Touch Events - Level 2

Draft Community Group Report 02 September 2016

Latest editor's draft:
http://w3c.github.io/touch-events/touchevents.html
Editors:
Doug Schepers, W3C
Sangwhan Moon, Opera Software ASA
Matt Brubeck, Mozilla
Arthur Barstow, Invited Expert
Rick Byers, Google
Patrick H. Lauke, The Paciello Group
Repository:
We are on Github.
File a bug.
Commit history.
Mailing list.

Abstract

The Touch Events specification defines a set of low-level events that represent one or more points of contact with a touch-sensitive surface, and changes of those points with respect to the surface and any DOM elements displayed upon it (e.g. for touch screens) or associated with it (e.g. for drawing tablets without displays). It also addresses pen-tablet devices, such as drawing tablets, with consideration toward stylus capabilities.

Status of This Document

This specification was published by the Touch Events Community Group. It is not a W3C Standard nor is it on the W3C Standards Track. Please note that under the W3C Community Contributor License Agreement (CLA) there is a limited opt-out and other conditions apply. Learn more about W3C Community and Business Groups.
By publishing this Recommendation, W3C expects that the functionality specified in this Touch Interface Recommendation will not be affected by changes to HTML5 or Web IDL as those specifications proceed to Recommendation. The WG has completed and approved this specification's Test Suite and created an Implementation Report that shows that two or more independent implementations pass each test. This version of the specification includes fixes and improvements to Level 1, and incorporates the features previously published as Touch Event Extentions.
If you wish to make comments regarding this document, please send them to public-touchevents@w3.org (subscribe, archives).

1. Introduction

This section is non-normative.
User Agents that run on terminals which provide touch input to use web applications typically use interpreted mouse events to allow users to access interactive web applications. However, these interpreted events, being normalized data based on the physical touch input, tend to have limitations on delivering the intended user experience. Additionally, it is not possible to handle concurrent input regardless of device capability, due to constraints of mouse events: both system level limitations and legacy compatibility.
Meanwhile, native applications are capable of handling both cases with the provided system APIs.
The Touch Events specification provides a solution to this problem by specifying interfaces to allow web applications to directly handle touch events, and multiple touch points for capable devices.

2. Conformance

As well as sections marked as non-normative, all authoring guidelines, diagrams, examples, and notes in this specification are non-normative. Everything else in this specification is normative.
This specification defines conformance criteria that apply to a single product: the user agent that implements the interfaces that it contains.

WindowProxy is defined in [*HTML5*].

WebIDL Conformance

The IDL blocks in this specification are conforming IDL fragments as defined by the WebIDL specification [*WEBIDL*]. A conforming user agent must also be a conforming ECMAScript implementation of this IDL fragments in this specification, with the following exception:

- section 4.4.6 of Web IDL requires that IDL attributes are reflected as accessor properties on interface prototype objects. Instead of this, the user agent may reflect IDL attributes as data properties on the platform objects that implement the relevant interface. These data properties must have the same behavior when getting and setting as would be exhibited when invoking the getter and setter of the accessor properties on the platform object.

Note: Both ways of reflecting IDL attributes allow for simply getting and setting the property on the platform object to work. For example, given a Touch object aTouch, evaluating aTouch.target would return the EventTarget for the Touch object. If the user agent implements IDL attributes as accessor properties, then the property access invokes the getter which returns the EventTarget. If the user agent implements IDL attributes as data properties on the platform object with the same behavior as would be found with the accessor properties, then the object would appear to have an own property named target whose value is an EventTarget object, and the property access would return this value.

3. Touch Interface

This interface describes an individual touch point for a touch event. Touch objects are immutable; after one is created, its attributes must not change.

```
dictionary TouchInit {
    required long        identifier;
    required EventTarget target;
             double      clientX = 0;
             double      clientY = 0;
             double      screenX = 0;
             double      screenY = 0;
             double      pageX = 0;
             double      pageY = 0;
             float       radiusX = 0;
             float       radiusY = 0;
             float       rotationAngle = 0;
             float       force = 0;
};

[Constructor(TouchInit touchInitDict)]
interface Touch {
    readonly attribute long        identifier;
    readonly attribute EventTarget target;
    readonly attribute double      screenX;
    readonly attribute double      screenY;
    readonly attribute double      clientX;
    readonly attribute double      clientY;
    readonly attribute double      pageX;
    readonly attribute double      pageY;
    readonly attribute float       radiusX;
    readonly attribute float       radiusY;
    readonly attribute float       rotationAngle;
    readonly attribute float       force;
};
```

identifier

An identification number for each touch point.

