the event in the tree. Both methods should be call before invoking each event listener on the current target.

IDL Definition

```
// Introduced in DOM Level 3:
interface CustomEvent : Event {
  void               setCurrentTarget(in Node target);
  void               setEventPhase(in unsigned short phase);
};
```

Methods

`setCurrentTarget`

The `setCurrentTarget` method is used by the DOM implementation to change the value of a `currentTarget` attribute on the `Event` [p.18] interface.

Parameters

`target` of type `Node`

Specifies the `currentTarget` attribute on the `Event` [p.18] interface.

No Return Value

No Exceptions

`setEventPhase`

The `setEventPhase` method is used by the DOM implementation to change the value of a `eventPhase` attribute on the `Event` [p.18] interface.

Parameters

`phase` of type `unsigned short`
Specifies the `eventPahse` attribute on the `Event` [p.18] interface.
No Return Value
No Exceptions

1.5. Event module definitions

The DOM Level 2 Event Model allows a DOM implementation to support multiple modules of events. The model has been designed to allow addition of new event modules as is required. The DOM will not attempt to define all possible events. For purposes of interoperability, the DOM will define a module of user interface events including lower level device dependent events, a module of UI logical events, and a module of document mutation events. Any new event types defined by third parties must not begin with any upper, lower, or mixed case version of the string "DOM". This prefix is reserved for future DOM event modules. It is also strongly recommended that third parties adding their own events use their own prefix to avoid confusion and lessen the probability of conflicts with other new events.

1.5.1. User Interface event types

The User Interface event module is composed of events listed in HTML 4.0 and additional events which are supported in *DOM Level 0* [p.67] browsers.

A DOM application may use the `hasFeature(feature, version)` method of the `DOMImplementation` interface with parameter values "UIEvents" and "3.0" (respectively) to determine whether or not the User Interface event module is supported by the implementation. In order to fully support this module, an implementation must also support the "Events" feature defined in this specification and the "Views" feature defined in the DOM Level 2 Views specification [DOM Level 2 Views]. Please, refer to additional information about *conformance* in the DOM Level 3 Core specification [DOM Level 3 Core]. The DOM Level 3 User Interface Events module is backward compatible with the DOM Level 2 User Interface Events [DOM Level 2 Events] module, i.e. a DOM Level 3 User Interface Events implementation who returns `true` for "UIEvents" with the `version` number `"3.0"` must also return `true` for this `feature` when the `version` number is `"2.0"`, `""` or, `null`.

Note: To create an instance of the `UIEvent` [p.23] interface, use the feature string "UIEvents" as the value of the input parameter used with the `createEvent` method of the `DocumentEvent` [p.21] interface.

Interface *UIEvent* (introduced in **DOM Level 2**)

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

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);
};
```

Attributes

`detail` of type `long`, readonly

Specifies some detail information about the `Event` [p.18] , 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`

The `initUIEvent` method is used to initialize the value of a `UIEvent` created through the `DocumentEvent` [p.21] interface. This method may only be called before the `UIEvent` has been dispatched via the `dispatchEvent` method, though it may be called multiple times during that phase if necessary. If called multiple times, the final invocation takes precedence.

Parameters

`typeArg` of type `DOMString`

Specifies the event type.

`canBubbleArg` of type `boolean`

Specifies whether or not the event can bubble.

`cancelableArg` of type `boolean`

Specifies whether or not the event's default action can be prevented.

`viewArg` of type `views::AbstractView`

Specifies the `Event` [p.18] 's `AbstractView`.

`detailArg` of type `long`

Specifies the `Event` [p.18] 's detail.

No Return Value

No Exceptions

The different types of such events that can occur are:

DOMFocusIn

The DOMFocusIn event occurs when an `EventTarget` [p.12] receives focus, for instance via a pointing device being moved onto an element or by tabbing navigation to the element. Unlike the HTML event focus, DOMFocusIn can be applied to any focusable `EventTarget`, not just FORM controls.

- Bubbles: Yes
- Cancelable: No
- Context Info: None

DOMFocusOut

The DOMFocusOut event occurs when a `EventTarget` [p.12] loses focus, for instance via a pointing device being moved out of an element or by tabbing navigation out of the element. Unlike the HTML event blur, DOMFocusOut can be applied to any focusable `EventTarget`, not just FORM controls.

- Bubbles: Yes
- Cancelable: No
- Context Info: None

DOMActivate

The activate event occurs when an element is activated, for instance, thru a mouse click or a keypress. A numerical argument is provided to give an indication of the type of activation that occurs: 1 for a simple activation (e.g. a simple click or Enter), 2 for hyperactivation (for instance a double click or Shift Enter).

- Bubbles: Yes
- Cancelable: Yes
- Context Info: detail (the numerical value)

1.5.2. Mouse event types

The Mouse event module is composed of events listed in HTML 4.0 and additional events which are supported in *DOM Level 0* [p.67] browsers. This event module is specifically designed for use with mouse input devices.

A DOM application may use the `hasFeature(feature, version)` method of the `DOMImplementation` interface with parameter values "MouseEvents" and "3.0" (respectively) to determine whether or not the Mouse event module is supported by the implementation. In order to fully support this module, an implementation must also support the "UIEvents" feature defined in this specification. Please, refer to additional information about *conformance* in the DOM Level 3 Core specification [DOM Level 3 Core]. The DOM Level 3 Mouse Events module is backward compatible with the DOM Level 2 Mouse Events [DOM Level 2 Events] module, i.e. a DOM Level 3 Mouse Events implementation who returns `true` for "MouseEvents" with the `version` number `"3.0"` must also return `true` for this `feature` when the `version` number is `"2.0"`, `""` or, `null`.

Note: To create an instance of the `MouseEvent` [p.25] interface, use the feature string "MouseEvents" as the value of the input parameter used with the `createEvent` method of the `DocumentEvent` [p.21] interface.

Interface *MouseEvent* (introduced in **DOM Level 2**)

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

The `detail` attribute inherited from `UIEvent` [p.23] indicates the number of times a mouse button has been pressed and released over the same screen location during a user action. The attribute value is 1 when the user begins this action and increments by 1 for each full sequence of pressing and releasing. If the user moves the mouse between the mousedown and mouseup the value will be set to 0, indicating that no click is occurring.

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 its descendent elements.

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);
};
```

Attributes

`altKey` of type `boolean`, readonly
: Used to indicate whether the 'alt' key was depressed during the firing of the event. On some platforms this key may map to an alternative key name.

`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. The values for `button` range from zero to indicate the left button of the mouse, one to indicate the middle button if present, and two to indicate the right button. For mice configured for left handed use in which the button actions are reversed the values are instead read from right to left.

`clientX` of type `long`, readonly
: The horizontal coordinate at which the event occurred relative to the DOM implementation's client area.

`clientY` of type `long`, readonly
: The vertical coordinate at which the event occurred relative to the DOM implementation's client area.

`ctrlKey` of type `boolean`, readonly
: Used to indicate whether the 'ctrl' key was depressed during the firing of the event.

`metaKey` of type `boolean`, readonly
: Used to indicate whether the 'meta' key was depressed during the firing of the event. On some platforms this key may map to an alternative key name.

`relatedTarget` of type `EventTarget` [p.12] , readonly
: Used to identify a secondary `EventTarget` [p.12] related to a UI event. Currently this attribute is used with the mouseover event to indicate the `EventTarget` which the pointing device exited and with the mouseout event to indicate the `EventTarget` which the pointing device entered.

`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
: Used to indicate whether the 'shift' key was depressed during the firing of the event.

Methods

`initMouseEvent`
: The `initMouseEvent` method is used to initialize the value of a `MouseEvent` created through the `DocumentEvent` [p.21] interface. This method may only be called before the `MouseEvent` has been dispatched via the `dispatchEvent` method, though it may be called multiple times during that phase if necessary. If called multiple times, the final invocation takes precedence.

  Parameters

  `typeArg` of type `DOMString`
  : Specifies the event type.

  `canBubbleArg` of type `boolean`
  : Specifies whether or not the event can bubble.

  `cancelableArg` of type `boolean`
  : Specifies whether or not the event's default action can be prevented.

  `viewArg` of type `views::AbstractView`
  : Specifies the `Event` [p.18] 's `AbstractView`.

  `detailArg` of type `long`
  : Specifies the `Event` [p.18] 's mouse click count.

  `screenXArg` of type `long`
  : Specifies the `Event` [p.18] 's screen x coordinate

  `screenYArg` of type `long`
  : Specifies the `Event` [p.18] 's screen y coordinate

  `clientXArg` of type `long`
  : Specifies the `Event` [p.18] 's client x coordinate

  `clientYArg` of type `long`
  : Specifies the `Event` [p.18] 's client y coordinate

  `ctrlKeyArg` of type `boolean`
  : Specifies whether or not control key was depressed during the `Event` [p.18] .

`altKeyArg` of type `boolean`
: Specifies whether or not alt key was depressed during the `Event` [p.18] .

`shiftKeyArg` of type `boolean`
: Specifies whether or not shift key was depressed during the `Event` [p.18] .

`metaKeyArg` of type `boolean`
: Specifies whether or not meta key was depressed during the `Event` [p.18] .

`buttonArg` of type `unsigned short`
: Specifies the `Event` [p.18] 's mouse button.

`relatedTargetArg` of type `EventTarget` [p.12]
: Specifies the `Event` [p.18] 's related `EventTarget`.

