**DOM:**  When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.

The **HTML DOM** model is constructed as a tree of **Objects**:

**The HTML DOM is a standard for how to get, change, add, or delete HTML elements.**

-- HTML DOM methods are **actions** you can perform (on HTML Elements)

-- HTML DOM properties are **values** (of HTML Elements) that you can set or change.

The HTML DOM can be accessed with JavaScript (and with other programming languages).In the DOM, all HTML elements are defined as **objects**.

A **property** is a value that you can get or set (like changing the content of an HTML element).

A **method** is an action you can do (like add or deleting an HTML element).

Example

The following example changes the content (the innerHTML) of the <p> element with id="demo":

<html>  
<body>  
<p id="demo"></p>  
<script>  
document.getElementById("demo").innerHTML = "Hello World!";  
</script>  
</body>  
</html>

O/P : Hello World!

**NOTE :** In the example above, getElementById is a **method**, while innerHTML is a **property**.

## The HTML DOM Document

In the HTML DOM object model, the document object represents your web page.

The document object is the owner of all other objects in your web page.

If you want to access objects in an HTML page, you always start with accessing the document object.

Below are some examples of how you can use the document object to access and manipulate HTML.

## Changing HTML Elements

|  |  |
| --- | --- |
| **Method** | **Description** |
| document.getElementById() | Find an element by element id |
| document.getElementsByTagName() | Find elements by tag name |
| document.getElementsByClassName() | Find elements by class name |

## Adding and Deleting Elements

|  |  |
| --- | --- |
| **Method** | **Description** |
| document.createElement() | Create an HTML element |
| document.removeChild() | Remove an HTML element |
| document.appendChild() | Add an HTML element |
| document.replaceChild() | Replace an HTML element |
| document.write(*text*) |  |

Adding Events Handlers

|  |  |
| --- | --- |
| **Method** | **Description** |
| document.getElementById(*id*).onclick=function(){*code*} | Adding event handler code to an onclick  event |

## Changing HTML Style

document.getElementById(*id*).style.*property* = *new style*

The onmouseover and onmouseout events can be used to trigger a function when the user mouses over, or out of, an HTML element.

*<!DOCTYPE html>*

*<html>*

*<body>*

*<div onmouseover="mOver(this)" onmouseout="mOut(this)" style="background-color:#D94A38;width:120px;height:20px;padding:40px;">*

*Mouse Over Me</div>*

*<script>*

*function mOver(obj) {*

*obj.innerHTML = "Thank You"*

*}*

*function mOut(obj) {*

*obj.innerHTML = "Mouse Over Me"*

*}*

*</script>*

*</body>*

*</html>*

## Add an Event Handler to an Element

*<!DOCTYPE html>*

*<html>*

*<body>*

*<p>This example uses the addEventListener() method to attach a click event to a button.</p>*

*<button id="myBtn">Try it</button>*

*<script>*

*document.getElementById("myBtn").addEventListener("click", function(){*

*alert("Hello World!");*

*});*

*</script>*

*Or*

*<script>*

*document.getElementById("myBtn").addEventListener("click", myFunction);*

*function myFunction() {*

*alert ("Hello World!");*

*}*

*</script>*

***Or***

***<script>***

***var x = document.getElementById("myBtn");***

***x.addEventListener("click", myFunction);***

***x.addEventListener("click", someOtherFunction);***

***function myFunction() {***

***alert ("Hello World!");***

***}***

***function someOtherFunction() {***

***alert ("This function was also executed!");***

***}***

# </script>

*</body>*

*</html>*

# JavaScript HTML DOM Navigation

According to the W3C HTML DOM standard, everything in an HTML document is a node:

* The entire document is a document node
* Every HTML element is an element node
* The text inside HTML elements are text nodes
* Every HTML attribute is an attribute node
* All comments are comment nodes



With the HTML DOM, all nodes in the node tree can be accessed by JavaScript.

New nodes can be created, and all nodes can be modified or deleted.

Node Relationships

The nodes in the node tree have a hierarchical relationship to each other.

The terms parent, child, and sibling are used to describe the relationships.

