Abstract

This specification defines the Document Object Model Level 2 Events, a platform- and language-neutral interface that gives to programs and scripts a generic event system. The Document Object Model Level 2 Events builds on the Document Object Model Level 2 Core and on Document Object Model Level 2 Views.

Status of this document

This is a W3C Proposed Recommendation for review by W3C members and other interested parties. W3C Advisory Committee Members are invited to send formal comments, visible only to the W3C Team, to dom-review@w3.org until October 25, 2000.

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/
Publication as a Proposed Recommendation does not imply endorsement by the W3C membership. This is still a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to cite W3C Proposed Recommendations as other than "work in progress."

This document has been produced as part of the [W3C DOM Activity](http://www.w3.org/DOM). The authors of this document are the DOM WG members. Different modules of the Document Object Model have different editors.

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

### Table of contents

- Expanded Table of Contents ........................................... .3
- Copyright Notice .......................................................... .5
- I. Document Object Model Events ...................................... .9
- Appendix A: IDL Definitions ............................................ 31
- Appendix B: Java Language Binding ................................. 35
- Appendix C: ECMA Script Language Binding ...................... 39
- Appendix D: Acknowledgements ....................................... 43
- Glossary ................................................................. 45
- References ............................................................... 47
- Index ........................................................................... 49
### Expanded Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expanded Table of Contents</td>
<td>3</td>
</tr>
<tr>
<td>Copyright Notice</td>
<td>5</td>
</tr>
<tr>
<td>W3C Document Copyright Notice and License</td>
<td>5</td>
</tr>
<tr>
<td>W3C Software Copyright Notice and License</td>
<td>6</td>
</tr>
<tr>
<td>1. Document Object Model Events</td>
<td>9</td>
</tr>
<tr>
<td>1.1. Overview of the DOM Level 2 Event Model</td>
<td>9</td>
</tr>
<tr>
<td>1.1.1. Terminology</td>
<td>9</td>
</tr>
<tr>
<td>1.2. Description of event flow</td>
<td>10</td>
</tr>
<tr>
<td>1.2.1. Basic event flow</td>
<td>10</td>
</tr>
<tr>
<td>1.2.2. Event capture</td>
<td>10</td>
</tr>
<tr>
<td>1.2.3. Event bubbling</td>
<td>11</td>
</tr>
<tr>
<td>1.2.4. Event cancelation</td>
<td>11</td>
</tr>
<tr>
<td>1.3. Event listener registration</td>
<td>12</td>
</tr>
<tr>
<td>1.3.1. Event registration interfaces</td>
<td>12</td>
</tr>
<tr>
<td>1.3.2. Interaction with HTML 4.0 event listeners</td>
<td>14</td>
</tr>
<tr>
<td>1.4. Event interface</td>
<td>15</td>
</tr>
<tr>
<td>1.5. DocumentEvent interface</td>
<td>18</td>
</tr>
<tr>
<td>1.6. Event set definitions</td>
<td>19</td>
</tr>
<tr>
<td>1.6.1. User Interface event types</td>
<td>19</td>
</tr>
<tr>
<td>1.6.2. Mouse event types</td>
<td>21</td>
</tr>
<tr>
<td>1.6.3. Key events</td>
<td>25</td>
</tr>
<tr>
<td>1.6.4. Mutation event types</td>
<td>25</td>
</tr>
<tr>
<td>1.6.5. HTML event types</td>
<td>29</td>
</tr>
<tr>
<td>Appendix A: IDL Definitions</td>
<td>31</td>
</tr>
<tr>
<td>Appendix B: Java Language Binding</td>
<td>35</td>
</tr>
<tr>
<td>Appendix C: ECMA Script Language Binding</td>
<td>39</td>
</tr>
<tr>
<td>Appendix D: Acknowledgements</td>
<td>43</td>
</tr>
<tr>
<td>D.1. Production Systems</td>
<td>43</td>
</tr>
<tr>
<td>Glossary</td>
<td>45</td>
</tr>
<tr>
<td>References</td>
<td>47</td>
</tr>
<tr>
<td>I. Normative references</td>
<td>47</td>
</tr>
<tr>
<td>Index</td>
<td>49</td>
</tr>
</tbody>
</table>
Copyright Notice