No Return Value

No Exceptions

The different types of Mouse events that can occur are:

click
: The click event occurs when the pointing device button is clicked over an element. A click is defined as a mousedown and mouseup over the same screen location. The sequence of these events is:

```
mousedown
mouseup
click
```

If multiple clicks occur at the same screen location, the sequence repeats with the `detail` attribute incrementing with each repetition. This event is valid for most elements.

- Bubbles: Yes
- Cancelable: Yes
- Context Info: screenX, screenY, clientX, clientY, altKey, ctrlKey, shiftKey, metaKey, button, detail, view

mousedown
: The mousedown event occurs when the pointing device button is pressed over an element. This event is valid for most elements.

- Bubbles: Yes
- Cancelable: Yes
- Context Info: screenX, screenY, clientX, clientY, altKey, ctrlKey, shiftKey, metaKey, button, detail, view

mouseup
: The mouseup event occurs when the pointing device button is released over an element. This event is valid for most elements.

- Bubbles: Yes
- Cancelable: Yes
- Context Info: screenX, screenY, clientX, clientY, altKey, ctrlKey, shiftKey, metaKey, button, detail, view

mouseover
: The mouseover event occurs when the pointing device is moved onto an element. This event is valid for most elements.

- Bubbles: Yes

- Cancelable: Yes
- Context Info: view, screenX, screenY, clientX, clientY, altKey, ctrlKey, shiftKey, metaKey, relatedTarget indicates the `EventTarget` [p.12] the pointing device is exiting.

mousemove

The mousemove event occurs when the pointing device is moved while it is over an element. This event is valid for most elements.

- Bubbles: Yes
- Cancelable: No
- Context Info: view, screenX, screenY, clientX, clientY, altKey, ctrlKey, shiftKey, metaKey

mouseout

The mouseout event occurs when the pointing device is moved away from an element. This event is valid for most elements..

- Bubbles: Yes
- Cancelable: Yes
- Context Info: view, screenX, screenY, clientX, clientY, altKey, ctrlKey, shiftKey, metaKey, relatedTarget indicates the `EventTarget` [p.12] the pointing device is entering.

1.5.3. Text events

A DOM application may use the `hasFeature(feature, version)` method of the `DOMImplementation` interface with parameter values "TextEvents" and "3.0" (respectively) to determine whether or not the Text event module is supported by the implementation. In order to fully support this module, an implementation must also support the "UIEvents" feature defined in this specification. Please, refer to additional information about *conformance* in the DOM Level 3 Core specification [DOM Level 3 Core].

Note: To create an instance of the `TextEvent` [p.29] interface, use the feature string "TextEvents" as the value of the input parameter used with the `createEvent` method of the `DocumentEvent` [p.21] interface.

Interface *TextEvent* (introduced in **DOM Level 3**)

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

IDL Definition

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

  // VirtualKeyCode
  const unsigned long       DOM_VK_UNDEFINED              = 0x0;
  const unsigned long       DOM_VK_RIGHT_ALT              = 0x01;
  const unsigned long       DOM_VK_LEFT_ALT               = 0x02;
  const unsigned long       DOM_VK_LEFT_CONTROL           = 0x03;
  const unsigned long       DOM_VK_RIGHT_CONTROL          = 0x04;
  const unsigned long       DOM_VK_LEFT_SHIFT             = 0x05;
  const unsigned long       DOM_VK_RIGHT_SHIFT            = 0x06;
  const unsigned long       DOM_VK_LEFT_META              = 0x07;
  const unsigned long       DOM_VK_RIGHT_META             = 0x08;
```

```
const unsigned long       DOM_VK_CAPS_LOCK              = 0x09;
const unsigned long       DOM_VK_DELETE                 = 0x0A;
const unsigned long       DOM_VK_END                    = 0x0B;
const unsigned long       DOM_VK_ENTER                  = 0x0C;
const unsigned long       DOM_VK_ESCAPE                 = 0x0D;
const unsigned long       DOM_VK_HOME                   = 0x0E;
const unsigned long       DOM_VK_INSERT                 = 0x0F;
const unsigned long       DOM_VK_NUM_LOCK               = 0x10;
const unsigned long       DOM_VK_PAUSE                  = 0x11;
const unsigned long       DOM_VK_PRINTSCREEN            = 0x12;
const unsigned long       DOM_VK_SCROLL_LOCK            = 0x13;
const unsigned long       DOM_VK_LEFT                   = 0x14;
const unsigned long       DOM_VK_RIGHT                  = 0x15;
const unsigned long       DOM_VK_UP                     = 0x16;
const unsigned long       DOM_VK_DOWN                   = 0x17;
const unsigned long       DOM_VK_PAGE_DOWN              = 0x18;
const unsigned long       DOM_VK_PAGE_UP                = 0x19;
const unsigned long       DOM_VK_F1                     = 0x1A;
const unsigned long       DOM_VK_F2                     = 0x1B;
const unsigned long       DOM_VK_F3                     = 0x1C;
const unsigned long       DOM_VK_F4                     = 0x1D;
const unsigned long       DOM_VK_F5                     = 0x1E;
const unsigned long       DOM_VK_F6                     = 0x1F;
const unsigned long       DOM_VK_F7                     = 0x20;
const unsigned long       DOM_VK_F8                     = 0x21;
const unsigned long       DOM_VK_F9                     = 0x22;
const unsigned long       DOM_VK_F10                    = 0x23;
const unsigned long       DOM_VK_F11                    = 0x24;
const unsigned long       DOM_VK_F12                    = 0x25;
const unsigned long       DOM_VK_F13                    = 0x26;
const unsigned long       DOM_VK_F14                    = 0x27;
const unsigned long       DOM_VK_F15                    = 0x28;
const unsigned long       DOM_VK_F16                    = 0x29;
const unsigned long       DOM_VK_F17                    = 0x2A;
const unsigned long       DOM_VK_F18                    = 0x2B;
const unsigned long       DOM_VK_F19                    = 0x2C;
const unsigned long       DOM_VK_F20                    = 0x2D;
const unsigned long       DOM_VK_F21                    = 0x2E;
const unsigned long       DOM_VK_F22                    = 0x2F;
const unsigned long       DOM_VK_F23                    = 0x30;
const unsigned long       DOM_VK_F24                    = 0x31;

         attribute DOMString       outputString;
         attribute unsigned long   keyVal;
         attribute unsigned long   virtKeyVal;
         attribute boolean         visibleOutputGenerated;
         attribute boolean         numPad;
boolean            checkModifier(in unsigned long modifier);
void               initTextEvent(in DOMString typeArg,
                                 in boolean canBubbleArg,
                                 in boolean cancelableArg,
                                 in views::AbstractView viewArg,
                                 in long detailArg,
                                 in DOMString outputStringArg,
                                 in unsigned long keyValArg,
                                 in unsigned long virtKeyValArg,
                                 in boolean visibleOutputGeneratedArg,
```