* In a node tree, the top node is called the root (or root node)
* Every node has exactly one parent, except the root (which has no parent)
* A node can have a number of children
* Siblings (brothers or sisters) are nodes with the same parent

Navigating Between Nodes

You can use the following node properties to navigate between nodes with JavaScript:

* parentNode
* childNodes[*nodenumber*]
* firstChild
* lastChild
* nextSibling
* previousSibling

## Creating New HTML Elements (Nodes)

<!DOCTYPE html>

<html>

<body>

<div id="div1">

<p id="p1">This is a paragraph.</p>

<p id="p2">This is another paragraph.</p>

</div>

<script>

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

var node = document.createTextNode("This is new.");

var paragraph = para.appendChild(node);

var element = document.getElementById("div1");

element.appendChild(paragraph );

</script>

</body>

</html>

The addEventListener() method

Add an event listener that fires when a user clicks a button:

document.getElementById("myBtn").addEventListener("click", displayDate);

The addEventListener() method attaches an event handler to the specified element.

The addEventListener() method attaches an event handler to an element without overwriting existing event handlers.

You can add many event handlers to one element.

You can add many event handlers of the same type to one element, i.e two "click" events.

You can add event listeners to any DOM object not only HTML elements. i.e the window object.

The addEventListener() method makes it easier to control how the event reacts to bubbling.

When using the addEventListener() method, the JavaScript is separated from the HTML markup, for better readability and allows you to add event listeners even when you do not control the HTML markup.

You can easily remove an event listener by using the removeEventListener() method.

Syntax

*element*.addEventListener(*event, function, useCapture*);

The first parameter is the type of the event (like "click" or "mousedown").

The second parameter is the function we want to call when the event occurs.

The third parameter is a boolean value specifying whether to use event bubbling or event capturing. This parameter is optional.

Alert "Hello World!" when the user clicks on an element:

*element*.addEventListener("click", function(){ alert("Hello World!"); });

**You can also refer to an external "named" function:**

Alert "Hello World!" when the user clicks on an element:

*element*.addEventListener("click", myFunction);  
  
function myFunction() {  
    alert ("Hello World!");  
}

The addEventListener() method allows you to add many events to the same element, without overwriting existing events:

Example

*element*.addEventListener("click", myFunction);  
*element*.addEventListener("click", mySecondFunction);

You can add events of different types to the same element:

Example

*element*.addEventListener("mouseover", myFunction);  
*element*.addEventListener("click", mySecondFunction);  
*element*.addEventListener("mouseout", myThirdFunction);

Example

Add an event listener that fires when a user resizes the window:

window.addEventListener("resize", function(){  
    document.getElementById("demo").innerHTML = *sometext*;  
});

Passing Parameters

When passing parameter values, use an "anonymous function" that calls the specified function with the parameters:

Example

*element*.addEventListener("click", function(){ myFunction(p1, p2); });

The removeEventListener() method

The removeEventListener() method removes event handlers that have been attached with the addEventListener() method:

Example

*element*.removeEventListener("mousemove", myFunction);

Creating New HTML Elements (Nodes)

To add a new element to the HTML DOM, you must create the element (element node) first, and then append it to an existing element.

<div id="div1">  
<p id="p1">This is a paragraph.</p>  
<p id="p2">This is another paragraph.</p>  
</div>  
  
<script>  
var para = document.createElement("p");  
var node = document.createTextNode("This is new.");  
para.appendChild(node);  
  
var element = document.getElementById("div1");  
element.appendChild(para);  
</script>

Example Explained

This code creates a new <p> element:

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

To add text to the <p> element, you must create a text node first. This code creates a text node:

var node = document.createTextNode("This is a new paragraph.");

Then you must append the text node to the <p> element:

para.appendChild(node);

Finally you must append the new element to an existing element.

This code finds an existing element:

var element = document.getElementById("div1");

This code appends the new element to the existing element:

element.appendChild(para);

Creating new HTML Elements - insertBefore()

The appendChild() method in the previous example, appended the new element as the last child of the parent.

If you don't want that you can use the insertBefore() method:

 Example

<div id="div1">  
<p id="p1">This is a paragraph.</p>  
<p id="p2">This is another paragraph.</p>  
</div>  
  
<script>  
var para = document.createElement("p");  
var node = document.createTextNode("This is new.");  
para.appendChild(node);  
  
var element = document.getElementById("div1");  
var child = document.getElementById("p1");  
element.insertBefore(para,child);  
</script>

Removing Existing HTML Elements

To remove an HTML element, you must know the parent of the element:

Example

<div id="div1">  
<p id="p1">This is a paragraph.</p>  
<p id="p2">This is another paragraph.</p>  
</div>  
  
<script>  
var parent = document.getElementById("div1");  
var child = document.getElementById("p1");  
parent.removeChild(child);  
</script>

[**Try it Yourself »**](http://www.w3schools.com/js/tryit.asp?filename=tryjs_dom_elementremove)

Example Explained

This HTML document contains a <div> element with two child nodes (two <p> elements):

<div id="div1">  
<p id="p1">This is a paragraph.</p>  
<p id="p2">This is another paragraph.</p>  
</div>

Find the element with id="div1":

var parent = document.getElementById("div1");

Find the <p> element with id="p1":

var child = document.getElementById("p1");

Remove the child from the parent:

parent.removeChild(child);

**A node list is a collection of nodes**

HTML DOM Node List

The getElementsByTagName() method returns a **node list**. A node list is an array-like collection of nodes.