When a touch point becomes active, it must be assigned an identifier that is distinct from any other active touch point. While the touch point remains active, all events that refer to it must assign it the same identifier.

target

The EventTarget on which the touch point started when it was first placed on the surface, even if the touch point has since moved outside the interactive area of that element.

screenX

The horizontal coordinate of point relative to the screen in pixels

screenY

The vertical coordinate of point relative to the screen in pixels

clientX

The horizontal coordinate of point relative to the viewport in pixels, excluding any scroll offset
clientY
The vertical coordinate of point relative to the viewport in pixels, excluding any scroll offset
pageX
The horizontal coordinate of point relative to the viewport in pixels, including any scroll offset
pageY
The vertical coordinate of point relative to the viewport in pixels, including any scroll offset
radiusX
The radius of the ellipse which most closely circumscribes the touching area (e.g. finger, stylus) along the axis indicated by rotationAngle, in CSS pixels (as defined by [*CSS-VALUES*]) of the same scale as screenX; 0 if no value is known. The value must not be negative.
radiusY
The radius of the ellipse which most closely circumscribes the touching area (e.g. finger, stylus) along the axis perpendicular to that indicated by rotationAngle, in CSS pixels (as defined by [*CSS-VALUES*]) of the same scale as screenY; 0 if no value is known. The value must not be negative.
rotationAngle
The angle (in degrees) that the ellipse described by radiusX and radiusY is rotated clockwise about its center; 0 if no value is known. The value must be greater than or equal to 0 and less than 90.
If the ellipse described by radiusX and radiusY is circular, then rotationAngle has no effect. The user agent may use 0 as the value in this case, or it may use any other value in the allowed range. (For example, the user agent may use the rotationAngle value from the previous touch event, to avoid sudden changes.)
force
A relative value of pressure applied, in the range 0 to 1, where 0 is no pressure, and 1 is the highest level of pressure the touch device is capable of sensing; 0 if no value is known. In environments where force is known, the absolute pressure represented by the force attribute, and the sensitivity in levels of pressure, may vary.

4. TouchList Interface

This interface defines a list of individual points of contact for a touch event. TouchList objects are immutable; after one is created, its contents must not change.
A TouchList object's *supported property indices* ([*WEBIDL*]) are the numbers in the range 0 to one less than the length of the list.

```
interface TouchList {
    readonly attribute unsigned long length;
    getter Touch? item(unsigned long index);
};
```

length
Returns the number of Touch objects in the list
item
Returns the Touch at the specified index in the list or null if the index is not less than the length of the list.

5. TouchEvent Interface

This interface defines the touchstart, touchend, touchmove, and touchcancel event types. TouchEvent objects are immutable; after one is created and initialized, its attributes must not change. TouchEvent inherits from the UIEvent interface defined in [*DOM-LEVEL-3-EVENTS*].
The TouchEventInit dictionary is used by the TouchEvent interface's constructor to provide a mechanism by which to construct untrusted (synthetic) touch events. It inherits from the EventModifierInit dictionary defined in [*DOM-LEVEL-3-EVENTS*]. The steps for constructing an event are defined in [*DOM4*]. See the example for sample code demonstrating how to fire an untrusted touch event.

```
dictionary TouchEventInit : EventModifierInit {
    sequence<Touch> touches = [];
    sequence<Touch> targetTouches = [];
    sequence<Touch> changedTouches = [];
};

[Constructor(DOMString type, optional TouchEventInit eventInitDict)]
interface TouchEvent : UIEvent {
    readonly attribute TouchList touches;
    readonly attribute TouchList targetTouches;
    readonly attribute TouchList changedTouches;
```

```
    readonly attribute boolean   altKey;
    readonly attribute boolean   metaKey;
    readonly attribute boolean   ctrlKey;
    readonly attribute boolean   shiftKey;
};
```

touches
A list of Touch objects for every point of contact currently touching the surface.
targetTouches
A list of Touch objects for every point of contact that is touching the surface *and* started on the element that is the target of the current event.
changedTouches
A list of Touch objects for every point of contact which contributed to the event.
For the touchstart event this must be a list of the touch points that just became active with the current event. For the touchmove event this must be a list of the touch points that have moved since the last event. For the touchend and touchcancel events this must be a list of the touch points that have just been removed from the surface, with the last known coordinates of the touch points before they were removed.
altKey
true if the alt (Alternate) key modifier is activated; otherwise false
metaKey
true if the meta (Meta) key modifier is activated; otherwise false. On some platforms this attribute may map to a differently-named key modifier.
ctrlKey
true if the ctrl (Control) key modifier is activated; otherwise false
shiftKey
true if the shift (Shift) key modifier is activated; otherwise false

Note
Some user agents implement an initTouchEvent method as part of the TouchEvent interface. When this method is available, scripts can use it to initialize the properties of a TouchEvent object, including its TouchList properties (which can be initialized with values returned from Document.createTouchList). The initTouchEvent method is not standardized because the signature varies between user agents in completely incompatible ways. It is superseded by the TouchEvent constructor.

