**HTML DOM**

# DOM-Attr Object

* Get the value of the onclick attribute of a button element:

let btn = document.getElementsByTagName("BUTTON")[0];  
btn.attributes.getNamedItem("onclick").value;

<button onclick="myFunction()">Try it</button>

// myFunction()

The getNamedItem() method returns the attribute node with the specified name from a NamedNodeMap object.

* Get the name of the first attribute of a <button> element:

<button onclick="myFunction()" class="example">Try it</button>

let a = document.getElementsByTagName("BUTTON")[0]

let x = a.attributes.item(0).nodeName;

// onclick

* Get the number of attributes of a <button> element:

document.getElementsByTagName("BUTTON")[0].attributes.length;

// 2

* Get the name of an attribute:

document.getElementsByTagName("BUTTON")[0].attributes[0].name;

// onclick

* Remove the type attribute from an input button:

<input type="button" value="OK">

<button onclick="myFunction()">Try it</button>

function myFunction() {

var btn = document.getElementsByTagName("INPUT")[0];

btn.attributes.removeNamedItem("type");

}

// input alanına dönüşür. Value ‘si OK olur. Button özelliği kaldırılmış olur.

* Set a H1's class attribute:

.democlass {

color: red; }

const h = document.getElementsByTagName("H1")[0];  
const typ = document.createAttribute("class");  
typ.value = "democlass";  
h.attributes.setNamedItem(typ);;

// H1 in rengi kırmızıya dönüşür. Yeni bir class ekleyip onu h1 ‘e entegre ettik.

* Find out if an attribute has been specified or not:

document.getElementById("demo").attributes[0].specified;

// true

# DOM-Document Object

* Get the currently focused element in the document:

let x = document.activeElement.tagName;

The activeElement property returns the currently focused element in the document.

* document.addEventListener(*event*, *function*, *useCapture*)

document.addEventListener("click", function(){  
  document.getElementById("demo").innerHTML = "Hello World";  
});

document.addEventListener("mouseover", myFunction);  
document.addEventListener("click", someOtherFunction);  
document.addEventListener("mouseout", someOtherFunction);

* Adopt the first <h1> element that appears in an iframe (another document):

const frame = document.getElementsByTagName("IFRAME")[0]  
const h = frame.contentWindow.document.getElementsByTagName("H1")[0];  
const x = document.adoptNode(h);

* Get the baseURI of the document:

const  x = document.baseURI;

**//** https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref\_doc\_baseuri

* Change the background color of the current document:

document.body.style.backgroundColor = "yellow";

**Tip:** The difference between this property and the [document.documentElement](https://www.w3schools.com/jsref/prop_document_documentelement.asp) property, is that the document.body element returns the <body> element, while the document.documentElement returns the <html> element.

Create a <p> element with some text and append it to the document's body:

const x = document.createElement("P"); // Create a <p> element  
const t = document.createTextNode("This is a paragraph."); // Create a text node  
x.appendChild(t); // Append the text to <p>  
document.body.appendChild(x); // Append <p> to <body>

* Open an output stream, add some text, then close the output stream:

document.open();  
document.write("<h1>Hello World</h1>");  
document.close();

* Get the cookies associated with the current document:

const x = document.cookie;

## **Syntax**

Return the cookie property:

document.cookie

Set the cookie property:

document.cookie = newCookie

*newCookie;* **expires=date** - Optional. & **path=path** - Optional.  & **domain=domainname** - Optional. & **secure**- Optional.

document.cookie="username=John Doe; expires=Thu, 18 Dec 2013 12:00:00 UTC; path=/";

* Display the character encoding for this document:

const  x = document.characterSet;

* Create a class attribute, with the value "democlass", and insert it to an <h1> element:

const  h1 = document.getElementsByTagName("H1")[0];// Get the first <h1> element in the document  
const  att = document.createAttribute("class");    // Create a "class" attribute  
att.value = "democlass";               // Set the value of the class attribute  
h1.setAttributeNode(att);            // Add the class attribute to <h1>

Before creating the attribute:

# Hello World

After inserting the attribute:

# Hello World

* Create a <button> element:

const btn = document.createElement("BUTTON");

Create a button with text:

const btn = document.createElement("BUTTON");// Create a <button> element  
btn.innerHTML = "CLICKME";           // Insert text  
document.body.appendChild(btn);       // Append <button> to <body>

## **Syntax**

document.createElement(nodename)