Copyright © 2000 [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 binding, the pragma prefix can no longer be 'w3c.org'; in the case of the Java 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 © 1994-2000 [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](http://www.w3.org/Consortium/Legal/copyright-software-19980720)

**Copyright © 1994-2000 [World Wide Web Consortium](http://www.w3.org/Consortium) ([Massachusetts Institute of Technology](http://www.mit.edu), [Institut National de Recherche en Informatique et en Automatique](http://www.inria.fr)[Keio University]. All Rights Reserved.**

[http://www.w3.org/Consortium/Legal/](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.
1. Document Object Model Events

Editors
Tom Pixley, Netscape Communications Corporation

1.1. Overview of the DOM Level 2 Event Model

The DOM Level 2 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 sets of events for user interface control and document mutation notifications, including defined contextual information for each of these event sets.

The second goal of the event model is to provide a common subset of the current event systems used in 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 compliant event sets 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 consumer can use the hasFeature of the DOMImplementation interface to determine whether the Event Model has been implemented by a DOM implementation. The feature string for the Event Model is "Events" and the version is "2.0". The existence within an implementation of each of the individual event sets can also be queried using the hasFeature method. Each event set describes its own feature string in the event set 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 before being handled by the event’s target.

Bubbling
The process by which an event propagates upward through its ancestors 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 level or centrally from an EventTarget higher in the document tree.

1.2.1. Basic event flow

Each event has an EventTarget toward which the event is directed by the DOM implementation. This EventTarget is specified in the Event’s target attribute. When the event reaches the target, any event listeners registered on the EventTarget are triggered. Although all EventListeners 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 on the EventTarget. If neither event capture or event bubbling are in use for that particular event, the event flow process will complete after all listeners have been triggered. If event capture or event bubbling is in use, the event flow will be modified as described in the sections below.

Any exceptions thrown inside an EventListener 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 EventListeners 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 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 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 being registered on an EventTarget 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 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.
If the capturing `EventListener` wishes to prevent further processing of the event from occurring it may call the `stopPropagation` method of the `Event` 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` of the capturing `EventTarget`. It does not allow interception of events targeted to the capturer’s `ancestors`, its `siblings`, or its sibling’s `descendants`.

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`.

### 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` 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`. Any `EventListener` s registered as capturers will not be triggered during this phase. The chain of `EventTarget`s 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` interface. If any `EventListener` calls this method, all additional `EventListeners` on the current `EventTarget` 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 active 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.

Cancellation is accomplished by calling the `Event`’s `preventDefault` method. If one or more `EventListener` 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.

1.3. 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 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);
};
```

Methods

addEventListener

This method allows the registration of event listeners on the event target. If an EventListener is added to an EventTarget while it is processing an event, it 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 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

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

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
  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
  UNSPECIFIED_EVENT_TYPE_ERR: Raised if the
  Event’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.

removeEventListener
This method allows the removal of event listeners from the event target. If an
EventListener is removed from an EventTarget while it is processing an
event, it will not be triggered by the current actions. EventListener can never be
invoked after being removed.

Calling removeEventListener with arguments which do not identify any currently
registered EventListener on the EventTarget has no effect.

Parameters

type of type DOMString
  Specifies the event type of the EventListener being removed.

listener of type EventListener
  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, one with capture and one 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

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` 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.

IDL Definition

```java
// 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.15]

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.14] behaves in the same manner as any other `EventListener` 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. Event interface

**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**

```idl
// 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].
1.4. Event interface

BUDDLING_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.14] 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.45].

Methods

initEvent
The initEvent method is used to initialize the value of an Event created through the DocumentEvent[p.18] 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.45]. 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 prevent further propagation of an event during event flow. If this method is called by any [EventListener][p.14] the event will cease propagating 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.17] as specified in their method descriptions.

IDL Definition

```idl
// 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’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.