The following code selects all <p> nodes in a document:

Example

var x = document.getElementsByTagName("p");

The nodes can be accessed by an index number. To access the second <p> node you can write:

**Note:** The index starts at 0.

HTML DOM Node List Length

The length property defines the number of nodes in a node list:

Example

var myNodelist = document.getElementsByTagName("p");  
document.getElementById("demo").innerHTML = myNodelist.length;

Example explained:

1. Get all <p> elements in a node list
2. Display the length of the node list

The length property is useful when you want to loop through the nodes in a node list:

The Browser Object Model (BOM)

There are no official standards for the **B**rowser **O**bject **M**odel (BOM).

Since modern browsers have implemented (almost) the same methods and properties for JavaScript interactivity, it is often referred to, as methods and properties of the BOM.

The Window Object

The **window** object is supported by all browsers. It represents the browser's window.

All global JavaScript objects, functions, and variables automatically become members of the window object.

Global variables are properties of the window object.

Global functions are methods of the window object.

Even the document object (of the HTML DOM) is a property of the window object:

window.document.getElementById("header");

is the same as:

document.getElementById("header");

<!DOCTYPE html>

<html>

<body>

<p id="demo"></p>

<script>

var w = window.innerWidth

|| document.documentElement.clientWidth

|| document.body.clientWidth;

var h = window.innerHeight

|| document.documentElement.clientHeight

|| document.body.clientHeight;

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

x.innerHTML = "Browser inner window width: " + w + ", height: " + h + ".";

</script>

</body>

</html>

Other Window Methods

Some other methods:

* window.open() - open a new window
* window.close() - close the current window
* window.moveTo() -move the current window
* window.resizeTo() -resize the current window

**The window.screen object contains information about the user's screen.**

Window Screen

The **window.screen** object can be written without the window prefix.

Properties:

* screen.width
* screen.height
* screen.availWidth
* screen.availHeight
* screen.colorDepth
* screen.pixelDepth

Window Screen Width

The screen.width property returns the width of the visitor's screen in pixels.

Example

Display the width of the screen in pixels:

document.getElementById("demo").innerHTML =  
"Screen Width: " + screen.width;

Result will be:

Screen Width: 1600

Window Screen Height

The screen.height property returns the height of the visitor's screen in pixels.

Example

Display the height of the screen in pixels:

document.getElementById("demo").innerHTML =  
"Screen Height: " + screen.height;

Result will be:

Screen Height: 900

Window Screen Available Width

The screen.availWidth property returns the width of the visitor's screen, in pixels, minus interface features like the Windows Taskbar.

Example

Display the available width of the screen in pixels:

document.getElementById("demo").innerHTML =  
"Available Screen Width: " + screen.availWidth;

Result will be:

Available Screen Width: 1600

Window Screen Available Height

The screen.availHeight property returns the height of the visitor's screen, in pixels, minus interface features like the Windows Taskbar.

Example

Display the available height of the screen in pixels:

document.getElementById("demo").innerHTML =  
"Available Screen Height: " + screen.availHeight;

Result will be:

Available Screen Height: 860

Window Screen Color Depth

The screen.colorDepth property returns the number of bits used to display one color.

All modern computers use 24 bit or 32 bit hardware for color resolution:

* 24 bits =      16,777,216 different "True Colors"
* 32 bits = 4,294,967,296 different "Deep Colors"

Older computers used 16 bits: 65,536 different "High Colors" resulution.

Very old computers, and old cell phones used 8 bits: 256 different "VGA colors".

Example

Display the color depth of the screen in bits:

document.getElementById("demo").innerHTML =   
"Screen Color Depth: " + screen.colorDepth;

Result will be:

Screen Color Depth: 24

The #rrggbb (rgb) values used in HTML represents "True Colors" (16,777,216 different colors)

Window Screen Pixel Depth

The screen.pixelDepth property returns the pixel depth of the screen.