<button onclick="myFunction()">Try it</button>

<script>

function myFunction() {

var para = document.createElement("P");

para.innerText = "This is a paragraph.";

document.body.appendChild(para);

}

</script>

<div id="myDIV">

A DIV element

</div>

<button onclick="myFunction()">Try it</button>

<script>

function myFunction() {

var para = document.createElement("P");

para.innerHTML = "This is a paragraph.";

document.getElementById("myDIV").appendChild(para);

}

</script>

* Simulate a mouseover event:

<script>

function myFunction(event) {

const x = document.createEvent("MouseEvent");

x.initMouseEvent("mouseover", true, true, window, 0, 0, 0, 0, 0, false, false, false, false, 0, null);

document.getElementById("myDiv").dispatchEvent(x);}

</script>

<h1>The createEvent() Method</h1>

<p>The createEvent() method allows you to simulate any event.</p>

<p>In this example, the red div will get a new star every time you mouse over it:</p>

<div onmouseover="this.innerHTML += '\*';" id="myDiv">\*</div>

<button onclick="myFunction(event)">Simulate Mouse Over</button>

The createEvent() method creates an event object.

## **Syntax**

document.createEvent(type)

Required. A String that specifies the type of the event.  
  
Possible values:  
  
AnimationEvent  
ClipboardEvent  
DragEvent  
FocusEvent  
HashChangeEvent  
InputEvent  
KeyboardEvent  
MouseEvent  
PageTransitionEvent  
PopStateEvent  
ProgressEvent  
StorageEvent  
TouchEvent  
TransitionEvent  
UiEvent  
WheelEvent

* Create a text node:

const t = document.createTextNode("Hello World");

document.body.appendChild(t);

const  para = document.createElement("P");             // Create a <p> element  
const  t = document.createTextNode("This is a paragraph."); // Create a text node  
para.appendChild(t);                                  // Append the text to <p>

* Get the doctype name of an HTML document:

const  x = document.doctype.name;

* Return the documentElement of the document, as a node name:

const  x = document.documentElement.nodeName;

* Get the domain name of the server that loaded the document:

const  x = document.domain;

* Show a <video> element in fullscreen mode:

/\* Get the element you want displayed in fullscreen \*/  
const  elem = document.getElementById("myvideo");  
  
/\* Function to open fullscreen mode \*/  
function openFullscreen() {  
  /\* If fullscreen mode is available, show the element in fullscreen \*/  
  if (document.fullscreenEnabled) {  
    /\* Show the element in fullscreen \*/  
    elem.requestFullscreen();  
  }  
}

document.getElementById("demo");

document.getElementsByClassName("example");

document.getElementsByName("fname");

document.getElementsByTagName("LI");

* Get the id of the <head> element of the current document:

<head id="myHead"> ...

document.head.id; // myHead

* Find out how many <script> elements there are in the document:

document.scripts.length;

* Find out how many links there are in the document:

const  x = document.links.length;

* Find out how many <img> elements there are in the document:

const  x = document.images.length;

const  x = document.body. childNodes.length;

* Get the first element in the document with class="example":

document.querySelector(".example");

* Get the first <p> element in the document where the parent is a <div> element.

document.querySelector("div > p");

* Get the first <a> element in the document that has a "target" attribute:

document.querySelector("a[target]");

* Get the first <p> element in the document:

document.querySelector("p");

* Get all elements in the document with class="example":

document.querySelectorAll(".example");

* Remove a "mousemove" event that has been attached with the addEventListener() method:

// Attach an event handler to the document  
document.addEventListener("mousemove", myFunction);  
  
// Remove the event handler from the document  
document.removeEventListener("mousemove", myFunction);

* Write some text directly to the HTML document:

document.write("Hello World!");

# DOM-Element Object

* Append an item in a list:

const node = document.createElement("LI");             // Create a <li> node  
const textnode = document.createTextNode("Water");   // Create a text node  
node.appendChild(textnode);                      // Append the text to <li>  
document.getElementById("myList").appendChild(node);  // Append <li> to <ul> with id="myList"

Before appending:

* Coffee
* Tea

After appending:

* Coffee
* Tea
* Water
* Find out how many attributes a <button> element have:

const  x = document.getElementById("myBtn").attributes.length;

* Remove focus from an <a> element:

document.getElementById("myAnchor").blur();

* Find out how many child elements a <div> element has:

const  x = document.getElementById("myDIV").childElementCount;

* Get a collection of the <body> element's child nodes:

const  c = document.body.childNodes;

#comment  
#text  
P  
#text  
BUTTON  
#text  
P  
#text  
P  
#text  
SCRIPT  
#text

The childNodes property returns a collection of a node's child nodes, as a NodeList object.