```
                                    in boolean numPadArg);
  void               initModifier(in unsigned long modifier,
                                  in boolean value);
};
```

Definition group *VirtualKeyCode*

An integer indicating which key was pressed.

Defined Constants

`DOM_VK_CAPS_LOCK`

`DOM_VK_DELETE`

`DOM_VK_DOWN`

`DOM_VK_END`

`DOM_VK_ENTER`

`DOM_VK_ESCAPE`

`DOM_VK_F1`
: Constant for the F1 function key.

`DOM_VK_F10`
: Constant for the F10 function key.

`DOM_VK_F11`
: Constant for the F11 function key.

`DOM_VK_F12`
: Constant for the F12 function key.

`DOM_VK_F13`
: Constant for the F13 function key.

`DOM_VK_F14`
: Constant for the F14 function key.

`DOM_VK_F15`
: Constant for the F15 function key.

`DOM_VK_F16`
: Constant for the F16 function key.

`DOM_VK_F17`
: Constant for the F17 function key.

`DOM_VK_F18`
: Constant for the F18 function key.

`DOM_VK_F19`
: Constant for the F19 function key.

`DOM_VK_F2`
: Constant for the F2 function key.

`DOM_VK_F20`
: Constant for the F20 function key.

`DOM_VK_F21`
: Constant for the F21 function key.

`DOM_VK_F22`
: Constant for the F22 function key.

`DOM_VK_F23`
: Constant for the F23 function key.

`DOM_VK_F24`
: Constant for the F24 function key.

`DOM_VK_F3`
: Constant for the F3 function key.

`DOM_VK_F4`
: Constant for the F4 function key.

`DOM_VK_F5`
: Constant for the F5 function key.

`DOM_VK_F6`
: Constant for the F6 function key.

`DOM_VK_F7`
: Constant for the F7 function key.

`DOM_VK_F8`
: Constant for the F8 function key.

`DOM_VK_F9`
: Constant for the F9 function key.

`DOM_VK_HOME`

`DOM_VK_INSERT`

`DOM_VK_LEFT`

`DOM_VK_LEFT_ALT`
: This key is a modifier key

`DOM_VK_LEFT_CONTROL`
: This key is a modifier key

`DOM_VK_LEFT_META`
: This key is a modifier key

`DOM_VK_LEFT_SHIFT`
: This key is a modifier key

`DOM_VK_NUM_LOCK`

`DOM_VK_PAGE_DOWN`

`DOM_VK_PAGE_UP`

`DOM_VK_PAUSE`

`DOM_VK_PRINTSCREEN`

`DOM_VK_RIGHT`

`DOM_VK_RIGHT_ALT`
: This key is a modifier key

`DOM_VK_RIGHT_CONTROL`
: This key is a modifier key

`DOM_VK_RIGHT_META`
: This key is a modifier key

`DOM_VK_RIGHT_SHIFT`
: This key is a modifier key

`DOM_VK_SCROLL_LOCK`

`DOM_VK_UNDEFINED`
: Used for key events which do not have a virtual key code available.

DOM_VK_UP

Attributes

keyVal of type unsigned long

The value of keyVal holds the value of the Unicode character associated with the depressed key. If the key has no Unicode representation or no Unicode character is available the value is 0..

numPad of type boolean

The numPad attribute indicates whether or not the key event was generated on the number pad section of the keyboard. If the number pad was used to generate the key event the value is true, otherwise the value is false.

outputString of type DOMString

outputString holds the value of the output generated by the key event. This may be a single Unicode character or it may be a string. It may also be null in the case where no output was generated by the key event.

virtKeyVal of type unsigned long

When the key associated with a key event is not representable via a Unicode character virtKeyVal holds the virtual key code associated with the depressed key. If the key has a Unicode representation or no virtual code is available the value is DOM_VK_UNDEFINED.

visibleOutputGenerated of type boolean

The visibleOutputGenerated attribute indicates whether the key event will normally cause visible output. If the key event does not generate any visible output, such as the use of a function key or the combination of certain modifier keys used in conjunction with another key, then the value will be false. If visible output is normally generated by the key event then the value will be true.

The value of visibleOutputGenerated does not guarantee the creation of a character. If a key event causing visible output is cancelable it may be prevented from causing visible output. This attribute is intended primarily to differentiate between keys events which may or may not produce visible output depending on the system state.

Methods

checkModifier

The checkModifier method is used to check the status of a single modifier key associated with a TextEvent. The identifier of the modifier in question is passed into the checkModifier function. If the modifier is triggered it will return true. If not, it will return false.

The list of keys below represents the allowable modifier paramaters for this method.

- DOM_VK_LEFT_ALT
- DOM_VK_RIGHT_ALT
- DOM_VK_LEFT_CONTROL
- DOM_VK_RIGHT_CONTROL
- DOM_VK_LEFT_SHIFT
- DOM_VK_RIGHT_SHIFT
- DOM_VK_META

Parameters

`modifier` of type `unsigned long`
The modifier which the user wishes to query.

Return Value

`boolean` The status of the modifier represented as a boolean.

No Exceptions

`initModifier`

The `initModifier` method is used to initialize the values of any modifiers associated with a `TextEvent` created through the `DocumentEvent` [p.21] interface. This method may only be called before the `TextEvent` has been dispatched via the dispatchEvent method, though it may be called multiple times during that phase if necessary. If called multiple times with the same `modifier` property the final invocation takes precedence. Unless explicitly give a value of true, all modifiers have a value of false. This method has no effect if called after the event has been dispatched.

The list of keys below represents the allowable modifier paramaters for this method.

- DOM_VK_LEFT_ALT
- DOM_VK_RIGHT_ALT
- DOM_VK_LEFT_CONTROL
- DOM_VK_RIGHT_CONTROL
- DOM_VK_LEFT_SHIFT
- DOM_VK_RIGHT_SHIFT
- DOM_VK_META

Parameters

`modifier` of type `unsigned long`
The modifier which the user wishes to initialize

`value` of type `boolean`
The new value of the modifier.

No Return Value

No Exceptions

`initTextEvent`

The `initTextEvent` method is used to initialize the value of a `TextEvent` created through the `DocumentEvent` [p.21] interface. This method may only be called before the `TextEvent` has been dispatched via the dispatchEvent method, though it may be called multiple times during that phase if necessary. If called multiple times, the final invocation takes precedence. This method has no effect if called after the event has been dispatched.

Parameters

`typeArg` of type `DOMString`
Specifies the event type.

`canBubbleArg` of type `boolean`
Specifies whether or not the event can bubble.

`cancelableArg` of type `boolean`
Specifies whether or not the event's default action can be prevent.

`viewArg` of type `views::AbstractView`
: Specifies the `TextEvent`'s `AbstractView`.

`detailArg` of type `long`
: Specifies the number of repeated keypresses, if available.

`outputStringArg` of type `DOMString`
: Specifies the `TextEvent`'s `outputString` attribute

`keyValArg` of type `unsigned long`
: Specifies the `TextEvent`'s `keyVal`attribute

`virtKeyValArg` of type `unsigned long`
: Specifies the `TextEvent`'s `virtKeyVal`attribute

`visibleOutputGeneratedArg` of type `boolean`
: Specifies the `TextEvent`'s `visibleOutputGenerated`attribute

`numPadArg` of type `boolean`
: Specifies the `TextEvent`'s `numPad`attribute

No Return Value

No Exceptions

There are two major groups of key events. The first contains the `textInput` event. The `textInput` event indicates that text information has been entered, either in the form of printable characters or non-printable text information such as modifier keys. `textInput` events are not necessarily accompanied by the events of the second major groups of key events, `keydown` and `keyup`.

textInput

The textInput event indicates that text information has been entered. The text information entered can originate from a variety of sources. It could, for example, be a character resulting from a keypress. It could also be a string resulting from an input method.

The `detail` attribute inherited from `UIEvent` [p.23] is used to indicated the number of keypresses which have occurred during key repetition. If this information is not available this value should be 0.

- Bubbles: Yes
- Cancelable: Yes
- Context Info: view, detail, visibleOutputGenerated, outputString, keyVal, virtKeyVal, numPad.

The `keydown` and `keyup` events comprise the second group of key events. These events are fired to indicate the physical motion of the keys on the character generation device. Depending on the input system being used, `textEvent` events may or may not be generated for each pair of `keydown` and `keyup` events.

keydown

The keydown event occurs when a key is pressed down.

- Bubbles: Yes
- Cancelable: Yes
- Context Info: view, keyVal, virtKeyVal, numPad.

keyup

The keyup event occurs when a key is released.

- Bubbles: Yes
- Cancelable: Yes

- Context Info: view, keyVal, virtKeyVal, numPad.

1.5.4. Mutation event types

The mutation event module is designed to allow notification of any changes to the structure of a document, including attr and text modifications. It may be noted that none of the mutation events listed 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 related 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 fired. 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.

A DOM application may use the `hasFeature(feature, version)` method of the `DOMImplementation` interface with parameter values "MutationEvents" and "3.0" (respectively) to determine whether or not the Mutation event module is supported by the implementation. In order to fully support this module, an implementation must also support the "Events" feature defined in this specification. Please, refer to additional information about *conformance* in the DOM Level 3 Core specification [DOM Level 3 Core]. The DOM Level 3 Mutation Events module is backward compatible with the DOM Level 2 Mutation Events [DOM Level 2 Events] module, i.e. a DOM Level 3 Mutation Events implementation who returns `true` for "MutationEvents" with the `version` number `"3.0"` must also return `true` for this `feature` when the `version` number is `"2.0"`, `""` or, `null`.

Note: To create an instance of the `MutationEvent` [p.36] interface, use the feature string "MutationEvents" as the value of the input parameter used with the `createEvent` method of the `DocumentEvent` [p.21] interface.

Interface *MutationEvent* (introduced in **DOM Level 2**)

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

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);
};
```

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`
: The `initMutationEvent` method is used to initialize the value of a `MutationEvent` created through the `DocumentEvent` [p.21] interface. This method may only be called before the `MutationEvent` has been dispatched via the `dispatchEvent` method, though it may be called multiple times during that phase if necessary. If called multiple times, the final invocation takes precedence.

  Parameters

`typeArg` of type `DOMString`
: Specifies the event type.

`canBubbleArg` of type `boolean`
: Specifies whether or not the event can bubble.

`cancelableArg` of type `boolean`
: Specifies whether or not the event's default action can be prevented.

`relatedNodeArg` of type `Node`
: Specifies the `Event` [p.18] 's related Node.

`prevValueArg` of type `DOMString`
: Specifies the `Event` [p.18] 's `prevValue` attribute. This value may be null.

`newValueArg` of type `DOMString`
: Specifies the `Event` [p.18] 's `newValue` attribute. This value may be null.

`attrNameArg` of type `DOMString`
: Specifies the `Event` [p.18] 's `attrName` attribute. This value may be null.

`attrChangeArg` of type `unsigned short`
: Specifies the `Event` [p.18] 's `attrChange` attribute

No Return Value

No Exceptions

The different types of Mutation events that can occur are:

DOMSubtreeModified
: This is a general event for notification of all changes to the document. It can be used instead of the more specific events listed below. It may be fired 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 accomodate 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 have fired.
  - Bubbles: Yes
  - Cancelable: No
  - Context Info: None

DOMNodeInserted
: Fired when a node has been added as a *child* [p.67] of another node. This event is dispatched after the insertion has taken place. The target of this event is the node being inserted.
  - Bubbles: Yes
  - Cancelable: No
  - Context Info: relatedNode holds the parent node

DOMNodeRemoved
: Fired when a node is being removed from its parent node. This event is dispatched before the node is removed from the tree. The target of this event is the node being removed.
  - Bubbles: Yes
  - Cancelable: No
  - Context Info: relatedNode holds the parent node

DOMNodeRemovedFromDocument
: Fired when 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. This event is dispatched before the removal takes place.

The target of this event is the Node being removed. If the Node is being directly removed the DOMNodeRemoved event will fire before the DOMNodeRemovedFromDocument event.

- Bubbles: No
- Cancelable: No
- Context Info: None

DOMNodeInsertedIntoDocument

Fired when a node is being inserted into a document, either through direct insertion of the Node or insertion of a subtree in which it is contained. This event is dispatched after the insertion has taken place. The target of this event is the node being inserted. If the Node is being directly inserted the DOMNodeInserted event will fire before the DOMNodeInsertedIntoDocument event.