5.1 TouchEvent Implementer's Note

This section is non-normative.
Note
User agents should ensure that all Touch objects available from a given TouchEvent are all associated to the same document that the TouchEvent was dispatched to. To implement this, user agents should maintain a notion of the current *touch-active* document. On first touch, this is set to the target document where the touch was created. When all active touch points are released, the *touch-active* document is cleared. All TouchEvents are dispatched to the current *touch-active* document, and each Touch object it contains refers only to DOM elements (and co-ordinates) in that document. If a touch starts entirely outside the currently *touch-active* document, then it is ignored entirely.

5.2 Usage Examples

This section is non-normative.
The examples below demonstrate the relations between the different TouchList members defined in a TouchEvent.

5.2.1 touches and targetTouches of a TouchEvent

This example demonstrates the utility and relations between the touches and targetTouches members defined in the TouchEvent interface. The following code will generate different output based on the number of touch points on the touchable element and the document:
Example 1

```
<div id='touchable'>This element is touchable.</div>

<script>
document.getElementById('touchable').addEventListener('touchstart', function(ev)
{

    if (ev.touches.item(0) == ev.targetTouches.item(0))
    {
        /**
```

```
         * If the first touch on the surface is also targeting the
         * "touchable" element, the code below should execute.
         * Since targetTouches is a subset of touches which covers the
         * entire surface, TouchEvent.touches >= TouchEvents.targetTouches
         * is always true.
         */

        document.write('Hello Touch Events!');
    }

    if (ev.touches.length == ev.targetTouches.length)
    {
        /**
         * If all of the active touch points are on the "touchable"
         * element, the length properties should be the same.
         */

        document.write('All points are on target element')
    }

    if (ev.touches.length > 1)
    {
        /**
         * On a single touch input device, there can only be one point
         * of contact on the surface, so the following code can only
         * execute when the terminal supports multiple touches.
         */

        document.write('Hello Multiple Touch!');
    }

}, false);
</script>
```

5.2.2 changedTouches of a TouchEvent

This example demonstrates the utility of changedTouches and it's relation with the other TouchList members of the TouchEvent interface. The code is a example which triggers whenever a touch point is removed from the defined touchable element:

Example 2

```
<div id='touchable'>This element is touchable.</div>

<script>
document.getElementById('touchable').addEventListener('touchend', function(ev) {

    /**
     * Example output when three touch points are on the surface,
     * two of them being on the "touchable" element and one point
     * in the "touchable" element is lifted from the surface:
     *
     * Touch points removed: 1
     * Touch points left on element: 1
     * Touch points left on document: 2
     */

    document.write('Touch points removed: ' + ev.changedTouches.length);
    document.write('Touch points left on element: ' + ev.targetTouches.length);
    document.write('Touch points left on document: ' + ev.touches.length);

}, false);
</script>
```

5.2.3 Firing a synthetic TouchEvent from script

This example demonstrates how to create and fire a TouchEvent from script.
Example 3

```
if (Touch.length < 1 || TouchEvent.length < 1)
  throw "TouchEvent constructors not supported";

var touch = new Touch({
    identifier: 42,
    target: document.body,
    clientX: 200,
    clientY: 200,
    screenX: 300,
    screenY: 300,
    pageX: 200,
    pageY: 200,
    radiusX: 5,
    radiusY: 5
});

var touchEvent = new TouchEvent("touchstart", {
    cancelable: true,
    bubbles: true,
    composed: true,
    touches: [touch],
    targetTouches: [touch],
    changedTouches: [touch]
});

document.body.dispatchEvent(touchEvent);
```

5.3 List of TouchEvent types

This section is non-normative.
The following table provides a summary of the TouchEvent event types defined in this specification. All events should accomplish the bubbling phase. All events should be composed [*WHATWG-DOM*] events.