The nodes in the collection are sorted as they appear in the source code and can be accessed by index numbers. The index starts at 0.

* Get a collection of the <body> element's children:

const  c = document.body.children;

P  
BUTTON  
P  
SCRIPT

The children property returns a collection of an element's child elements, as an HTMLCollection object.

The elements in the collection are sorted as they appear in the source code and can be accessed by index numbers. The index starts at 0.

* Add the "mystyle" class to a <div> element:

document.getElementById("myDIV").classList.add("mystyle");

* Set the class for a <div> element with id="myDIV":

document.getElementById("myDIV").className = "mystyle";

* Simulate a mouse-click when moving the mouse pointer over a checkbox:

<input type="checkbox" id="myCheck" onmouseover="myFunction()" onclick="alert('clicked')">  
  
<script>  
// On mouse-over, execute myFunction  
function myFunction() {  
  document.getElementById("myCheck").click(); // Click on the checkbox  
}  
</script>

* Get the height and width of a <div> element, including padding:

const elmnt = document.getElementById("myDIV");  
const  txt = "Height with padding: " + elmnt.clientHeight + "px<br>";  
txt += "Width with padding: " + elmnt.clientWidth + "px";

The clientHeight property returns the viewable height of an element in pixels, including padding, but not the border, scrollbar or margin.

elmnt.clientHeight

elmnt.clientWidth

The clientTop property returns the width of the top border of an element, in pixels. This property does not include the element's top padding or top margin.

elmnt.clientTop // border

elmnt.clientLeft //border

* Copy a <li> element from one list to another:

// Get the last <li> element ("Milk") of <ul> with id="myList2"  
const itm = document.getElementById("myList2").lastChild;  
  
// Copy the <li> element and its child nodes  
const  cln = itm.cloneNode(true);  
  
// Append the cloned <li> element to <ul> with id="myList1"  
document.getElementById("myList1").appendChild(cln);

Before cloning:

* Coffee
* Tea
* Water
* Milk

After cloning:

* Coffee
* Tea
* Milk
* Water
* Milk
* Find out if a <span> element is a descendant of a <div> element:

const span = document.getElementById("mySPAN");  
const div = document.getElementById("myDIV").contains(span); // true

* Change the text direction of a <p> element to "right-to-left":

document.getElementById("myP").dir = "rtl";

* Get the HTML content of the first child node of an <ul> element:

document.getElementById("myList").firstChild.innerHTML;

The firstChild property returns the first child node of the specified node, as a Node object. Whitespace inside elements is considered as text, and text is considered as nodes

The difference between this property and [firstElementChild](https://www.w3schools.com/jsref/prop_element_firstelementchild.asp), is that firstChild returns the first child node as an element node, a text node or a comment node (depending on which one's first), while firstElementChild returns the first child node as an element node (ignores text and comment nodes).

<!--  
Whitespace inside elements is considered as text, and text is considered as nodes  
In this example, there is whitespace before <p>, before <span> and after <span>  
Therefore, the first child node of <div> is a #text node, and not the <p> element you expected  
-->

<div id="myDIV">  
  <p>Looks like first child</p>  
  <span>Looks like last Child</span>  
</div>  
<script>  
const x = document.getElementById("myDIV").firstChild.nodeName;  
document.getElementById("demo").innerHTML = x;  
</script>

// #text

However, if we remove the whitespace from the source, there are no #text nodes in <div>, which will make the <p> element the first child node:

<div id="myDIV"><p>First child</p><span>Last Child</span></div>  
<script>  
const x = document.getElementById("myDIV").firstChild.nodeName;  
document.getElementById("demo").innerHTML = x;  
</script>

// P

Get the HTML content of the first child element of an <ul> element:

document.getElementById("myList").firstElementChild.innerHTML;

The firstElementChild property returns the first child element of the specified element.

The difference between this property and [firstChild](https://www.w3schools.com/jsref/prop_node_firstchild.asp), is that firstChild returns the first child node as an element node, a text node or a comment node (depending on which one's first), while firstElementChild returns the first child node as an element node (ignores text and comment nodes).