- Bubbles: No
- Cancelable: No
- Context Info: None

DOMAttrModified

Fired after an `Attr` has been modified on a node. The target of this event is the `Node` whose `Attr` changed. The value of attrChange indicates whether the `Attr` was modified, added, or removed. The value of relatedNode indicates the `Attr` node whose value has been affected. It is expected that string based replacement of an `Attr` value will be viewed as a modification of the `Attr` since its identity does not change. Subsequently replacement of the `Attr` node with a different `Attr` node is viewed as the removal of the first `Attr` node and the addition of the second.

- Bubbles: Yes
- Cancelable: No
- Context Info: attrName, attrChange, prevValue, newValue, relatedNode

DOMCharacterDataModified

Fired after CharacterData within a node has been modified but the node itself has not been inserted or deleted. This event is also triggered by modifications to PI elements. The target of this event is the CharacterData node.

- Bubbles: Yes
- Cancelable: No
- Context Info: prevValue, newValue

1.5.5. HTML event types

The HTML event module is composed of events listed in HTML 4.0 and additional events which are supported in *DOM Level 0* [p.67] browsers.

A DOM application may use the `hasFeature(feature, version)` method of the `DOMImplementation` interface with parameter values "HTMLEvents" and "3.0" (respectively) to determine whether or not the HTML event module is supported by the implementation. In order to fully support this module, an implementation must also support the "Events" feature defined in this specification. Please, refer to additional information about *conformance* in the DOM Level 3 Core specification [DOM Level 3 Core]. The DOM Level 3 HTML Events module is backward compatible with the DOM Level 2 HTML Events [DOM Level 2 Events] module, i.e. a DOM Level 3 HTML Events implementation who returns `true` for "HTMLEvents" with the `version` number `"3.0"` must also return `true` for this `feature` when the `version` number is `"2.0"`, `""` or, `null`.

Note: To create an instance of the `Event` [p.18] interface for the HTML event module, use the feature string "HTMLEvents" as the value of the input parameter used with the `createEvent` method of the `DocumentEvent` [p.21] interface.

The HTML events use the base DOM Event interface to pass contextual information.

The different types of such events that can occur are:

load
: The load event occurs when the DOM implementation finishes loading all content within the BODY element, all frames within a FRAMESET, or an OBJECT element.
  - Bubbles: No
  - Cancelable: No
  - Context Info: None

unload
: The unload event occurs when the DOM implementation removes a document from a window or frame. This event is valid for BODY and FRAMESET elements.
  - Bubbles: No
  - Cancelable: No
  - Context Info: None

abort
: The abort event occurs when page loading is stopped before an image has been allowed to completely load. This event applies to OBJECT elements.
  - Bubbles: Yes
  - Cancelable: No
  - Context Info: None

error
: The error event occurs when an image does not load properly or when an error occurs during script execution. This event is valid for OBJECT elements, BODY elements, and FRAMESET element.
  - Bubbles: Yes
  - Cancelable: No
  - Context Info: None

select
: The select event occurs when a user selects some text in a text field. This event is valid for INPUT and TEXTAREA elements.
  - Bubbles: Yes
  - Cancelable: No
  - Context Info: None

change
: The change event occurs when a control loses the input focus and its value has been modified since gaining focus. This event is valid for INPUT, SELECT, and TEXTAREA. element.
  - Bubbles: Yes
  - Cancelable: No
  - Context Info: None

submit
The submit event occurs when a form is submitted. This event only applies to the FORM element.
- Bubbles: Yes
- Cancelable: Yes
- Context Info: None

reset
The reset event occurs when a form is reset. This event only applies to the FORM element.
- Bubbles: Yes
- Cancelable: No
- Context Info: None

focus
The focus event occurs when an element receives focus either via a pointing device or by tabbing navigation. This event is valid for the following elements: A, AREA, LABEL, INPUT, SELECT, TEXTAREA, and BUTTON.
- Bubbles: No
- Cancelable: No
- Context Info: None

blur
The blur event occurs when an element loses focus either via the pointing device or by tabbing navigation. This event is valid for the following elements: A, AREA, LABEL, INPUT, SELECT, TEXTAREA, and BUTTON.
- Bubbles: No
- Cancelable: No
- Context Info: None

resize
The resize event occurs when a document view is resized.
- Bubbles: Yes
- Cancelable: No
- Context Info: None

scroll
The scroll event occurs when a document view is scrolled.
- Bubbles: Yes
- Cancelable: No
- Context Info: None

1.6. Issues

Issue getModifier:
Why is modifier state exposed through a method rather than an attribute?
Resolution: The modifier keys are not currently representable as bit flags. Setting them individually would therefore require an attribute for each. Rather than bloat the api, especially given the addition of left and right modifier keys, the modifiers are exposed via a single method.

Issue ISO-IEC-9995:
Have you coordinated this set with that defined by ISO/IEC 9995 which addresses various Keyboard symbol issues.

Resolution: Upon examination of the ISO spec we found it to be insufficient to our needs. It does not represent the left/right differentiation between some keys. It also lacks function keys.

Issue ISO-IEC-14755:

Review ISO/IEC 14755 "Input methods to enter characters from the repertoire of ISO/IEC 10646 with a keyboard or other input device" to insure that the treatment of input state is consistent with that expected by current practice when it comes to platforms which support input methods.

Issue offsets:

(This issue is related with mouse events and Views?)
it would be useful if MouseEvent class had a property that would enable listners to learn about coordinates of the event within the element's own coordinate system.
Resolution: We are not doing views at the momewnt.

Issue unicodeidents:

Some of the unicode chars are pretty esoteric (i.e. home, end, scroll lock). Do we want to adopt these or will this be harder on users than defining them in the DOM Event Spec. About a dozen keys fit this pattern.
Resolution: There are use cases to keep them.

Issue texteventwithoutchargeneration:

The results of the discussions on switching the keypress event out for the textEvent were inconclusive on the question of whether to fire textEvents for non character generating keys input. This includes modifier keys, function keys, etc.
Resolution: There are use cases to keep them.

Issue public0198-1:

From 0198, being able to manipulate the "default" group using EventGroup functions?
Resolution: Non-groups methods can always be used to access the default group. We didn't find a good reason to have access to the default group through the groups methods for the moment.

Issue public0198-2:

From 0198, no dispatchEvent for a specific group?
Resolution: The default and correct behavior is to dispatch the event on all listeners, independently from the groups. We think we should stick with this model without further reasons.

Issue public0198-3:

From 0198, assume that dispatchEvent's return value would be false if any listener in any group called preventDefault?
Resolution: Yes. (no change in the spec)

Issue public0198-4:

From 0198, you could potentially add the same listener to the same event target on multiple groups. Should the Event or EventListener interface be extended so that you could detect which group is active?
Resolution: We didn't find good use cases to do so for the moment.

Issue public0279-1:

From 0279, event types and case sensitivity.
Resolution: `Event.type` [p.19] returns event names in lowercase.

Issue public0283-1:

From 0283, an implementation can support the MutationEvents module even if it never fires anything more specific than DOMSubtreeModified events. Furthermore, although it is not recommended, an implementation may legally choose to fire an event of this type only after every 100 modifications to the document, or every 30 seconds if any changes have happened during that interval.

Resolution: Yes, the spec doesn't prevent to do so. (no change in the spec)

Issue public0294-1:

From 0294, handleEvent should be allowed to throw exceptions given that the dispatch is ignoring them.

Issue public0294-2:

From 0294, if a user provides his own implementation of Event instead of using createEvent, how does the DOM implementation do to set the currentTarget or phase? (see also 0296)

Resolution: A new CustomEvent interface was added in the draft.

Issue public0294-3:

From 0294, Name collisions between EventTarget.addEventListener() and EventTargetGroup.addEventListener() and removeEventListener*s()

Resolution: Fixed.

Issue public0294-4:

From 0294, method of creating generic Event (i.e. createEvent("Events")) in case you want to use Event dispatching, but don't care if the implementation supports any other feature. createEvents("Events") could return whatever implementation that was most convienient for it. For example, an HTML implementation could return an object that coincidentally supported HTMLEvent.

Resolution: Yes. we don't prevent that. (no change in the spec)

Issue public0294-5:

From 0294, merge the Event groups with the existing interfaces, don't create new ones.

Resolution: Done.

Issue public0295-1:

From 0295, should we rename the event "textEvent" to "text"?

Resolution: "textInput"

Issue public0295-2:

From 0295, include a note that explains the interaction between addEventListener(), removeEventListener() and eventListenerList.item(). Does removeEventListener() preserve the order of the registered event listeners that are not removed? Are EventListenerList live?

Resolution: Fixed: unordered and not lived.

Issue public0296-1:

From 0296, hasFeature("MouseEvents", "3.0") returns true and similar passages imply that level 2 implementations don't support the corresponding event modules, since they would return false to hasFeature(modulename,"3.0"). They do support it, just not at the L3 version. An L3 version of an L2 introduced module, should return true if the version is null, "", "2.0" or "3.0".

Resolution: Fixed.

Issue public0296-2:

From 0296, It might be useful to cause a call to init*Event() after dispatch has started to raise an exception. Dispatching an event a second time should also raise an exception.

Resolution: waiting for a use case and clarification.

Issue public0301:

From 0301, the capture phase can't be disabled. text is misleading.

Resolution: clarified.

Appendix A: Changes

Editor:
Philippe Le Hégaret, W3C

A.1: Changes between DOM Level 2 Events and DOM Level 3 Events

A.1.1: Changes to DOM Level 2 Events interfaces

(***ED:*** This page needs update...)