Example

Display the pixel depth of the screen in bits:

document.getElementById("demo").innerHTML =  
"Screen Pixel Depth: " + screen.pixelDepth;

Result will be:

Screen Pixel Depth: 24

For modern computers, Color Depth and Pixel Depth are equal.

**The window.location object can be used to get the current page address (URL) and to redirect the browser to a new page.**

Window Location

The **window.location** object can be written without the window prefix.

Some examples:

* window.location.href returns the href (URL) of the current page
* window.location.hostname returns the domain name of the web host
* window.location.pathname returns the path and filename of the current page
* window.location.protocol returns the web protocol used (http:// or https://)
* window.location.assign loads a new document

Window Location Href

The **window.location.href** property returns the URL of the current page.

Example

Display the href (URL) of the current page:

document.getElementById("demo").innerHTML =  
"Page location is " + window.location.href;

Result is:

Page location is http://www.w3schools.com/js/js\_window\_location.asp

Window Location Hostname

The **window.location.hostname** property returns the name of the internet host (of the current page).

Example

Display the name of the host:

document.getElementById("demo").innerHTML =  
"Page hostname is " + window.location.hostname;

Result is:

Page hostname is www.w3schools.com

Window Location Pathname

The **window.location.pathname** property returns the pathname of the current page.

Example

Display the path name of the current URL:

document.getElementById("demo").innerHTML =  
"Page path is " + window.location.pathname;

Result is:

/js/js\_window\_location.asp

Window Location Protocol

The **window.location.protocol** property returns the web protocol of the page.

Example

Display the web protocol:

document.getElementById("demo").innerHTML =  
"Page protocol is " + window.location.protocol;

Result is:

Page protocol is http:

Window Location Assign

The **window.location.assign()** method loads a new document.

Example

Load a new document:

<html>  
<head>  
<script>  
function newDoc() {  
    window.location.assign("http://www.w3schools.com")  
}  
</script>  
</head>  
<body>  
  
<input type="button" value="Load new document" onclick="newDoc()">  
  
</body>  
</html>

**The window.history object contains the browsers history.**

Window History

The **window.history** object can be written without the window prefix.

To protect the privacy of the users, there are limitations to how JavaScript can access this object.

Some methods:

* history.back() - same as clicking back in the browser
* history.forward() - same as clicking forward in the browser

Window History Back

The history.back() method loads the previous URL in the history list.

This is the same as clicking the Back button in the browser.

Example

Create a back button on a page:

<html>  
<head>  
<script>  
function goBack() {  
    window.history.back()  
}  
</script>  
</head>  
<body>  
  
<input type="button" value="Back" onclick="goBack()">  
  
</body>  
</html>

Window History Forward

The history forward() method loads the next URL in the history list.

This is the same as clicking the Forward button in the browser.

Example

Create a forward button on a page:

<html>  
<head>  
<script>  
function goForward() {  
    window.history.forward()  
}  
</script>  
</head>  
<body>  
  
<input type="button" value="Forward" onclick="goForward()">  
  
</body>  
</html>

**The window.navigator object contains information about the visitor's browser.**

**Window Navigator**

The **window.navigator** object can be written without the window prefix.

Some examples:

* navigator.appName
* navigator.appCodeName
* navigator.platform

**Navigator Cookie Enabled**

The property cookieEnabled returns true if cookies are enabled, otherwise false:

**Example**

<p id="demo"></p>  
<script>  
document.getElementById("demo").innerHTML =  
"Cookies Enabled is " + navigator.cookieEnabled;  
</script>

**The Browser Names**

The properties **appName** and **appCodeName** return the name of the browser:

**Example**

<p id="demo"></p>  
  
<script>  
document.getElementById("demo").innerHTML =  
"Name is " + navigator.appName + ". Code name is " + navigator.appCodeName;  
</script>

**Did you know?**  
IE11, Chrome, Firefox, and Safari return appName "Netscape".  
Chrome, Firefox, IE, Safari, and Opera all return appCodeName "Mozilla".

The Browser Engine

The property **product**returns the engine name of the browser:

Example

<p id="demo"></p>  
  
<script>  
document.getElementById("demo").innerHTML = navigator.product;  
</script>

The Browser Version II

The property **userAgent also**returns version information about the browser:

Example

<p id="demo"></p>  
  
<script>  
document.getElementById("demo").innerHTML = navigator.userAgent;  
</script>