<div id="myDIV">  
  <p>Looks like first child</p>  
  <span>Looks like last Child</span>  
</div>  
<script>  
const x = document.getElementById("myDIV").firstElementChild.tagName;  
document.getElementById("demo").innerHTML = x;  
</script>

// P

<select id="mySelect" size="4">

<option>Audi</option>

<option>BMW</option>

<option>Saab</option>

<option>Volvo</option>

</select>

<br><br>

...

<script>

function myFunction() {

const x = document.getElementById("mySelect").firstElementChild.text;

document.getElementById("demo").innerHTML = x;

}

</script>

// Audi

* Get the value of the class attribute of an <h1> element:

document.getElementsByTagName("H1")[0].getAttribute("class");

* Get the value of the class attribute node of an <h1> element:

const elmnt = document.getElementsByTagName("H1")[0];  
const attr = elmnt.getAttributeNode("class").value;

* Return the size of an element and its position relative to the viewport:

obj.getBoundingClientRect();

* Find out if a <button> element has an onclick attribute:

document.getElementById("myBtn").hasAttribute("onclick");

* Find out if the <body> element has any attributes:

document.body.hasAttributes()

* Find out if an <ul> element has any child nodes:

document.getElementById("myList").hasChildNodes(); //true or false

* Get the inner text of an element:

document.getElementById("myBtn").innerText;

textContent returns the text content of **all** elements, while innerText returns the content of all elements, **except** for <script> and <style> elements.

innerText will not return the text of elements that are hidden with CSS (textContent will).

<p id="demo">   This element has extra spacing     and contains <span>a span element</span>.</p>  
<script>  
function getInnerText() {  
  alert(document.getElementById("demo").innerText)  
}  
function getHTML() {  
  alert(document.getElementById("demo").innerHTML)  
}  
function getTextContent() {  
  alert(document.getElementById("demo").textContent)  
}  
</script>

innerText returns: "This element has extra spacing and contains a span element."

innerHTML returns: " This element has extra spacing and contains <span>a span element</span>."

textContent returns: " This element has extra spacing and contains a span element."

insertAdjacentText()

insertAdjacentElement()

insertAdjacentHTML()

insertBefore()

isContentEditable

isDefaultNamespace

isEqualNode

isSameNode

isSupported

* Ge the language code of a <p> element:

document.getElementById("myP").lang;

* Get the HTML content of the last child node of an <ul> element:

document.getElementById("myList").lastChild.innerHTML;

* Get the HTML content of the last child element of an <ul> element:

document.getElementById("myList").lastElementChild.innerHTML;

* Get the HTML content of the next sibling of a list item:

document.getElementById("item1").nextSibling.innerHTML;

* Get the HTML content of the next sibling of a list item:

document.getElementById("item1").nextElementSibling.innerHTML;

* Get the node name of a <p> element:

document.getElementById("myP").nodeName;

If the node is an element node, the nodeName property will return the tag name.

If the node is an attribute node, the nodeName property will return the name of the attribute.

For other node types, the nodeName property will return different names for different node types.

Returns the tagname for element nodes, in uppercase

Returns the name of the attribute for attribute nodes

Returns "#text" for text nodes

Returns "#comment" for comment nodes

Returns "#document" for document nodes

* Get the node type of the body element:

document.getElementById("myP").nodeType;

The nodeType property returns the node type, as a number, of the specified node.

If the node is an element node, the nodeType property will return 1.

If the node is an attribute node, the nodeType property will return 2.

If the node is a text node, the nodeType property will return 3.

If the node is a comment node, the nodeType property will return 8.

* Get the node value of the first <button> element in the document:

document.getElementsByTagName("BUTTON")[0].childNodes[0].nodeValue;

Returns *null* for element nodes and document nodes

Returns the value of the attribute for attribute nodes

Returns the content for text nodes

Returns the content for comment nodes

<div id="myDIV">This is a div element.</div>  
<script>  
var x = document.getElementById("myDIV").firstChild;  
var txt = "";  
txt += "The node name: " + x.nodeName + "<br>";  
txt += "The node value: " + x.nodeValue + "<br>";  
txt += "The node type: " + x.nodeType;  
</script>

The node name: #text  
The node value: This is a div element.  
The node type: 3

* Normalize an element:

document.getElementById("demo").normalize();

* Get the height and width of a <div> element, including padding and border:

const elmnt = document.getElementById("myDIV");  
const  txt = "Height with padding and border: " + elmnt.offsetHeight + "px<br>";  
txt += "Width with padding and border: " + elmnt.offsetWidth + "px";

The offsetHeight property returns the viewable height of an element in pixels, including padding, border and scrollbar, but not the margin.