Interface `EventTarget` [p.12]
The `Event` [p.18] interface has one new attribute: `eventListenerList`.

A.1.2: New Interfaces

The interfaces `EventListenerList`, `EventGroup` [p.16] , `EventTargetGroup`, `DocumentEventGroup`, and `TextEvent` [p.29] were added to the Events module.

A.1.2: New Interfaces

Appendix B: IDL Definitions

This appendix contains the complete OMG IDL [OMGIDL] for the Level 3 Document Object Model Events definitions.

The IDL files are also available as:
http://www.w3.org/TR/2002/WD-DOM-Level-3-Events-20020208/idl.zip

events.idl:

```
// File: events.idl

#ifndef _EVENTS_IDL_
#define _EVENTS_IDL_

#include "dom.idl"
#include "views.idl"

#pragma prefix "dom.w3c.org"
module events
{

  typedef dom::DOMString DOMString;
  typedef dom::DOMTimeStamp DOMTimeStamp;
  typedef dom::Node Node;

  interface EventListener;
  interface Event;
  interface EventGroup;

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


  // 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);
    boolean            dispatchEvent(in Event evt)
                                        raises(EventException);
    // Introduced in DOM Level 3:
    void               addGroupedEventListener(in DOMString type,
                                               in EventListener listener,
                                               in boolean useCapture,
                                               in EventGroup evtGroup);
    // Introduced in DOM Level 3:
```

```
  void               removeGroupedEventListener(in DOMString type,
                                                in EventListener listener,
                                                in boolean useCapture,
                                                in EventGroup evtGroup);
  // Introduced in DOM Level 3:
  boolean            canTrigger(in DOMString type);
  // Introduced in DOM Level 3:
  boolean            isRegisteredHere(in DOMString type);
};

// Introduced in DOM Level 3:
interface EventGroup {
  boolean            isSameEventGroup(in EventGroup other);
};

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

// 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 2:
interface DocumentEvent {
  Event              createEvent(in DOMString eventType)
                                 raises(dom::DOMException);
  EventGroup         createEventGroup();
};

// Introduced in DOM Level 3:
interface CustomEvent : Event {
  void               setCurrentTarget(in Node target);
  void               setEventPhase(in unsigned short phase);
};

// 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 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:
interface TextEvent : UIEvent {

  // VirtualKeyCode
  const unsigned long       DOM_VK_UNDEFINED              = 0x0;
  const unsigned long       DOM_VK_RIGHT_ALT              = 0x01;
  const unsigned long       DOM_VK_LEFT_ALT               = 0x02;
  const unsigned long       DOM_VK_LEFT_CONTROL           = 0x03;
  const unsigned long       DOM_VK_RIGHT_CONTROL          = 0x04;
  const unsigned long       DOM_VK_LEFT_SHIFT             = 0x05;
  const unsigned long       DOM_VK_RIGHT_SHIFT            = 0x06;
  const unsigned long       DOM_VK_LEFT_META              = 0x07;
  const unsigned long       DOM_VK_RIGHT_META             = 0x08;
  const unsigned long       DOM_VK_CAPS_LOCK              = 0x09;
  const unsigned long       DOM_VK_DELETE                 = 0x0A;
  const unsigned long       DOM_VK_END                    = 0x0B;
  const unsigned long       DOM_VK_ENTER                  = 0x0C;
  const unsigned long       DOM_VK_ESCAPE                 = 0x0D;
  const unsigned long       DOM_VK_HOME                   = 0x0E;
```

```
  const unsigned long       DOM_VK_INSERT                 = 0x0F;
  const unsigned long       DOM_VK_NUM_LOCK               = 0x10;
  const unsigned long       DOM_VK_PAUSE                  = 0x11;
  const unsigned long       DOM_VK_PRINTSCREEN            = 0x12;
  const unsigned long       DOM_VK_SCROLL_LOCK            = 0x13;
  const unsigned long       DOM_VK_LEFT                   = 0x14;
  const unsigned long       DOM_VK_RIGHT                  = 0x15;
  const unsigned long       DOM_VK_UP                     = 0x16;
  const unsigned long       DOM_VK_DOWN                   = 0x17;
  const unsigned long       DOM_VK_PAGE_DOWN              = 0x18;
  const unsigned long       DOM_VK_PAGE_UP                = 0x19;
  const unsigned long       DOM_VK_F1                     = 0x1A;
  const unsigned long       DOM_VK_F2                     = 0x1B;
  const unsigned long       DOM_VK_F3                     = 0x1C;
  const unsigned long       DOM_VK_F4                     = 0x1D;
  const unsigned long       DOM_VK_F5                     = 0x1E;
  const unsigned long       DOM_VK_F6                     = 0x1F;
  const unsigned long       DOM_VK_F7                     = 0x20;
  const unsigned long       DOM_VK_F8                     = 0x21;
  const unsigned long       DOM_VK_F9                     = 0x22;
  const unsigned long       DOM_VK_F10                    = 0x23;
  const unsigned long       DOM_VK_F11                    = 0x24;
  const unsigned long       DOM_VK_F12                    = 0x25;
  const unsigned long       DOM_VK_F13                    = 0x26;
  const unsigned long       DOM_VK_F14                    = 0x27;
  const unsigned long       DOM_VK_F15                    = 0x28;
  const unsigned long       DOM_VK_F16                    = 0x29;
  const unsigned long       DOM_VK_F17                    = 0x2A;
  const unsigned long       DOM_VK_F18                    = 0x2B;
  const unsigned long       DOM_VK_F19                    = 0x2C;
  const unsigned long       DOM_VK_F20                    = 0x2D;
  const unsigned long       DOM_VK_F21                    = 0x2E;
  const unsigned long       DOM_VK_F22                    = 0x2F;
  const unsigned long       DOM_VK_F23                    = 0x30;
  const unsigned long       DOM_VK_F24                    = 0x31;

           attribute DOMString       outputString;
           attribute unsigned long   keyVal;
           attribute unsigned long   virtKeyVal;
           attribute boolean         visibleOutputGenerated;
           attribute boolean         numPad;
  boolean             checkModifier(in unsigned long modifier);
  void                initTextEvent(in DOMString typeArg,
                                    in boolean canBubbleArg,
                                    in boolean cancelableArg,
                                    in views::AbstractView viewArg,
                                    in long detailArg,
                                    in DOMString outputStringArg,
                                    in unsigned long keyValArg,
                                    in unsigned long virtKeyValArg,
                                    in boolean visibleOutputGeneratedArg,
                                    in boolean numPadArg);
  void                initModifier(in unsigned long modifier,
                                   in boolean value);
};

// 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);
  };
};

#endif // _EVENTS_IDL_
```

events.idl:

Appendix C: Java Language Binding

This appendix contains the complete Java [Java] bindings for the Level 3 Document Object Model Events.

The Java files are also available as
http://www.w3.org/TR/2002/WD-DOM-Level-3-Events-20020208/java-binding.zip

org/w3c/dom/events/EventException.java:

```
package org.w3c.dom.events;

public class EventException extends RuntimeException {
    public EventException(short code, String message) {
       super(message);
       this.code = code;
    }
    public short   code;
    // EventExceptionCode
    public static final short UNSPECIFIED_EVENT_TYPE_ERR = 0;

}
```

org/w3c/dom/events/EventTarget.java:

```
package org.w3c.dom.events;

public interface EventTarget {
    public void addEventListener(String type,
                                 EventListener listener,
                                 boolean useCapture);

    public void removeEventListener(String type,
                                    EventListener listener,
                                    boolean useCapture);

    public boolean dispatchEvent(Event evt)
                                 throws EventException;

    public void addGroupedEventListener(String type,
                                        EventListener listener,
                                        boolean useCapture,
                                        EventGroup evtGroup);

    public void removeGroupedEventListener(String type,
                                           EventListener listener,
                                           boolean useCapture,
                                           EventGroup evtGroup);

    public boolean canTrigger(String type);

    public boolean isRegisteredHere(String type);

}
```

org/w3c/dom/events/EventGroup.java:

```
package org.w3c.dom.events;

public interface EventGroup {
    public boolean isSameEventGroup(EventGroup other);

}
```

org/w3c/dom/events/EventListener.java:

```
package org.w3c.dom.events;

public interface EventListener {
    public void handleEvent(Event evt);

}
```

org/w3c/dom/events/Event.java:

```
package org.w3c.dom.events;

public interface Event {
    // PhaseType
    public static final short CAPTURING_PHASE           = 1;
    public static final short AT_TARGET                 = 2;
    public static final short BUBBLING_PHASE            = 3;

    public String getType();

    public EventTarget getTarget();

    public EventTarget getCurrentTarget();

    public short getEventPhase();

    public boolean getBubbles();

    public boolean getCancelable();

    public long getTimeStamp();

    public void stopPropagation();

    public void preventDefault();

    public void initEvent(String eventTypeArg,
                          boolean canBubbleArg,
                          boolean cancelableArg);

}
```

org/w3c/dom/events/DocumentEvent.java:

```
package org.w3c.dom.events;

import org.w3c.dom.DOMException;

public interface DocumentEvent {
    public Event createEvent(String eventType)
                             throws DOMException;

    public EventGroup createEventGroup();

}
```

org/w3c/dom/events/CustomEvent.java:

```
package org.w3c.dom.events;

import org.w3c.dom.Node;

public interface CustomEvent extends Event {
    public void setCurrentTarget(Node target);

    public void setEventPhase(short phase);

}
```