### 1.5. DocumentEvent interface

**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**

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

**Methods**

**createEvent**

**Parameters**

**eventType** of type DOMString

The eventType parameter specifies the type of Event 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 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 Events 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.

**Return Value**

Event The newly created Event
1.6. Event set definitions

The DOM Level 2 Event Model allows a DOM implementation to support multiple sets of events. The model has been designed to allow addition of new event sets as is required. The DOM will not attempt to define all possible events. For purposes of interoperability, the DOM will define a set of user interface events including lower level device dependent events, a set of UI logical events, and a set 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 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.

1.6.1. User Interface event types

The User Interface event set is composed of events listed in HTML 4.0 and additional events which are supported in DOM Level 0 browsers.

A DOM consumer can use the hasFeature of the DOMImplementation interface to determine whether the User Interface event set has been implemented by a DOM implementation. The feature string for this event set is "UIEvents" and the version is "2.0". This string is also used with the createEvent method.

Interface UIEvent (introduced in DOM Level 2)

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

IDL Definition

```idl
// 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, 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.18] 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.15]’s AbstractView.

detailArg of type long

Specifies the Event[p.15]’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 an 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)

1.6.2. Mouse event types

The Mouse event set is composed of events listed in HTML 4.0 and additional events which are supported in [DOM Level 0][p.45] browsers. This event set is specifically designed for use with mouse input devices.

A DOM consumer can use the hasFeature of the DOMImplementation interface to determine whether the User Interface event set has been implemented by a DOM implementation. The feature string for this event set is "MouseEvents" and the version is "2.0". This string is also used with the createEvent method.

**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.19] 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**

```javascript
// 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,
```

21
### 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.
1.6.2. Mouse event types

**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` 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`'s `AbstractView`.

- **detailArg** of type `long`
  Specifies the `Event`’s mouse click count.

- **screenXArg** of type `long`
  Specifies the `Event`’s screen x coordinate

- **screenYArg** of type `long`
  Specifies the `Event`’s screen y coordinate

- **clientXArg** of type `long`
  Specifies the `Event`’s client x coordinate

- **clientYArg** of type `long`
  Specifies the `Event`’s client y coordinate

- **ctrlKeyArg** of type `boolean`
  Specifies whether or not control key was depressed during the `Event`.

- **altKeyArg** of type `boolean`
  Specifies whether or not alt key was depressed during the `Event`.
shiftKeyArg of type boolean
Specifies whether or not shift key was depressed during the Event[p.15].

metaKeyArg of type boolean
Specifies whether or not meta key was depressed during the Event[p.15].

buttonArg of type unsigned short
Specifies the Event[p.15]'s mouse button.

relatedTargetArg of type EventTarget[p.12]
Specifies the Event[p.15]'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

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

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

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

The DOM Level 2 Event specification does not provide a key event set. An event set designed for use with keyboard input devices will be included in a later version of the DOM specification.

1.6.4. Mutation event types

The mutation event set 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 consumer can use the hasFeature of the DOMImplementation interface to determine whether the mutation event set has been implemented by a DOM implementation. The feature string for this event set is "MutationEvents" and the version is "2.0". This string is also used with the createEvent method.

Interface MutationEvent (introduced in DOM Level 2)

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