The reason why the "viewable" word is specified, is because if the element's content is taller than the actual height of the element, this property will only return the height that is visible

**Information about this div:**  
Height: 250px  
Width: 400px  
padding: 10px  
margin: 15px  
border: 5px

Height including padding and border: 280px  
Width including padding and border: 429px

*element*.offsetWidth

The offsetLeft property returns the left position (in pixels) relative to the left side the offsetParent element.

the left position, and margin of the element

the left padding, scrollbar and border of the offsetParent element

*object*.offsetLeft

*object*.offsetParent

*object*.offsetTop

document.getElementById("myH1").outerText = "Changed content!";

document.getElementsByTagName("h1").outerHTML = "<h3>Header Changed!</h3>";

* Get the node name of the parent node of a <li> element:

document.getElementById("myLI").parentNode.nodeName;

* Get the node name of the parent element of a <li> element:

document.getElementById("myLI").parentElement.nodeName;

The difference between parentElement and [parentNode](https://www.w3schools.com/jsref/prop_node_parentnode.asp), is that parentElement returns null if the parent node is not an element node:

document.body.parentNode; // Returns the <html> element  
document.body.parentElement; // Returns the <html> element  
  
document.documentElement.parentNode; // Returns the Document node  
document.documentElement.parentElement; // Returns null (<html> does not have a parent ELEMENT node)

* Get the HTML content of the previous sibling of a list item:

document.getElementById("item2").previousSibling.innerHTML;

* Get the HTML content of the previous sibling of a list item:

document.getElementById("item2").previousElementSibling.innerHTML;

* Remove the selected element from the document:

const myobj = document.getElementById("demo");  
myobj.remove();

* Remove the class attribute from an <h1> element:

document.getElementsByTagName("H1")[0].removeAttribute("class");

removeAttribute() yöntemi, belirtilen niteliği bir öğeden kaldırır.

Bu yöntem ile removeAttributeNode() yöntemi arasındaki fark, removeAttributeNode() yönteminin belirtilen Attr nesnesini kaldırması, bu yöntemin ise belirtilen ada sahip özniteliği kaldırmasıdır. Sonuç aynı olacaktır. Ayrıca bu yöntemin bir dönüş değeri yoktur, removeAttributeNode() yöntemi ise kaldırılan özniteliği bir Attr nesnesi olarak döndürür.

*element*.removeAttributeNode(*attributenode*)

*node*.removeChild(*node*)

The removeChild() method removes a specified child node of the specified element.

Returns the removed node as a Node object, or null if the node does not exist.

*node*.replaceChild(*newnode*,*oldnode*)

* Get the entire height and width of an element, including padding:

const elmnt = document.getElementById("content");  
const y = elmnt.scrollHeight;  
const x = elmnt.scrollWidth;

The scrollHeight property returns the entire height of an element in pixels, including padding, but not the border, scrollbar or margin.

The scrollWidth and scrollHeight properties return the entire height and width of an element, including the height and width that is not viewable (because of overflow).

* Get the number of pixels the content of a <div> element is scrolled horizontally and vertically:

const  elmnt = document.getElementById("myDIV");  
const  x = elmnt.scrollLeft;  
const  y = elmnt.scrollTop;

The scrollLeft property sets or returns the number of pixels an element's content is scrolled horizontally.

* Add the class attribute with the value of "democlass" to a <h1> element:

document.getElementsByTagName("H1")[0].setAttribute("class", "democlass");

* Set the class attribute node of a <h1> element:

const  h1 = document.getElementsByTagName("H1")[0]; // Get the first <h1> element in the document  
const  att = document.createAttribute("class"); // Create a "class" attribute  
att.value = "democlass";        // Set the value of the class attribute  
h1.setAttributeNode(att);       // Add the class attribute to <h1>

*element*.style.*property*

HTML DOM reference: [node.childNodes Property](https://www.w3schools.com/jsref/prop_node_childnodes.asp)

HTML DOM reference: [node.firstChild Property](https://www.w3schools.com/jsref/prop_node_firstchild.asp)

HTML DOM reference: [node.lastChild Property](https://www.w3schools.com/jsref/prop_node_lastchild.asp)