org/w3c/dom/events/UIEvent.java:

```
package org.w3c.dom.events;

import org.w3c.dom.views.AbstractView;

public interface UIEvent extends Event {
    public AbstractView getView();

    public int getDetail();

    public void initUIEvent(String typeArg,
                            boolean canBubbleArg,
                            boolean cancelableArg,
                            AbstractView viewArg,
                            int detailArg);

}
```

org/w3c/dom/events/MouseEvent.java:

```
package org.w3c.dom.events;

import org.w3c.dom.views.AbstractView;

public interface MouseEvent extends UIEvent {
    public int getScreenX();
```

```
    public int getScreenY();

    public int getClientX();

    public int getClientY();

    public boolean getCtrlKey();

    public boolean getShiftKey();

    public boolean getAltKey();

    public boolean getMetaKey();

    public short getButton();

    public EventTarget getRelatedTarget();

    public void initMouseEvent(String typeArg,
                               boolean canBubbleArg,
                               boolean cancelableArg,
                               AbstractView viewArg,
                               int detailArg,
                               int screenXArg,
                               int screenYArg,
                               int clientXArg,
                               int clientYArg,
                               boolean ctrlKeyArg,
                               boolean altKeyArg,
                               boolean shiftKeyArg,
                               boolean metaKeyArg,
                               short buttonArg,
                               EventTarget relatedTargetArg);

}
```

org/w3c/dom/events/TextEvent.java:

```
package org.w3c.dom.events;

import org.w3c.dom.views.AbstractView;

public interface TextEvent extends UIEvent {
    // VirtualKeyCode
    public static final int DOM_VK_UNDEFINED          = 0x0;
    public static final int DOM_VK_RIGHT_ALT          = 0x01;
    public static final int DOM_VK_LEFT_ALT           = 0x02;
    public static final int DOM_VK_LEFT_CONTROL       = 0x03;
    public static final int DOM_VK_RIGHT_CONTROL      = 0x04;
    public static final int DOM_VK_LEFT_SHIFT         = 0x05;
    public static final int DOM_VK_RIGHT_SHIFT        = 0x06;
    public static final int DOM_VK_LEFT_META          = 0x07;
    public static final int DOM_VK_RIGHT_META         = 0x08;
    public static final int DOM_VK_CAPS_LOCK          = 0x09;
    public static final int DOM_VK_DELETE             = 0x0A;
    public static final int DOM_VK_END                = 0x0B;
```

```
public static final int DOM_VK_ENTER              = 0x0C;
public static final int DOM_VK_ESCAPE             = 0x0D;
public static final int DOM_VK_HOME               = 0x0E;
public static final int DOM_VK_INSERT             = 0x0F;
public static final int DOM_VK_NUM_LOCK           = 0x10;
public static final int DOM_VK_PAUSE              = 0x11;
public static final int DOM_VK_PRINTSCREEN        = 0x12;
public static final int DOM_VK_SCROLL_LOCK        = 0x13;
public static final int DOM_VK_LEFT               = 0x14;
public static final int DOM_VK_RIGHT              = 0x15;
public static final int DOM_VK_UP                 = 0x16;
public static final int DOM_VK_DOWN               = 0x17;
public static final int DOM_VK_PAGE_DOWN          = 0x18;
public static final int DOM_VK_PAGE_UP            = 0x19;
public static final int DOM_VK_F1                 = 0x1A;
public static final int DOM_VK_F2                 = 0x1B;
public static final int DOM_VK_F3                 = 0x1C;
public static final int DOM_VK_F4                 = 0x1D;
public static final int DOM_VK_F5                 = 0x1E;
public static final int DOM_VK_F6                 = 0x1F;
public static final int DOM_VK_F7                 = 0x20;
public static final int DOM_VK_F8                 = 0x21;
public static final int DOM_VK_F9                 = 0x22;
public static final int DOM_VK_F10                = 0x23;
public static final int DOM_VK_F11                = 0x24;
public static final int DOM_VK_F12                = 0x25;
public static final int DOM_VK_F13                = 0x26;
public static final int DOM_VK_F14                = 0x27;
public static final int DOM_VK_F15                = 0x28;
public static final int DOM_VK_F16                = 0x29;
public static final int DOM_VK_F17                = 0x2A;
public static final int DOM_VK_F18                = 0x2B;
public static final int DOM_VK_F19                = 0x2C;
public static final int DOM_VK_F20                = 0x2D;
public static final int DOM_VK_F21                = 0x2E;
public static final int DOM_VK_F22                = 0x2F;
public static final int DOM_VK_F23                = 0x30;
public static final int DOM_VK_F24                = 0x31;

public String getOutputString();
public void setOutputString(String outputString);

public int getKeyVal();
public void setKeyVal(int keyVal);

public int getVirtKeyVal();
public void setVirtKeyVal(int virtKeyVal);

public boolean getVisibleOutputGenerated();
public void setVisibleOutputGenerated(boolean visibleOutputGenerated);

public boolean getNumPad();
public void setNumPad(boolean numPad);

public boolean checkModifier(int modifier);

public void initTextEvent(String typeArg,
```

```
                                boolean canBubbleArg,
                                boolean cancelableArg,
                                AbstractView viewArg,
                                int detailArg,
                                String outputStringArg,
                                int keyValArg,
                                int virtKeyValArg,
                                boolean visibleOutputGeneratedArg,
                                boolean numPadArg);

    public void initModifier(int modifier,
                             boolean value);

}
```

org/w3c/dom/events/MutationEvent.java:

```
package org.w3c.dom.events;

import org.w3c.dom.Node;

public interface MutationEvent extends Event {
    // attrChangeType
    public static final short MODIFICATION             = 1;
    public static final short ADDITION                 = 2;
    public static final short REMOVAL                  = 3;

    public Node getRelatedNode();

    public String getPrevValue();

    public String getNewValue();

    public String getAttrName();

    public short getAttrChange();

    public void initMutationEvent(String typeArg,
                                  boolean canBubbleArg,
                                  boolean cancelableArg,
                                  Node relatedNodeArg,
                                  String prevValueArg,
                                  String newValueArg,
                                  String attrNameArg,
                                  short attrChangeArg);

}
```

Appendix D: ECMAScript Language Binding

This appendix contains the complete ECMAScript [ECMAScript] binding for the Level 3 Document Object Model Events definitions.

Objects that implement the **EventTarget** interface:

Functions of objects that implement the **EventTarget** interface:

addEventListener(type, listener, useCapture)

This function has no return value.
The **type** parameter is a **String**.
The **listener** parameter is an object that implements the **EventListener** interface.
The **useCapture** parameter is a **Boolean**.

removeEventListener(type, listener, useCapture)

This function has no return value.
The **type** parameter is a **String**.
The **listener** parameter is an object that implements the **EventListener** interface.
The **useCapture** parameter is a **Boolean**.

dispatchEvent(evt)

This function returns a **Boolean**.
The **evt** parameter is an object that implements the **Event** interface.
This function can raise an object that implements the **EventException** interface.

addGroupedEventListener(type, listener, useCapture, evtGroup)

This function has no return value.
The **type** parameter is a **String**.
The **listener** parameter is an object that implements the **EventListener** interface.
The **useCapture** parameter is a **Boolean**.
The **evtGroup** parameter is an object that implements the **EventGroup** interface.

removeGroupedEventListener(type, listener, useCapture, evtGroup)

This function has no return value.
The **type** parameter is a **String**.
The **listener** parameter is an object that implements the **EventListener** interface.
The **useCapture** parameter is a **Boolean**.
The **evtGroup** parameter is an object that implements the **EventGroup** interface.

canTrigger(type)

This function returns a **Boolean**.
The **type** parameter is a **String**.

isRegisteredHere(type)

This function returns a **Boolean**.
The **type** parameter is a **String**.

Objects that implement the **EventGroup** interface:

Functions of objects that implement the **EventGroup** interface:

isSameEventGroup(other)

This function returns a **Boolean**.
The **other** parameter is an object that implements the **EventGroup** interface.

EventListener function:

This function has no return value. The parameter is an object that implements the **Event** interface.

Properties of the **Event** Constructor function:

Event.CAPTURING_PHASE

The value of the constant **Event.CAPTURING_PHASE** is **1**.

Event.AT_TARGET

The value of the constant **Event.AT_TARGET** is **2**.

Event.BUBBLING_PHASE

The value of the constant **Event.BUBBLING_PHASE** is **3**.

Objects that implement the **Event** interface:

Properties of objects that implement the **Event** interface:

type

This read-only property is a **String**.

target

This read-only property is an object that implements the **EventTarget** interface.

currentTarget

This read-only property is an object that implements the **EventTarget** interface.

eventPhase

This read-only property is a **Number**.

bubbles

This read-only property is a **Boolean**.

cancelable

This read-only property is a **Boolean**.

timeStamp

This read-only property is an object that implements the **Date** interface.

Functions of objects that implement the **Event** interface:

stopPropagation()

This function has no return value.

preventDefault()

This function has no return value.

initEvent(eventTypeArg, canBubbleArg, cancelableArg)

This function has no return value.

The **eventTypeArg** parameter is a **String**.

The **canBubbleArg** parameter is a **Boolean**.

The **cancelableArg** parameter is a **Boolean**.

Properties of the **EventException** Constructor function:

EventException.UNSPECIFIED_EVENT_TYPE_ERR

The value of the constant **EventException.UNSPECIFIED_EVENT_TYPE_ERR** is **0**.

Objects that implement the **EventException** interface:

Properties of objects that implement the **EventException** interface:

code

This property is a **Number**.

Objects that implement the **DocumentEvent** interface:

Functions of objects that implement the **DocumentEvent** interface:

createEvent(eventType)

This function returns an object that implements the **Event** interface.