```javascript
// 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);
}
```

**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 DOMCharDataModified events.

- **prevValue** of type DOMString, readonly
  prevValue indicates the previous value of the Attr node in DOMAttrModified events, and of the CharacterData node in DOMCharDataModified 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 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’s related Node.

prevValueArg of type DOMString
Specifies the Event’s prevValue attribute.

newValueArg of type DOMString
Specifies the Event’s newValue attribute.

attrNameArg of type DOMString
Specifies the Event’s attrName 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 accommodate multiple changes which occur either simultaneously or in rapid succession. The target of this event is the lowest common parent of the changes which have taken place.
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 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 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
1.6.5. HTML event types

The HTML event set is composed of events listed in HTML 4.0 and additional events which are supported in DOM Level 0 browsers.

A DOM consumer can use the hasFeature of the DOMImplementation interface to determine whether the HTML event set has been implemented by a DOM implementation. The feature string for this event set is "HTMLEvents" and the version is "2.0". This string is also used with the createEvent method.

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 a document, 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.
1.6.5. HTML event types

- 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: 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: 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
Appendix A: IDL Definitions

This appendix contains the complete OMG IDL [OMGIDL] for the Level 2 Document Object Model Events definitions.

The IDL files are also available as:
http://www.w3.org/TR/2000/PR-DOM-Level-2-Events-20000927/idl.zip

**events.idl:**

```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;

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

31
// 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);
};

// 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,
events.idl:

```idl

in long screenXArg,
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 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);

};

#endif // _EVENTS_IDL_
```
events.idl:
Appendix B: Java Language Binding

This appendix contains the complete Java bindings for the Level 2 Document Object Model Events.

The Java files are also available as http://www.w3.org/TR/2000/PR-DOM-Level-2-Events-20000927/java-binding.zip

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

```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:

```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;
}
```

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

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

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

package org.w3c.dom.events;
import org.w3c.dom.DOMException;

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

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);
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);
}


Appendix C: ECMA Script Language Binding

This appendix contains the complete ECMA Script (ECMScript) binding for the Level 2 Document Object Model Events definitions.

Note: Exceptions handling is only supported by ECMAScript implementation compliant with the Standard ECMA-262 3rd. Edition (ECMScript).

Object EventTarget
The EventTarget object has the following methods:

addEventListener(type, listener, useCapture)
This method has no return value.
The type parameter is of type String.
The listener parameter is of type EventListener.
The useCapture parameter is of type boolean.

removeEventListener(type, listener, useCapture)
This method has no return value.
The type parameter is of type String.
The listener parameter is of type EventListener.
The useCapture parameter is of type boolean.

dispatchEvent(evt)
This method returns a boolean.
The evt parameter is of type Event.
This method can raise a EventException.

Object EventListener
This is an ECMAScript function reference. This method returns a void. The parameter is of type Event.

Class Event
The Event class has the following constants:

Event.CAPTURING_PHASE
This constant is of type short and its value is 1.

Event.AT_TARGET
This constant is of type short and its value is 2.

Event.BUBBLING_PHASE
This constant is of type short and its value is 3.

Object Event
The Event object has the following properties:

type
This read-only property is of type String.

target
This read-only property is of type EventTarget.

currentTarget
This read-only property is of type EventTarget.

eventPhase
This read-only property is of type short.
bubbles
   This read-only property is of type boolean.
cancelable
   This read-only property is of type boolean.
timeStamp
   This read-only property is of type Date.
The Event object has the following methods:
stopPropagation()
   This method has no return value.
preventDefault()
   This method has no return value.
initEvent(eventTypeArg, canBubbleArg, cancelableArg)
   This method has no return value.
   The eventTypeArg parameter is of type String.
   The canBubbleArg parameter is of type boolean.
   The cancelableArg parameter is of type boolean.

Class EventException
   The EventException class has the following constants:
   EventException.UNSPECIFIED_EVENT_TYPE_ERR
      This constant is of type short and its value is 0.

Exception EventException
   The EventException object has the following properties:
   code
      This property is of type unsigned short.

Object DocumentEvent
   The DocumentEvent object has the following methods:
createEvent(eventType)
   This method returns a Event.
   The eventType parameter is of type String.
   This method can raise a DOMException.

Object UIEvent
   UIEvent has all the properties and methods of Event as well as the properties and methods defined below.
   The UIEvent object has the following properties:
view
   This read-only property is of type AbstractView.
detail
   This read-only property is of type long.

The UIEvent object has the following methods:
initUIEvent(typeArg, canBubbleArg, cancelableArg, viewArg, detailArg)
   This method has no return value.
   The typeArg parameter is of type String.
   The canBubbleArg parameter is of type boolean.
   The cancelableArg parameter is of type boolean.
   The viewArg parameter is of type AbstractView.
   The detailArg parameter is of type long.
Object **MouseEvent**

**MouseEvent** has the all the properties and methods of **UIEvent** as well as the properties and methods defined below.

The **MouseEvent** object has the following properties:

- `screenX`
  This read-only property is of type **long**.

- `screenY`
  This read-only property is of type **long**.

- `clientX`
  This read-only property is of type **long**.

- `clientY`
  This read-only property is of type **long**.

- `ctrlKey`
  This read-only property is of type **boolean**.

- `shiftKey`
  This read-only property is of type **boolean**.

- `altKey`
  This read-only property is of type **boolean**.

- `metaKey`
  This read-only property is of type **boolean**.

- `button`
  This read-only property is of type **short**.

- `relatedTarget`
  This read-only property is of type **EventTarget**.

The **MouseEvent** object has the following methods:

- `initMouseEvent(typeArg, canBubbleArg, cancelableArg, viewArg, detailArg, screenXArg, screenYArg, clientXArg, clientYArg, ctrlKeyArg, altKeyArg, shiftKeyArg, metaKeyArg, buttonArg, relatedTargetArg)`
  This method has no return value.

  The `typeArg` parameter is of type **String**.

  The `canBubbleArg` parameter is of type **boolean**.

  The `cancelableArg` parameter is of type **boolean**.

  The `viewArg` parameter is of type **AbstractView**.

  The `detailArg` parameter is of type **long**.

  The `screenXArg` parameter is of type **long**.

  The `screenYArg` parameter is of type **long**.

  The `clientXArg` parameter is of type **long**.

  The `clientYArg` parameter is of type **long**.

  The `ctrlKeyArg` parameter is of type **boolean**.

  The `altKeyArg` parameter is of type **boolean**.

  The `shiftKeyArg` parameter is of type **boolean**.

  The `metaKeyArg` parameter is of type **boolean**.

  The `buttonArg` parameter is of type **short**.

  The `relatedTargetArg` parameter is of type **EventTarget**.

Class **MutationEvent**
The `MutationEvent` class has the following constants:

- **MutationEvent.MODIFICATION**
  This constant is of type `short` and its value is 1.

- **MutationEvent.ADDITION**
  This constant is of type `short` and its value is 2.

- **MutationEvent.REMOVAL**
  This constant is of type `short` and its value is 3.

Object `MutationEvent`

`MutationEvent` has the all the properties and methods of `Event` as well as the properties and methods defined below.

The `MutationEvent` object has the following properties:

- **relatedNode**
  This read-only property is of type `Node`.

- **prevValue**
  This read-only property is of type `String`.

- **newValue**
  This read-only property is of type `String`.

- **attrName**
  This read-only property is of type `String`.

- **attrChange**
  This read-only property is of type `short`.

The `MutationEvent` object has the following methods:

- **initMutationEvent(typeArg, canBubbleArg, cancelableArg, relatedNodeArg, prevValueArg, newValueArg, attrNameArg)**
  This method has no return value.
  
  - The `typeArg` parameter is of type `String`.
  - The `canBubbleArg` parameter is of type `boolean`.
  - The `cancelableArg` parameter is of type `boolean`.
  - The `relatedNodeArg` parameter is of type `Node`.
  - The `prevValueArg` parameter is of type `String`.
  - The `newValueArg` parameter is of type `String`.
  - The `attrNameArg` parameter is of type `String`.

The following example will add an ECMA Script based EventListener to the Node `exampleNode`:

```javascript
// Given the Node 'exampleNode'

// Define the EventListener function
function clickHandler(evt) {
  // Function contents
}

// The following line will add a non-capturing 'click' listener
// to 'exampleNode'.
exampleNode.addEventListener("click", clickHandler, false);
```
Appendix D: Acknowledgements

Many people contributed to this specification, including members of the DOM Working Group and the DOM Interest Group. We especially thank the following:

Lauren Wood (SoftQuad Software Inc., chair), Andrew Watson (Object Management Group), Andy Heninger (IBM), Arnaud Le Hors (W3C and IBM), Ben Chang (Oracle), Bill Smith (Sun), Bill Shea (Merrill Lynch), Bob Sutor (IBM), Chris Lovett (Microsoft), Chris Wilson (Microsoft), David Brownell (Sun), David Singer (IBM), Don Park (invited), Eric Vasilik (Microsoft), Gavin Nicol (INSO), Ian Jacobs (W3C), James Clark (invited), James Davidson (Sun), Jared Sorensen (Novell), Joe Kesselman (IBM), Joe Lapp (webMethods), Joe Marini (Macromedia), Johnny Stenback (Netscape), Jonathan Marsh (Microsoft), Jonathan Robie (Texcel Research and Software AG), Kim Adamson-Sharpe (SoftQuad Software Inc.), Laurence Cable (Sun), Mark Davis (IBM), Mark Scardina (Oracle), Martin Dürst (W3C), Mick Goulish (Software AG), Mike Champion (Arbortext and Software AG), Miles Sabin (Cromwell Media), Patti Lutsky (Arbortext), Paul Grosso (Arbortext), Peter Sharpe (SoftQuad Software Inc.), Phil Karlton (Netscape), Philippe Le Hégaret (W3C, W3C team contact), Ramesh Lekshminarayanan (Merrill Lynch), Ray Whitmer (iMall, Excite@Home and Netscape), Rich Rollman (Microsoft), Rick Gessner (Netscape), Scott Isacae (Microsoft), Sharon Adler (INSO), Steve Byrne (JavaSoft), Tim Bray (invited), Tom Pixley (Netscape), Vidur Apparao (Netscape), Vinod Anupam (Lucent).

Thanks to all those who have helped to improve this specification by sending suggestions and corrections.

D.1: Production Systems

This specification was written in XML. The HTML, OMG IDL, Java and ECMA Script bindings were all produced automatically.

Thanks to Joe English, author of Cost which was used as the basis for producing DOM Level 1. Thanks also to Gavin Nicol, who wrote the scripts which run on top of cost. Arnaud Le Hors and Philippe Le Hégaret maintained the scripts.

For DOM Level 2, we used Xerces as the basis DOM implementation and wish to thank the authors. Philippe Le Hégaret and Arnaud Le Hors wrote the Java programs which are the DOM application.

Thanks also to Jan Kärrman, author of html2ps which we use in creating the PostScript version of the specification.
D.1: Production Systems
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 "above" means "toward the root."

**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 and only 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.

**XML name**
See [XML name](#) in the XML specification [XML].
Glossary
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 2 Core

ECMAScript

Java

OMGIDL

DOM Level 2 Views

XML
F.1: Normative references
Index

addEventListener  ADDITION  altKey
ancestor  9, 10, 45  AT_TARGET  attrChange
attrName

bubbles  BUBBLING_PHASE  button

cancelable  CAPTURING_PHASE  child  25, 45
clientX  clientY  createEvent
ctrlKey  currentTarget

descendant  10, 45  detail  dispatchEvent
DocumentEvent  DOM Level 0  9, 19, 21, 29, 45  DOM Level 2 Core
DOM Level 2 Views

ECMAScript  Event  EventException
EventListener  eventPhase  EventTarget

handleEvent

initEvent  initMouseEvent  initMutationEvent
initUIEvent

Java

metaKey  MODIFICATION  MouseEvent
MutationEvent