HTML DOM reference: [node.nextSibling Property](https://www.w3schools.com/jsref/prop_node_nextsibling.asp)

HTML DOM reference: [node.previousSibling Property](https://www.w3schools.com/jsref/prop_node_previoussibling.asp)

HTML DOM reference: [node.nodeName Property](https://www.w3schools.com/jsref/prop_node_nodename.asp)

HTML DOM reference: [node.parentElement Property](https://www.w3schools.com/jsref/prop_node_parentelement.asp)

## **HTML DOM Events**

## **The MouseEvent Object**

**altKey** Returns whether the "ALT" key was pressed when the mouse event was triggered

**button** Returns which mouse button was pressed when the mouse event was triggered

**buttons** Returns which mouse buttons were pressed when the mouse event was triggered

**clientX**  Returns the horizontal coordinate of the mouse pointer, relative to the current window, when the mouse event was triggered

**clientY** Returns the vertical coordinate of the mouse pointer, relative to the current window, when the mouse event was triggered

**ctrlKey** Returns whether the "CTRL" key was pressed when the mouse event was triggered

**offsetX** Returns the horizontal coordinate of the mouse pointer relative to the position of the edge of the target element

**offsetY** Returns the vertical coordinate of the mouse pointer relative to the position of the edge of the target element

**pageX** Returns the horizontal coordinate of the mouse pointer, relative to the document, when the mouse event was triggered

**pageY** Returns the vertical coordinate of the mouse pointer, relative to the document, when the mouse event was triggered