The **eventType** parameter is a **String**.

This function can raise an object that implements the **DOMException** interface.

createEventGroup()

This function returns an object that implements the **EventGroup** interface.

Objects that implement the **CustomEvent** interface:

Objects that implement the **CustomEvent** interface have all properties and functions of the **Event** interface as well as the properties and functions defined below.

Functions of objects that implement the **CustomEvent** interface:

setCurrentTarget(target)

This function has no return value.

The **target** parameter is an object that implements the **Node** interface.

setEventPhase(phase)

This function has no return value.

The **phase** parameter is a **Number**.

Objects that implement the **UIEvent** interface:

Objects that implement the **UIEvent** interface have all properties and functions of the **Event** interface as well as the properties and functions defined below.

Properties of objects that implement the **UIEvent** interface:

view

This read-only property is an object that implements the **AbstractView** interface.

detail

This read-only property is a **Number**.

Functions of objects that implement the **UIEvent** interface:

initUIEvent(typeArg, canBubbleArg, cancelableArg, viewArg, detailArg)

This function has no return value.

The **typeArg** parameter is a **String**.

The **canBubbleArg** parameter is a **Boolean**.

The **cancelableArg** parameter is a **Boolean**.

The **viewArg** parameter is an object that implements the **AbstractView** interface.

The **detailArg** parameter is a **Number**.

Objects that implement the **MouseEvent** interface:

Objects that implement the **MouseEvent** interface have all properties and functions of the **UIEvent** interface as well as the properties and functions defined below.

Properties of objects that implement the **MouseEvent** interface:

screenX

This read-only property is a **Number**.

screenY

This read-only property is a **Number**.

clientX

This read-only property is a **Number**.

clientY

This read-only property is a **Number**.

ctrlKey

This read-only property is a **Boolean**.

shiftKey

This read-only property is a **Boolean**.

altKey
This read-only property is a **Boolean**.
metaKey
This read-only property is a **Boolean**.
button
This read-only property is a **Number**.
relatedTarget
This read-only property is an object that implements the **EventTarget** interface.

Functions of objects that implement the **MouseEvent** interface:

initMouseEvent(typeArg, canBubbleArg, cancelableArg, viewArg, detailArg, screenXArg, screenYArg, clientXArg, clientYArg, ctrlKeyArg, altKeyArg, shiftKeyArg, metaKeyArg, buttonArg, relatedTargetArg)
This function has no return value.
The **typeArg** parameter is a **String**.
The **canBubbleArg** parameter is a **Boolean**.
The **cancelableArg** parameter is a **Boolean**.
The **viewArg** parameter is an object that implements the **AbstractView** interface.
The **detailArg** parameter is a **Number**.
The **screenXArg** parameter is a **Number**.
The **screenYArg** parameter is a **Number**.
The **clientXArg** parameter is a **Number**.
The **clientYArg** parameter is a **Number**.
The **ctrlKeyArg** parameter is a **Boolean**.
The **altKeyArg** parameter is a **Boolean**.
The **shiftKeyArg** parameter is a **Boolean**.
The **metaKeyArg** parameter is a **Boolean**.
The **buttonArg** parameter is a **Number**.
The **relatedTargetArg** parameter is an object that implements the **EventTarget** interface.

Properties of the **TextEvent** Constructor function:

TextEvent.DOM_VK_UNDEFINED
The value of the constant **TextEvent.DOM_VK_UNDEFINED** is **0x0**.
TextEvent.DOM_VK_RIGHT_ALT
The value of the constant **TextEvent.DOM_VK_RIGHT_ALT** is **0x01**.
TextEvent.DOM_VK_LEFT_ALT
The value of the constant **TextEvent.DOM_VK_LEFT_ALT** is **0x02**.
TextEvent.DOM_VK_LEFT_CONTROL
The value of the constant **TextEvent.DOM_VK_LEFT_CONTROL** is **0x03**.
TextEvent.DOM_VK_RIGHT_CONTROL
The value of the constant **TextEvent.DOM_VK_RIGHT_CONTROL** is **0x04**.
TextEvent.DOM_VK_LEFT_SHIFT
The value of the constant **TextEvent.DOM_VK_LEFT_SHIFT** is **0x05**.
TextEvent.DOM_VK_RIGHT_SHIFT
The value of the constant **TextEvent.DOM_VK_RIGHT_SHIFT** is **0x06**.
TextEvent.DOM_VK_LEFT_META
The value of the constant **TextEvent.DOM_VK_LEFT_META** is **0x07**.

TextEvent.DOM_VK_RIGHT_META
The value of the constant **TextEvent.DOM_VK_RIGHT_META** is **0x08**.
TextEvent.DOM_VK_CAPS_LOCK
The value of the constant **TextEvent.DOM_VK_CAPS_LOCK** is **0x09**.
TextEvent.DOM_VK_DELETE
The value of the constant **TextEvent.DOM_VK_DELETE** is **0x0A**.
TextEvent.DOM_VK_END
The value of the constant **TextEvent.DOM_VK_END** is **0x0B**.
TextEvent.DOM_VK_ENTER
The value of the constant **TextEvent.DOM_VK_ENTER** is **0x0C**.
TextEvent.DOM_VK_ESCAPE
The value of the constant **TextEvent.DOM_VK_ESCAPE** is **0x0D**.
TextEvent.DOM_VK_HOME
The value of the constant **TextEvent.DOM_VK_HOME** is **0x0E**.
TextEvent.DOM_VK_INSERT
The value of the constant **TextEvent.DOM_VK_INSERT** is **0x0F**.
TextEvent.DOM_VK_NUM_LOCK
The value of the constant **TextEvent.DOM_VK_NUM_LOCK** is **0x10**.
TextEvent.DOM_VK_PAUSE
The value of the constant **TextEvent.DOM_VK_PAUSE** is **0x11**.
TextEvent.DOM_VK_PRINTSCREEN
The value of the constant **TextEvent.DOM_VK_PRINTSCREEN** is **0x12**.
TextEvent.DOM_VK_SCROLL_LOCK
The value of the constant **TextEvent.DOM_VK_SCROLL_LOCK** is **0x13**.
TextEvent.DOM_VK_LEFT
The value of the constant **TextEvent.DOM_VK_LEFT** is **0x14**.
TextEvent.DOM_VK_RIGHT
The value of the constant **TextEvent.DOM_VK_RIGHT** is **0x15**.
TextEvent.DOM_VK_UP
The value of the constant **TextEvent.DOM_VK_UP** is **0x16**.
TextEvent.DOM_VK_DOWN
The value of the constant **TextEvent.DOM_VK_DOWN** is **0x17**.
TextEvent.DOM_VK_PAGE_DOWN
The value of the constant **TextEvent.DOM_VK_PAGE_DOWN** is **0x18**.
TextEvent.DOM_VK_PAGE_UP
The value of the constant **TextEvent.DOM_VK_PAGE_UP** is **0x19**.
TextEvent.DOM_VK_F1
The value of the constant **TextEvent.DOM_VK_F1** is **0x1A**.
TextEvent.DOM_VK_F2
The value of the constant **TextEvent.DOM_VK_F2** is **0x1B**.
TextEvent.DOM_VK_F3
The value of the constant **TextEvent.DOM_VK_F3** is **0x1C**.
TextEvent.DOM_VK_F4
The value of the constant **TextEvent.DOM_VK_F4** is **0x1D**.
TextEvent.DOM_VK_F5
The value of the constant **TextEvent.DOM_VK_F5** is **0x1E**.

TextEvent.DOM_VK_F6
The value of the constant **TextEvent.DOM_VK_F6** is **0x1F**.
TextEvent.DOM_VK_F7
The value of the constant **TextEvent.DOM_VK_F7** is **0x20**.
TextEvent.DOM_VK_F8
The value of the constant **TextEvent.DOM_VK_F8** is **0x21**.
TextEvent.DOM_VK_F9
The value of the constant **TextEvent.DOM_VK_F9** is **0x22**.
TextEvent.DOM_VK_F10
The value of the constant **TextEvent.DOM_VK_F10** is **0x23**.
TextEvent.DOM_VK_F11
The value of the constant **TextEvent.DOM_VK_F11** is **0x24**.
TextEvent.DOM_VK_F12
The value of the constant **TextEvent.DOM_VK_F12** is **0x25**.
TextEvent.DOM_VK_F13
The value of the constant **TextEvent.DOM_VK_F13** is **0x26**.
TextEvent.DOM_VK_F14
The value of the constant **TextEvent.DOM_VK_F14** is **0x27**.
TextEvent.DOM_VK_F15
The value of the constant **TextEvent.DOM_VK_F15** is **0x28**.
TextEvent.DOM_VK_F16
The value of the constant **TextEvent.DOM_VK_F16** is **0x29**.
TextEvent.DOM_VK_F17
The value of the constant **TextEvent.DOM_VK_F17** is **0x2A**.
TextEvent.DOM_VK_F18
The value of the constant **TextEvent.DOM_VK_F18** is **0x2B**.
TextEvent.DOM_VK_F19
The value of the constant **TextEvent.DOM_VK_F19** is **0x2C**.
TextEvent.DOM_VK_F20
The value of the constant **TextEvent.DOM_VK_F20** is **0x2D**.
TextEvent.DOM_VK_F21
The value of the constant **TextEvent.DOM_VK_F21** is **0x2E**.
TextEvent.DOM_VK_F22
The value of the constant **TextEvent.DOM_VK_F22** is **0x2F**.
TextEvent.DOM_VK_F23
The value of the constant **TextEvent.DOM_VK_F23** is **0x30**.
TextEvent.DOM_VK_F24
The value of the constant **TextEvent.DOM_VK_F24** is **0x31**.