Event Type	Sync / Async	Bubbling phase	Composed	Trusted proximal event target types	DOM interface	Cancelable	Default Action
touchstart	Sync	Yes	Yes	Document, Element	TouchEvent	Varies	undefined
touchend	Sync	Yes	Yes	Document, Element	TouchEvent	Varies	Varies: user agents may dispatch mouse and click events
touchmove	Sync	Yes	Yes	Document, Element	TouchEvent	Varies	undefined
touchcancel	Sync	Yes	Yes	Document, Element	TouchEvent	No	none

5.4 Cancelability of touch events

Canceling a touch event can prevent or otherwise interrupt scrolling (which could be happening in parallel with script execution). For maximum scroll performance, a user agent may not wait for each touch event associated with the scroll to be processed to see if it will be canceled. In such cases the user agent should generate touch events whose cancelable property is false, indicating that preventDefault cannot be used to prevent or interrupt scrolling. Otherwise cancelable will be true.
In particular, a user agent may generate only uncancelable touch events when it observes that there are no non-passive listeners for the event.

5.5 The touchstart event

A user agent must dispatch this event type to indicate when the user places a touch point on the touch surface.

The target of this event must be an Element. If the touch point is within a frame, the event should be dispatched to an element in the child browsing context of that frame.
If this event is canceled, it should prevent any default actions caused by any touch events associated with the same active touch point, including mouse events or scrolling.

5.6 The touchend event

A user agent must dispatch this event type to indicate when the user removes a touch point from the touch surface, also including cases where the touch point physically leaves the touch surface, such as being dragged off of the screen.
The target of this event must be the same Element on which the touch point started when it was first placed on the surface, even if the touch point has since moved outside the interactive area of the target element.
The touch point or points that were removed must be included in the changedTouches attribute of the TouchEvent, and must not be included in the touches and targetTouches attributes.
If this event is canceled, any sequence of touch events that includes this event must not be interpreted as a click.

5.7 The touchmove event

A user agent must dispatch this event type to indicate when the user moves a touch point along the touch surface.
The target of this event must be the same Element on which the touch point started when it was first placed on the surface, even if the touch point has since moved outside the interactive area of the target element.
Note that the rate at which the user agent sends touchmove events is implementation-defined, and may depend on hardware capabilities and other implementation details.
A user agent should suppress the default action caused by any touchmove event until at least one touchmove event associated with the same active touch point is not canceled. Whether the default action is suppressed for touchmove events after at least one touchmove event associated with the same active touch point is not canceled is implementation dependent.

5.8 The touchcancel event

A user agent must dispatch this event type to indicate when a touch point has been disrupted in an implementation-specific manner, such as a synchronous event or action originating from the UA canceling the touch, or the touch point leaving the document window into a non-document area which is capable of handling user interactions (e.g. the UA's native user interface, or an area of the document which is managed by a plug-in). A user agent may also dispatch this event type when the user places more touch points on the touch surface than the device or implementation is configured to store, in which case the earliest Touch object in the TouchList should be removed.
The target of this event must be the same Element on which the touch point started when it was first placed on the surface, even if the touch point has since moved outside the interactive area of the target element.
The touch point or points that were removed must be included in the changedTouches attribute of the TouchEvent, and must not be included in the touches and targetTouches attributes.

6. Retargeting

The following section describes retargeting steps, defined in [*WHATWG-DOM*].
Touch object has an associated unadjustedTarget (null or EventTarget). Unless stated otherwise it is null.
TouchEvent's retargeting steps, given a touchEvent, must run these steps:
1. For each Touch touch in touchEvent's touches, targetTouches, and changedTouches members:
A Set touch's unadjustedTarget to touch's target if touch's unadjustedTarget is null.
B Set touch's target to the result of invoking retargeting touch's unadjustedTarget againt touchEvent's target.
C

7. Extensions to the GlobalEventHandlers interface

The following section describes extensions to the existing GlobalEventHandlers interface, defined in [*HTML5*], to facilitate the event handler registration.

```
partial interface GlobalEventHandlers {
    attribute EventHandler ontouchstart;
    attribute EventHandler ontouchend;
    attribute EventHandler ontouchmove;
    attribute EventHandler ontouchcancel;
};
```

ontouchstart
The event handler IDL attribute (see [*HTML5*]) for the touchstart event type.
ontouchend
The event handler IDL attribute (see [*HTML5*]) for the touchend event type.

ontouchmove
The event handler IDL attribute (see [*HTML5*]) for the touchmove event type.
ontouchcancel
The event handler IDL attribute (see [*HTML5*]) for the touchcancel event type.