**screenX** Returns the horizontal coordinate of the mouse pointer, relative to the screen, when an event was triggered

**screenY** Returns the vertical coordinate of the mouse pointer, relative to the screen, when an event was triggered

**shiftKey** Returns whether the "SHIFT" key was pressed when an event was triggered

**which** Returns which mouse button was pressed when the mouse event was triggered

**onclick** The event occurs when the user clicks on an element

**oncontextmenu** The event occurs when the user right-clicks on an element to open a context menu

**ondblclick** The event occurs when the user double-clicks on an element

**onmousedown** The event occurs when the user presses a mouse button over an element

**onmouseenter** The event occurs when the pointer is moved onto an element

**onmouseleave** The event occurs when the pointer is moved out of an element

**onmousemove** The event occurs when the pointer is moving while it is over an element

**onmouseout** The event occurs when a user moves the mouse pointer out of an element, or out of one of its children

**onmouseover** The event occurs when the pointer is moved onto an element, or onto one of its children

**onmouseup** The event occurs when a user releases a mouse button over an element

## **The KeyboardEvent Object**

**onkeydown** The event occurs when the user is pressing a key

**onkeypress** The event occurs when the user presses a key

**onkeyup** The event occurs when the user releases a key

## **The TouchEvent Object**

**ontouchcancel** The event occurs when the touch is interrupted

**ontouchend** The event occurs when a finger is removed from a touch screen

**ontouchmove** The event occurs when a finger is dragged across the screen

**ontouchstart** The event occurs when a finger is placed on a touch screen

## **The AnimationEvent Object**

**animationName** Returns the name of the animation

**elapsedTime** Returns the number of seconds an animation has been running

**pseudoElement** Returns the name of the pseudo-element of the animation

**animationend** The event occurs when a CSS animation has completed

**animationiteration** The event occurs when a CSS animation is repeated

**animationstart** The event occurs when a CSS animation has started

## **The UiEvent Object**

**abort** The event occurs when the loading of a media is aborted

**beforeunload** The event occurs before the document is about to be unloaded

**error** The event occurs when an error occurred during the loading of a media file

**load** The event occurs when an object has loaded

**resize** The event occurs when the document view is resized

**scroll** The event occurs when an element's scrollbar is being scrolled

**select** The event occurs after the user selects some text (for <input> and <textarea>)

**unload** The event occurs once a page has unloaded (for <body>)

## **The FocusEvent Object**

**onblur** The event occurs when an element loses focus

**onfocus** The event occurs when an element gets focus

**onfocusin** The event occurs when an element is about to get focus

**onfocusout** The event occurs when an element is about to lose focus

## **The ClipboardEvent Object**

**oncopy** The event occurs when the user copies the content of an element

**oncut** The event occurs when the user cuts the content of an element

**onpaste** The event occurs when the user pastes some content in an element

## **The DragEvent Object**

**ondrag** The event occurs when an element is being dragged

**ondragend** The event occurs when the user has finished dragging an element

**ondragenter** The event occurs when the dragged element enters the drop target

**ondragleave** The event occurs when the dragged element leaves the drop target

**ondragover** The event occurs when the dragged element is over the drop target

**ondragstart** The event occurs when the user starts to drag an element

**ondrop** The event occurs when the dragged element is dropped on the drop target

## **The HashChangeEvent Object**

**newURL** Returns the URL of the document, after the hash has been changed

**oldURL** Returns the URL of the document, before the hash was changed

**onhashchange** The event occurs when there has been changes to the anchor part of a URL

## **Event Object**

**createEvent**() Creates a new event

**composedPath**() Returns the event's path

**currentTarget** Returns the element whose event listeners triggered the event

**defaultPrevented** Returns whether or not the preventDefault() method was called for the event

**eventPhase** Returns which phase of the event flow is currently being evaluated

**preventDefault**() Cancels the event if it is cancelable, meaning that the default action that belongs to the event will not occur

**stopImmediatePropagation**() Prevents other listeners of the same event from being called

**stopPropagation**() Prevents further propagation of an event during event flow

**target** Returns the element that triggered the event

**timeStamp** Returns the time (in milliseconds relative to the epoch) at which the event was created

**change** The event occurs when the content of a form element, the selection, or the checked state have changed (for <input>, <select>, and <textarea>)

**error** The event occurs when an error occurs while loading an external file

**fullscreenchange** The event occurs when an element is displayed in fullscreen mode

**fullscreenerror** The event occurs when an element can not be displayed in fullscreen mode

**input** The event occurs when an element gets user input

**load** The event occurs when an object has loaded

**message** The event occurs when a message is received through the event source

**progress** The event occurs when the browser is in the process of getting the media data (downloading the media)

**ratechange** The event occurs when the playing speed of the media is changed

**resize** The event occurs when the document view is resized

**reset** The event occurs when a form is reset

**scroll** The event occurs when an element's scrollbar is being scrolled

**search** The event occurs when the user writes something in a search field (for <input="search">)

**select** The event occurs after the user selects some text (for <input> and <textarea>)

**show** The event occurs when a <menu> element is shown as a context menu

**submit** The event occurs when a form is submitted

**toggle** The event occurs when the user opens or closes the <details> element

**waiting** The event occurs when the media has paused but is expected to resume (like when the media pauses to buffer more data)

## **Location Object**

**host** Sets or returns the hostname and port number of a URL

**hostname** Sets or returns the hostname of a URL

**href** Sets or returns the entire URL

**pathname** Sets or returns the path name of a URL

**protocol** Sets or returns the protocol of a URL

## **Navigator Object**

**appCodeName** Returns the code name of the browser

**appName** Returns the name of the browser

**appVersion** Returns the version information of the browser

**cookieEnabled** Determines whether cookies are enabled in the browser

**geolocation** Returns a Geolocation object that can be used to locate the user's position

**language** Returns the language of the browser

**onLine** Determines whether the browser is online

**platform** Returns for which platform the browser is compiled

**product** Returns the engine name of the browser

**userAgent** Returns the user-agent header sent by the browser to the server

## **Console Object**

**assert**() Writes an error message to the console if the assertion is false

**clear**() Clears the console

**count**() Logs the number of times that this particular call to count() has been called

**error**() Outputs an error message to the console

**info**() Outputs an informational message to the console

**log**() Outputs a message to the console

## **Geolocation Object**

**coordinates** Returns the position and altitude of the device on Earth

**position** Returns the position of the concerned device at a given time

**positionError** Returns the reason of an error occurring when using the geolocating device

**positionOptions** Describes an object containing option properties to pass as a parameter of Geolocation.getCurrentPosition() and Geolocation.watchPosition()

## **History Object**

**back**() Loads the previous URL in the history list

**forward**() Loads the next URL in the history list

**go**() Loads a specific URL from the history list

## **Storage Object**

**getItem**(keyname) Returns the value of the specified key name

**setItem**(keyname, value) Adds that key to the storage, or update that key's value if it already exists

**removeItem**(keyname) Removes that key from the storage

**clear**() Empty all key out of the storage

**window**.**localStorage** Allows to save key/value pairs in a web browser. Stores the data with no expiration date

**window**.**sessionStorage** Allows to save key/value pairs in a web browser. Stores the data for one session