Objects that implement the **TextEvent** interface:

Objects that implement the **TextEvent** interface have all properties and functions of the **UIEvent** interface as well as the properties and functions defined below.

Properties of objects that implement the **TextEvent** interface:

outputString
This property is a **String**.
keyVal
This property is a **Number**.

virtKeyVal
This property is a **Number**.
visibleOutputGenerated
This property is a **Boolean**.
numPad
This property is a **Boolean**.

Functions of objects that implement the **TextEvent** interface:

checkModifier(modifier)
This function returns a **Boolean**.
The **modifier** parameter is a **Number**.
initTextEvent(typeArg, canBubbleArg, cancelableArg, viewArg, detailArg, outputStringArg, keyValArg, virtKeyValArg, visibleOutputGeneratedArg, numPadArg)
This function has no return value.
The **typeArg** parameter is a **String**.
The **canBubbleArg** parameter is a **Boolean**.
The **cancelableArg** parameter is a **Boolean**.
The **viewArg** parameter is an object that implements the **AbstractView** interface.
The **detailArg** parameter is a **Number**.
The **outputStringArg** parameter is a **String**.
The **keyValArg** parameter is a **Number**.
The **virtKeyValArg** parameter is a **Number**.
The **visibleOutputGeneratedArg** parameter is a **Boolean**.
The **numPadArg** parameter is a **Boolean**.
initModifier(modifier, value)
This function has no return value.
The **modifier** parameter is a **Number**.
The **value** parameter is a **Boolean**.

Properties of the **MutationEvent** Constructor function:

MutationEvent.MODIFICATION
The value of the constant **MutationEvent.MODIFICATION** is **1**.
MutationEvent.ADDITION
The value of the constant **MutationEvent.ADDITION** is **2**.
MutationEvent.REMOVAL
The value of the constant **MutationEvent.REMOVAL** is **3**.

Objects that implement the **MutationEvent** interface:

Objects that implement the **MutationEvent** interface have all properties and functions of the **Event** interface as well as the properties and functions defined below.

Properties of objects that implement the **MutationEvent** interface:

relatedNode
This read-only property is an object that implements the **Node** interface.
prevValue
This read-only property is a **String**.
newValue
This read-only property is a **String**.
attrName
This read-only property is a **String**.

attrChange

This read-only property is a **Number**.

Functions of objects that implement the **MutationEvent** interface:

initMutationEvent(typeArg, canBubbleArg, cancelableArg, relatedNodeArg, prevValueArg, newValueArg, attrNameArg, attrChangeArg)

This function has no return value.

The **typeArg** parameter is a **String**.

The **canBubbleArg** parameter is a **Boolean**.

The **cancelableArg** parameter is a **Boolean**.

The **relatedNodeArg** parameter is an object that implements the **Node** interface.

The **prevValueArg** parameter is a **String**.

The **newValueArg** parameter is a **String**.

The **attrNameArg** parameter is a **String**.

The **attrChangeArg** parameter is a **Number**.

Glossary

Editors:
Arnaud Le Hors, W3C
Robert S. Sutor, IBM Research (for DOM Level 1)

Several of the following term definitions have been borrowed or modified from similar definitions in other W3C or standards documents. See the links within the definitions for more information.

ancestor
An *ancestor* node of any node A is any node above A in a tree model of a document, where "above" means "toward the root."

child
A *child* is an immediate descendant node of a node.

descendant
A *descendant* node of any node A is any node below A in a tree model of a document, where "below" means "away from the root."

document element
There is only one document element in a `Document`. This element node is a child of the `Document` node. See *Well-Formed XML Documents* in XML [XML 1.0].

document order
There is an ordering, *document order*, defined on all the nodes in the document corresponding to the order in which the first character of the XML representation of each node occurs in the XML representation of the document after expansion of general entities. Thus, the *document element* [p.67] node will be the first node. Element nodes occur before their children. Thus, document order orders element nodes in order of the occurrence of their start-tag in the XML (after expansion of entities). The attribute nodes of an element occur after the element and before its children. The relative order of attribute nodes is implementation-dependent.

DOM Level 0
The term "DOM Level 0" refers to a mix (not formally specified) of HTML document functionalities offered by Netscape Navigator version 3.0 and Microsoft Internet Explorer version 3.0. In some cases, attributes or methods have been included for reasons of backward compatibility with "DOM Level 0".

sibling
Two nodes are *siblings* if they have the same parent node.

tokenized
The description given to various information items (for example, attribute values of various types, but not including the StringType CDATA) after having been processed by the XML processor. The process includes stripping leading and trailing white space, and replacing multiple space characters by one. See the definition of tokenized type.

well-formed document
A document is *well-formed* if it is tag valid and entities are limited to single elements (i.e., single sub-trees).

XML name
See *XML name* in the XML specification ([XML 1.0]).

References

For the latest version of any W3C specification please consult the list of W3C Technical Reports available at http://www.w3.org/TR.

F.1: Normative references

DOM Level 3 Core
W3C (World Wide Web Consortium) Document Object Model Level 3 Core Specification, January 2002. Available at http://www.w3.org/TR/DOM-Level-3-Core

DOM Level 2 Events
W3C (World Wide Web Consortium) Document Object Model Level 2 Events Specification, November 2000. Available at http://www.w3.org/TR/2000/REC-DOM-Level-2-Events-20001113

DOM Level 2 Views
W3C (World Wide Web Consortium) Document Object Model Level 2 Views Specification, November 2000. Available at http://www.w3.org/TR/2000/REC-DOM-Level-2-Views-20001113

ECMAScript
ISO (International Organization for Standardization). ISO/IEC 16262:1998. ECMAScript Language Specification. Available from ECMA (European Computer Manufacturers Association) at http://www.ecma.ch/ecma1/STAND/ECMA-262.HTM

Java
Sun Microsystems Inc. The Java Language Specification, James Gosling, Bill Joy, and Guy Steele, September 1996. Available at http://java.sun.com/docs/books/jls

OMGIDL
OMG (Object Management Group) IDL (Interface Definition Language) defined in The Common Object Request Broker: Architecture and Specification, version 2.3.1, October 1999. Available from http://www.omg.org

F.2: Informative references

XML 1.0
W3C (World Wide Web Consortium) Extensible Markup Language (XML) 1.0, October 2000. Available at http://www.w3.org/TR/2000/REC-xml-20001006

F.2: Informative references

Index

addEventListener
addGroupedEventListener
ADDITION
altKey
ancestor 9, 10, 67
AT_TARGET
attrChange
attrName
bubbles
BUBBLING_PHASE
button
cancelable
canTrigger
CAPTURING_PHASE
checkModifier
child 36, 67
clientX
clientY
createEvent
createEventGroup
ctrlKey
currentTarget
CustomEvent
descendant 10, 67
detail
dispatchEvent
document element
document order
DocumentEvent
DOM Level 0 9, 23, 25, 39, 67
DOM Level 2 Events 9, 23, 25, 36, 39, 69
DOM Level 2 Views 23, 69
DOM Level 3 Core 9, 23, 25, 29, 36, 39, 69
DOM_VK_CAPS_LOCK
DOM_VK_DELETE
DOM_VK_DOWN
DOM_VK_END
DOM_VK_ENTER
DOM_VK_ESCAPE
DOM_VK_F1
DOM_VK_F10
DOM_VK_F11
DOM_VK_F12
DOM_VK_F13
DOM_VK_F14
DOM_VK_F15
DOM_VK_F16
DOM_VK_F17
DOM_VK_F18
DOM_VK_F19
DOM_VK_F2
DOM_VK_F20
DOM_VK_F21
DOM_VK_F22
DOM_VK_F23
DOM_VK_F24
DOM_VK_F3
DOM_VK_F4
DOM_VK_F5
DOM_VK_F6
DOM_VK_F7
DOM_VK_F8
DOM_VK_F9
DOM_VK_HOME
DOM_VK_INSERT
DOM_VK_LEFT
DOM_VK_LEFT_ALT
DOM_VK_LEFT_CONTROL
DOM_VK_LEFT_META
DOM_VK_LEFT_SHIFT
DOM_VK_NUM_LOCK

DOM_VK_PAGE_DOWN DOM_VK_PAGE_UP DOM_VK_PAUSE
DOM_VK_PRINTSCREEN DOM_VK_RIGHT DOM_VK_RIGHT_ALT
DOM_VK_RIGHT_CONTROL DOM_VK_RIGHT_META DOM_VK_RIGHT_SHIFT
DOM_VK_SCROLL_LOCK DOM_VK_UNDEFINED DOM_VK_UP

ECMAScript Event EventException
EventGroup EventListener eventPhase
EventTarget

handleEvent

initEvent initModifier initMouseEvent
initMutationEvent initTextEvent initUIEvent
isRegisteredHere isSameEventGroup

Java

keyVal

metaKey MODIFICATION MouseEvent
MutationEvent

newValue numPad

OMGIDL outputString

preventDefault prevValue

relatedNode relatedTarget REMOVAL
removeEventListener removeGroupedEventListener

screenX
screenY
setCurrentTarget
setEventPhase
shiftKey
sibling 10, 67
stopPropagation

target
TextEvent
timeStamp
tokenized
type

UIEvent
UNSPECIFIED_EVENT_TYPE_ERR

view
virtKeyVal
visibleOutputGenerated

well-formed document

XML 1.0 67, 67, 69
XML name 19, 19, 67