8. Interaction with Mouse Events and click

The user agent may dispatch both touch events and (for compatibility with web content not designed for touch) mouse events [*DOM-LEVEL-2-EVENTS*] in response to the same user input. If the user agent dispatches both touch events and mouse events in response to a single user action, then the touchstart event type must be dispatched before any mouse event types for that action. If touchstart, touchmove, or touchend are canceled, the user agent should not dispatch any mouse event that would be a consequential result of the prevented touch event.
Note
If a Web application can process touch events, it can cancel the events, and no corresponding mouse events would need to be dispatched by the user agent. If the Web application is not specifically written for touch input devices, it will react to the subsequent mouse events instead.
Note
User agents will typically dispatch mouse and click events only for single-finger activation gestures (like tap and long press). Gestures involving movement of the touch point or multi-touch interactions – with two or more active touch points – will usually only generate touch events.
If the user agent interprets a sequence of touch events as a tap gesture, then it should dispatch mousemove, mousedown, mouseup, and click events (in that order) at the location of the touchend event for the corresponding touch input. If the contents of the document have changed during processing of the touch events, then the user agent may dispatch the mouse events to a different target than the touch events.
The default actions and ordering of any further touch and mouse events are implementation-defined, except as specified elsewhere.
Note
The activation of an element (e.g., in some implementations, a tap) would typically produce the following event sequence (though this may vary slightly, depending on specific user agent behavior):
1. touchstart
2. Zero or more touchmove events, depending on movement of the finger
3. touchend
4. mousemove (for compatibility with legacy mouse-specific code)
5. mousedown
6. mouseup
7. click

If, however, either the touchstart, touchmove or touchend event has been canceled during this interaction, no mouse or click events will be fired, and the resulting sequence of events would simply be:
1. touchstart
2. Zero or more touchmove events, depending on movement of the finger
3. touchend

Note
Even if a user agent supports Touch Events, this does not necessarily mean that a touchscreen is the only input mechanism available to users. Particularly in the case of touch-enabled laptops, or traditional "touch only" devices (such as phones and tablets) with paired external input devices, users may use the touchscreen in conjunction with a trackpad, mouse or keyboard. For this reason, developers should avoid binding event listeners with "either touch or mouse/keyboard" conditional code, as this results in sites/application that become touch-exclusive, preventing users from being able to use any other input mechanism.
Example 4

```
// conditional "touch OR mouse/keyboard" event binding
// DON'T DO THIS, as it makes interactions touch-exclusive
// on devices that have both touch and mouse/keyboard

if ('ontouchstart' in window) {
  // set up event listeners for touch
  target.addEventListener('touchend', ...);
  ...
} else {
  // set up event listeners for mouse/keyboard
  target.addEventListener('click', ...);
  ...
```

```
}
```

Instead, developers should handle different forms of input concurrently.
Example 5

```
// concurrent "touch AND mouse/keyboard" event binding

// set up event listeners for touch
target.addEventListener('touchend', function(e) {
  // prevent compatibility mouse events and click
  e.preventDefault();
  ...
});
...

// set up event listeners for mouse/keyboard
target.addEventListener('click', ...);
...
```

To avoid processing the same interaction twice for touch (once for the touch event, and once for the compatibility mouse events), developers should make sure to cancel the touch event, suppressing the generation of any further mouse or click events. Alternatively, see the InputDeviceCapabilities API for a way to detect mouse events that were generated as a result of touch events.

9. Glossary

active touch point
A touch point which is currently on the screen and is being tracked by the user agent. The touch point becomes active when the user agent first dispatches a touchstart event indicating its appearance. It ceases to be active after the user agent dispatches a touchend or touchcancel event indicating that the touch point is removed from the surface or no longer tracked.
touch point
The coordinate point at which a pointer (e.g finger or stylus) intersects the target surface of an interface. This may apply to a finger touching a touch-screen, or an digital pen writing on a piece of paper.
canceled event
An event whose default action was prevented by means of preventDefault(), returning false in an event handler, or other means as defined by [*DOM-LEVEL-3-EVENTS*] and [*HTML5*].

10. Issues

This section is non-normative.
The working group maintains a list of open issues in this specification. These issues may be addressed in future revisions of the specification.

A. Legacy Event Initializers

The following features are obsolete and should only be implemented by user agents that require compatibility with legacy software.

A.1 Extensions to the Document Interface

The Document interface [*DOM-LEVEL-3-CORE*] contains methods by which the user can create Touch and TouchList objects.
Note
Document.createTouch is superseded by the Touch constructor.
Document.createTouchList creates a TouchList object consisting of zero or more Touch objects. Document.createTouchList is superseded by passing sequences of Touch objects directly to the TouchEvent constructor.

```
partial interface Document {
    // Deprecated in this specification
    Touch     createTouch(WindowProxy view,
                          EventTarget target,
                          long identifier,
                          double pageX,
                          double pageY,
```

```
                          double screenX,
                          double screenY);
    // Deprecated in this specification
    TouchList createTouchList(Touch... touches);
};
```

B. Acknowledgements

This section is non-normative.
Many thanks to the WebKit engineers for developing the model used as a basis for this spec, Neil Roberts (SitePen) for his summary of WebKit touch events, Peter-Paul Koch (PPK) for his write-ups and suggestions, Robin Berjon for developing the ReSpec.js spec authoring tool, and the WebEvents WG for their many contributions.
Many others have made additional comments as the spec developed, which have led to steady improvements. Among them are Matthew Schinckel, Andrew Grieve, Cathy Chan, Boris Zbarsky and Patrick H. Lauke. If we inadvertently omitted your name, please let me know.
The group acknowledges the following contributors to this specification's test suite: Matt Brubeck, Olli Pettay, Art Barstow, Cathy Chan and Rick Byers.

C. Changes Since Last Publication

This section is non-normative.
This is a summary of the major changes made since the 10 October 2013 Recommendation was published. Full commit history is also available.

- Added force attribute to Touch (commit)
- Added radiusX and radiusY attributes to Touch (commit)
- Added rotationAngle attribute to Touch (commit)
- Upgraded co-ordinates to double type instead of long (commit)
- Update touchmove behavior on preventDefault (commit)
- Clarify effect of canceling touchend event (commit)
- Add constructor for TouchEvent and Touch (commit)
- Added legacy event initializer initTouchEvent (commit)
- Added support for uncancelable touch events and described scroll performance implications.
- Indicate that all events should be "composed".
- Note about avoiding conditional "touch OR mouse/keyboard" event handling
- Added TouchEvent's retargeting steps

D. References

D.1 Normative references

[CSS-VALUES]
Tab Atkins Jr.; Elika Etemad. W3C. *CSS Values and Units Module Level 3*. 29 September 2016. W3C Candidate Recommendation. URL: https://www.w3.org/TR/css-values-3/
[DOM-LEVEL-2-EVENTS]
Tom Pixley. W3C. *Document Object Model (DOM) Level 2 Events Specification*. 13 November 2000. W3C Recommendation. URL: https://www.w3.org/TR/DOM-Level-2-Events/
[DOM-LEVEL-3-CORE]
Arnaud Le Hors; Philippe Le Hégaret; Lauren Wood; Gavin Nicol; Jonathan Robie; Mike Champion; Steven B Byrne et al. W3C. *Document Object Model (DOM) Level 3 Core Specification*. 7 April 2004. W3C Recommendation. URL: https://www.w3.org/TR/DOM-Level-3-Core/
[DOM-LEVEL-3-EVENTS]
Gary Kacmarcik; Travis Leithead. W3C. *UI Events*. 4 August 2016. W3C Working Draft. URL: https://www.w3.org/TR/uievents/
[DOM4]
Anne van Kesteren; Aryeh Gregor; Ms2ger; Alex Russell; Robin Berjon. W3C. *W3C DOM4*. 19 November 2015. W3C Recommendation. URL: https://www.w3.org/TR/dom/
[HTML5]
Ian Hickson; Robin Berjon; Steve Faulkner; Travis Leithead; Erika Doyle Navara; Theresa O'Connor; Silvia Pfeiffer. W3C. *HTML5*. 28 October 2014. W3C Recommendation. URL: https://www.w3.org/TR/html5/
[RFC2119]
S. Bradner. IETF. *Key words for use in RFCs to Indicate Requirement Levels*. March 1997. Best Current Practice. URL: https://tools.ietf.org/html/rfc2119
[WEBIDL]

Cameron McCormack; Boris Zbarsky; Tobie Langel. W3C. *Web IDL*. 15 September 2016. W3C Working Draft. URL: https://www.w3.org/TR/WebIDL-1/
[WHATWG-DOM]
Anne van Kesteren. WHATWG. *DOM Standard*. Living Standard. URL: https://dom.spec.whatwg.org/
↑